Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

_____ С.Т. Князев

«_15_» _декабря_____ 2021 г.

Инжиниринг данных

Учебно-методические материалы по направлению подготовки
09.04.01 Информатика и вычислительная техника
Образовательная программа «Инженерия искусственного интеллекта»

Екатеринбург

2021

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент, канд.техн.наук  Корелин Иван Андреевич

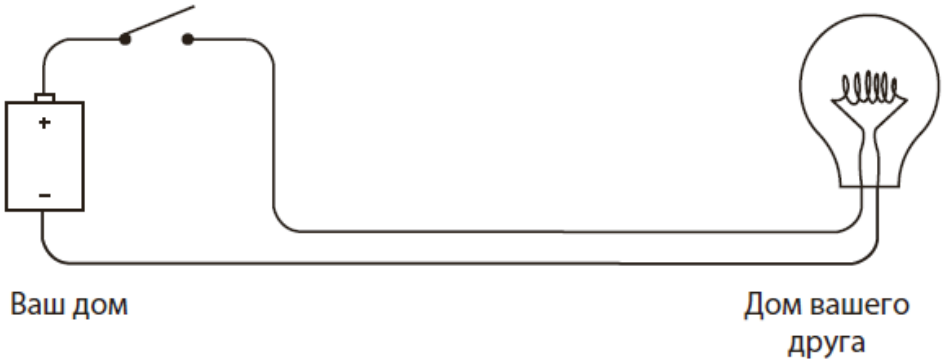
СОДЕРЖАНИЕ

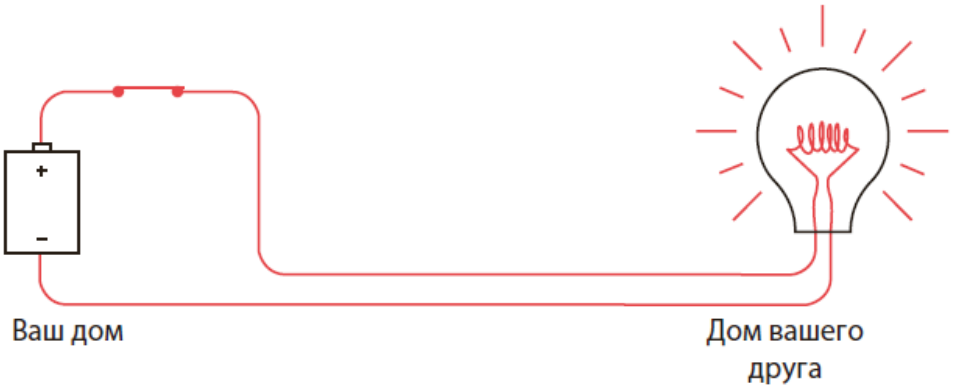
1. Введение в язык Python

1.1. Что такое код?

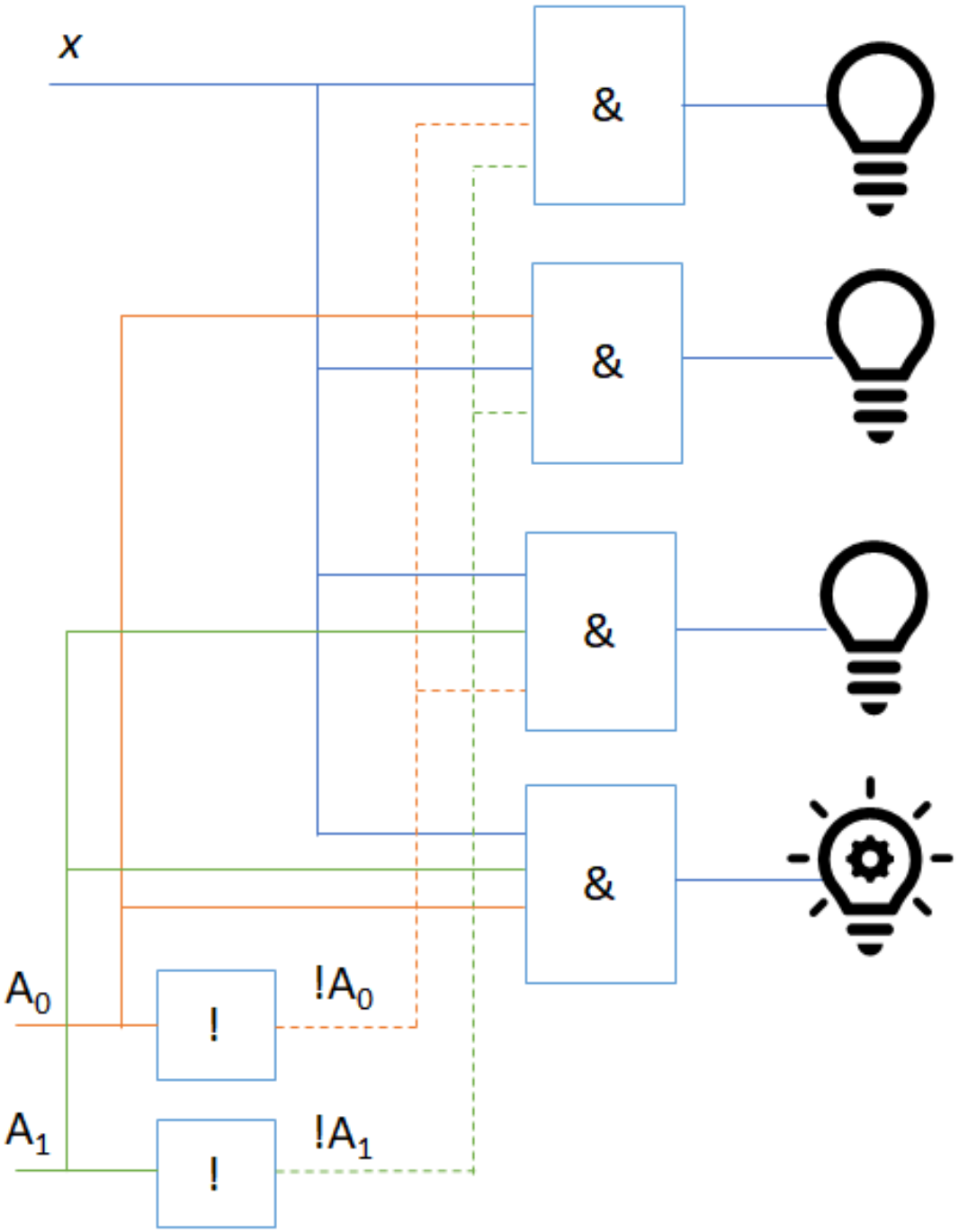
Язык программирования – это инструмент, применимый в конкретной области вычислительной техники.

Прежде чем обсуждать языки программирования для начала рассмотрим основные принципы построения вычислительных машин. Атомарным элементом является понятие ключа. Если вы хотя бы раз собирали цепь, в которой источник и лампа связаны проводами через электрический ключ, то вы понимаете, что у ключа и соответственно привязанной к нему лампе 2 состояния: 1 - работает и 0 - выключен.





Следующим уровнем являются логические элементы. Такие как отрицание (!), И (&), ИЛИ (|). Принципы их построения не сложны, но если вы не знаете и хотите узнать, рассмотрите, например, книгу Чарльза Петцольда "Код. Тайный язык информатики". Эти логические операторы неотъемлемая и важная часть любого языка программирования. Что бы забраться глубже в то как работает компьютер рассмотрим как выглядит такое электронное цифровое устройство, как демультиплексор.



Основное достижение этого прибора заключается в том, что двумя ключами A0 и A1 возможно управлять поведением одной из 4-х лампочек. То есть **код управляет** тем как работает **система**. На таких же принципах построены арифметико-логическое устройство (АЛУ) и процессорное устройство (ПУ). Т.е. машины работают на основе **кода** который описывает как им выполнять вычисления. Но людям комфортнее соотносить команды со **словами**.

1.2. Уровни языков программирования

В зависимости от уровня абстракции от физической организации машины языки программирования делятся на несколько уровней. Самый низкоуровневый - это язык кодов (1 уровень). Поверх кодов организован машинно-ориентированный язык - например язык ассемблера (2).

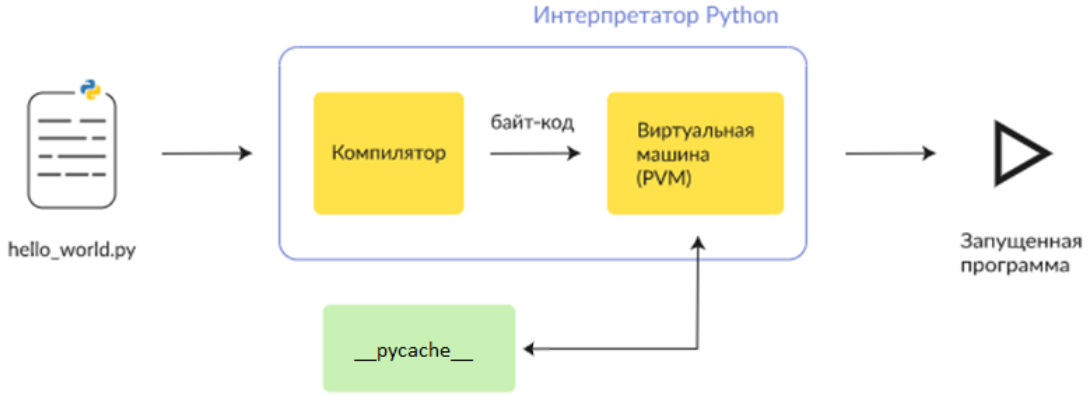
Языки высокого уровня (ЯВУ) чаще всего оперируют уже более понятными человеку понятиями логики, цикличности, подпрограмм (процедур и функций). Но для преобразования из ЯВУ (3 уровень) в более низкоуровневые требуется специальное программное обеспечение - компилятор (или интерпретатор в зависимости от природы языка).

Существуют и ещё более высокоуровневые языки описывающие решение задач в специфической предметно ориентированной области Domain-specific language. Например SQL (Structured Query Language — «язык структурированных запросов»). Или GPSS (General Purpose Simulation System — система моделирования общего назначения).

1.3. Интерпретатор Python

Python — интерпретируемый язык программирования. Он не конвертирует свой код в машинный, который понимает железо (в отличие от C и C++). Вместо этого, Python-интерпретатор переводит код программы в байт-код, который запускается на виртуальной машине Python (PVM).

Интерпретатор — это программа, которая конвертирует ваши инструкции, написанные на ЯВУ, в байт-код и выполняет их. По сути интерпретатор — это программный слой между вашим исходным кодом и железом. Интерпретатор "Питона" выполняет любую программу поэтапно.



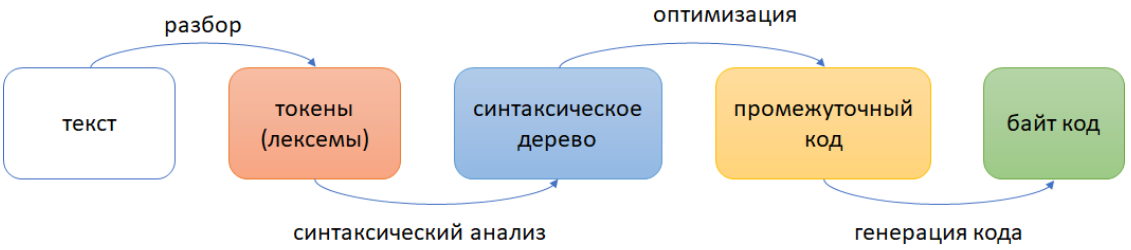
Этап #1. Инициализация После запуска вашей программы, Python-интерпретатор читает код, проверяет форматирование и синтаксис. При обнаружении ошибки он незамедлительно останавливается и показывает сообщение об ошибке.

Помимо этого, происходит ряд подготовительных процессов:

- анализ аргументов командной строки;

- установка флагов программы;
- чтение переменных среды и т.д.

Этап #2. Компиляция Интерпретатор транслирует (переводит) исходные инструкции вашей программы в байт-код (низкоуровневое, платформонезависимое представление исходного текста). Такая трансляция необходима в первую очередь для повышения скорости — байт-код выполняется в разы быстрее, чем исходные инструкции.



Если Python-интерпретатор обладает правом записи, он будет сохранять байт-код в виде файла с расширением .pyc. Если исходный текст программы не изменился с момента последней компиляции, при следующем запуске вашей программы, Python сразу загрузит файл .pyc, минуя этап компиляции (тем самым ускорит процесс запуска программы).

Этап #3. Выполнения Как только байт-код скомпилирован, он отправляется на виртуальную машину Python (PVM). Здесь выполняется байт-код на PVM. Если во время этого выполнения возникает ошибка, то выполнение останавливается с сообщением об ошибке.

PVM является частью Python-интерпретатора. По сути это просто большой цикл, который выполняет перебор инструкций в байт-коде и выполняет соответствующие им операции.

1.4. Циклы в Python

Цикл while

While - один из самых универсальных циклов в Python, поэтому довольно медленный. Выполняет тело цикла до тех пор, пока условие цикла истинно.

```
<Начальное значение или счётчик>
while <Логическое выражение, условие>:
            <Инструкции>
            <Приращение или увеличение значения в условии, чтобы не было бесконечного цикла>
      else:
    <Блок, выполняемый, если не было break>
```

```
i = 5
while i < 15:
  print(i)
  i = i + 3
```

```
5
8
11
14
```

Цикл for

Цикл for уже чуточку сложнее, чуть менее универсальный, но выполняется гораздо быстрее цикла while. Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
for <текущий элемент> in <последовательность>:
            <инструкции внутри цикла, которые нужно выполнить>
      else:
            <блок, выполняемый, если не использовался оператор break>
```

```
for i in 'hello world':
  print(i * 2, end='')
```

```
hheellllloo  wwoorrlldd
```

Оператор continue

Оператор continue начинает следующий проход цикла, минуя оставшееся тело цикла (for или while)

```
for i in 'hello world':
  if i == 'l':
    break
  print(i * 2, end='')
```

```
hhee
```

Ключевое слово else

Слово else, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же "естественным" образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

```
for i in 'hello world':
  if i == 'a':
    break
else:
    print('Буквы a в строке нет')
```

```
Буквы a в строке нет
```

1.5. Условная инструкция if-elif-else

Инструкция if-elif-else, проверка истинности, трехместное выражение if/else (её ещё иногда называют оператором ветвления) - основной инструмент выбора в Python. Проще говоря, она выбирает, какое действие следует выполнить, в зависимости от значения переменных в момент проверки условия.

Синтаксис инструкции if

Сначала записывается часть if с условным выражением, далее могут следовать одна или более необязательных частей elif, и, наконец, необязательная часть else. Общая форма записи условной инструкции if выглядит следующим образом:

```
if test1:
    state1
elif test2:
    state2
else:
    state3
```

Простой пример (напечатает 'true', так как 1 - истина):

```
if 0:
  print('true')
else:
  print('false')
```

```
true
```

Чуть более сложный пример (результат будет зависеть от того, что ввёл пользователь):

```
a = int(input())
if a < -5:
    print('Low')
elif -5 <= a <= 5:
    print('Mid')
else:
    print('High')
```

```
-10
Low
```

Конструкция с несколькими elif может также служить отличной заменой конструкции switch - case в других языках программирования.

Проверка истинности в Python

- Любое число, не равное 0, или непустой объект - истина.
- Числа, равные 0, пустые объекты и значение None - ложь
- Операции сравнения применяются к структурам данных рекурсивно
- Операции сравнения возвращают True или False
- Логические операторы and и or возвращают истинный или ложный объект-операнд

Логические операторы: логическое И (&)

```
X and Y
```

Истина, если оба значения X и Y истинны. логическое ИЛИ (|)

```
X or Y
```

логическое отрицание (!) Истина, если хотя бы одно из значений X или Y истинно.

```
not X
```

Истина, если X ложно.

Трехместное выражение if/else Следующая инструкция:

```
if X:
    A = Y
else:
    A = Z
```

довольно короткая, но, тем не менее, занимает целых 4 строки. Специально для таких случаев и было придумано выражение if/else:

```
A = Y if X else Z
```

В данной инструкции интерпретатор выполнит выражение Y, если X истинно, в противном случае выполнится выражение Z.

```
size = 1500000
a = 'city' if size > 100000 else 'town'
a
```

```
{"type":"string"}
```

1.6. Работа со строками

В программировании, строковый тип (англ. string «нить, вереница») — тип данных, значениями которого является произвольная последовательность (строка) символов алфавита. Каждая переменная такого типа (строковая переменная) может быть представлена фиксированным количеством байтов либо иметь произвольную длину.

Преобразование к строке `str([object], [кодировка], [ошибки])` - строковое представление объекта. Использует метод `__str__`.

- Если не передать параметры, то функция str вернёт пустую строку;
- Если указан только первый параметр, то функция str вернёт строковое представление;
- Если указать объект типа байт и кодировку, то функция str осуществит декодирование объекта и вернёт его строковое представление в конкретной кодировке.

Базовые операции Конкатенация (сложение)

```
s1 = 'Hello '
s2 = 'world!'
s = s1 + s2
print(s)
```

```
Hello world!
```

Длина строки (функция len)

```
len('Hello world!')
```

```
12
```

Доступ по индексу

```
print(s1[0])
print(s1[3])
print(s[-2])
```

```
H
l
d
```

Как видно из примера, в Python возможен и доступ по отрицательному индексу, при этом отсчет идет от конца строки.

Извлечение среза

Оператор извлечения среза: [X:Y]. X – начало среза, а Y – окончание; символ с номером Y в срез не входит. По умолчанию первый индекс равен 0, а второй - длине строки.

```
s[3:5]
```

```
{"type":"string"}
```

```
s[2:-2]
```

```
{"type":"string"}
```

```
s[:]
```

```
{"type":"string"}
```

Кроме того, можно задать шаг, с которым нужно извлекать срез.

```
s[3:5:-1]
```

```
{"type":"string"}
```

```
s[2::2]
```

```
{"type":"string"}
```

Расщепление строки

```
<строка>.split(<разделитель>)
```

```
arr = "Привет вам из python".split(' ')
arr
```

```
['Привет', 'вам', 'из', 'python']
```

Сцепление списка в одну строку

```
'<разделитель>'.join(<список>)
```

```
print(','.join(arr))
```

```
Привет,вам,из,python
```

Индексы и срезы

Взятие элемента по индексу Как и в других языках программирования, взятие по индексу:

```
a = [1, 3, 8, 7]
a[0]
```

```
1
```

Как и во многих других языках, нумерация элементов начинается с нуля. При попытке доступа к несуществующему индексу возникает исключение IndexError.

В данном примере переменная a являлась списком, однако взять элемент по индексу можно и у других типов: строк, кортежей.

В Python также поддерживаются отрицательные индексы, при этом нумерация идёт с конца, например:

```
a[-1]
```

```
7
```

Срезы В Python, кроме индексов, существуют ещё и срезы.

`item[START:STOP:STEP]` - берёт срез от номера START, до STOP (не включая его), с шагом STEP. По умолчанию START = 0, STOP = длине объекта, STEP = 1. Соответственно, какие-нибудь (а возможно, и все) параметры могут быть опущены.

```
a[:]
```

```
[1, 3, 8, 7]
```

1.7. Файлы. Работа с файлами.

Открытие файла Прежде, чем работать с файлом, его надо открыть. С этим замечательно справится встроенная функция open:

```
f = open('text.txt', 'r')
```

Путь к файлу может быть относительным или абсолютным. Второй аргумент, это режим, в котором мы будем открывать файл.

Чтение из файла Файл открыт, а теперь мы хотим прочитать из него информацию. Для этого есть несколько способов, но большого интереса заслуживают лишь два из них. Первый - метод read, читающий весь файл целиком, если был вызван без аргументов, и n символов, если был вызван с аргументом (целым числом n).

```
f = open('sample_data/mnist_test.csv')
f.read(11)
```

```
{"type":"string"}
```

Ещё один способ сделать это - прочитать файл построчно, воспользовавшись циклом for:

```
f = open('text.txt')
for line in f:
  line
```

```
i = 0
for line in f:
  i = i + 1
  if i < 10:
    print(line)
  else:
    break
```

После окончания работы с файлом его обязательно нужно закрыть с помощью метода close:

```
f.close()
```

1.8. With ... as - менеджеры контекста

Конструкция with ... as используется для оборачивания выполнения блока инструкций менеджером контекста. Иногда это более удобная конструкция, чем try...except...finally. Синтаксис конструкции with ... as:

```
"with" expression ["as" target] ("," expression ["as" target])* ":"
    suite
```

Теперь по порядку о том, что происходит при выполнении данного блока:

1. Выполняется выражение в конструкции with ... as.
2. Загружается специальный метод __exit__ для дальнейшего использования.
3. Выполняется метод __enter__. Если конструкция with включает в себя слово as, то возвращаемое методом __enter__ значение записывается в переменную.
4. Выполняется suite.
5. Вызывается метод __exit__, причём неважно, выполнилось ли suite или произошло исключение. В этот метод передаются параметры исключения, если оно произошло, или во всех аргументах значение None, если исключения не было.

Если в конструкции with - as было несколько выражений, то это эквивалентно нескольким вложенным конструкциям:

```
with A() as a, B() as b:
    suite
```

эквивалентно

```
with A() as a:
    with B() as b:
        suite
```

Для чего применяется конструкция with ... as? Для гарантии того, что критические функции выполнятся в любом случае. Самый распространённый пример использования этой конструкции - открытие файлов. Я уже рассказывал об открытии файлов с помощью функции open, однако конструкция with ... as, как правило, является более удобной и гарантирует закрытие файла в любом случае.

Например:

```
sum = 0
with open('sample_data/mnist_test.csv') as f:
  for line in f:
    lst = line.split(',') # расщепили строку на список
    first_column = lst[0] # берем только первую колонку
    sum = sum + int(first_column) #чтобы суммировать преобразуем в число
sum
```

```
0
```

1.9. Функции и их аргументы

Именные функции, инструкция def Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции def.

Определим простейшую функцию:

```
def add(x, y):
    return x + y
```

Инструкция return говорит, что нужно вернуть значение. В нашем случае функция возвращает сумму x и y.

Теперь мы ее можем вызвать:

```
add(1, 10)
```

```
11
```

Аргументы функции

Функция может принимать произвольное количество аргументов или не принимать их вовсе. Также распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными.

```
def func(a, b, c=2): # c - необязательный аргумент
  return a + b + c

>>> func(1, 2)  # a = 1, b = 2, c = 2 (по умолчанию)
5
>>> func(a=1, b=3)  # a = 1, b = 3, c = 2
6
```

Функция также может принимать переменное количество позиционных аргументов, тогда перед именем ставится *:

```
def func(*args):
  return args

>>> func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func()
()
```

```
>>> func(1)
(1,)
```

Как видно из примера, args - это кортеж из всех переданных аргументов функции, и с переменной можно работать также, как и с кортежем.

Функция может принимать и произвольное число именованных аргументов, тогда перед именем ставится **:

```
def func(**kwargs):
  return kwargs

>>> func(a=1, b=2, c=3)
{'a': 1, 'c': 3, 'b': 2}
>>> func()
{}
>>> func(a='python')
{'a': 'python'}
```

Анонимные функции, инструкция lambda

Анонимные функции могут содержать лишь одно выражение, но и выполняются они быстрее. Анонимные функции создаются с помощью инструкции lambda. Кроме этого, их не обязательно присваивать переменной, как делали мы инструкцией def func():

```
func = lambda x, y: x + y
>>> func(1, 2)
3
```

Попробуем переписать пример работы с файлом с использованием функций:

```
parseFirstInt = lambda strList: int(strList[0]) # берем только первую
колонку и преобразуем в число

def parseLine(s):
  return parseFirstInt(s.split(',')) # расщепили строку на список

sum = 0
with open('sample_data/mnist_test.csv') as f:
  for line in f:
    sum = sum + parseLine(line) # что бы суммировать
sum
```

```
44434
```

Выводы

1. Машины работают с кодом, людям же удобнее писать программы словами. За долгое время развития вычислительной техники люди придумали очень много языков программирования для разных задач.
2. Python - один из самых популярных языков программирования с низким порогом вхождения, реализующий большую часть необходимых в работе по автоматизации рутинных действий функциональностей.
3. Работа с файлами одна из базовых возможностей языка.

4. Можно поизучать сайт pythonworld, а так же документацию или ещё документация и руководство

2. Основы NumPy и Pandas

2.1. Библиотека NumPy

NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами. Основным объектом NumPy является однородный многомерный массив (в numpy называется numpy.ndarray). Это многомерный массив элементов (обычно чисел), одного типа.

Наиболее важные атрибуты объектов ndarray:

ndarray.ndim - число измерений (чаще их называют "оси") массива.

ndarray.shape - размеры массива, его форма. Это кортеж натуральных чисел, показывающий длину массива по каждой оси. Для матрицы из n строк и m столбов, shape будет (n,m). Число элементов кортежа shape равно ndim.

ndarray.size - количество элементов массива. Очевидно, равно произведению всех элементов атрибута shape.

ndarray.dtype - объект, описывающий тип элементов массива. Можно определить dtype, используя стандартные типы данных Python. NumPy здесь предоставляет целый букет возможностей, как встроенных, например: bool_, character, int8, int16, int32, int64, float8, float16, float32, float64, complex64, object_, так и возможность определить собственные типы данных, в том числе и составные.

ndarray.itemsize - размер каждого элемента массива в байтах.

ndarray.data - буфер, содержащий фактические элементы массива. Обычно не нужно использовать этот атрибут, так как обращаться к элементам массива проще всего с помощью индексов.

Создание массивов В NumPy существует много способов создать массив. Один из наиболее простых - создать массив из обычных списков или кортежей Python, используя функцию numpy.array() (запомните: array - функция, создающая объект типа ndarray):

Создание NumPy объектов

```
import numpy as np
a = np.array([1, 2, 3])
a
```

```
array([1, 2, 3])
```

Функция array() трансформирует вложенные последовательности в многомерные массивы. Тип элементов массива зависит от типа элементов исходной последовательности (но можно и переопределить его в момент создания).

```
b = np.array([[1.5, 2, 3], [4, 5, 6]])
b
```

```
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
```

Можно также переопределить тип в момент создания:

```
b = np.array([[1.5, 2, 3], [4, 5, 6]], dtype=np.complex128)
b
```

```
array([[1.5+0.j, 2. +0.j, 3. +0.j],
       [4. +0.j, 5. +0.j, 6. +0.j]])
```

Функция array() не единственная функция для создания массивов. Обычно элементы массива вначале неизвестны, а массив, в котором они будут храниться, уже нужен. Поэтому имеется несколько функций для того, чтобы создавать массивы с каким-то исходным содержимым (по умолчанию тип создаваемого массива — float64).

Функция zeros() создает массив из нулей, а функция ones() — массив из единиц. Обе функции принимают кортеж с размерами, и аргумент dtype:

```
np.zeros((3, 5), dtype=np.complex128)
```

```
array([[0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j]])
```

```
np.ones((2, 2, 2))
```

```
array([[[1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.]]],


      [[[1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.]]]])
```

Функция eye() создаёт единичную матрицу (двумерный массив)

```
np.eye(5)
```

```
array([[1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.]])
```

Функция empty() создает массив без его заполнения. Исходное содержимое случайно и зависит от состояния памяти на момент создания массива (то есть от того мусора, что в ней хранится):

```
np.empty((2, 2))
```

```
array([[7.74860419e-304, 7.74860419e-304],
       [7.74860419e-304, 7.74860419e-304]])
```

```
np.empty((3, 3))
```

```
array([[2.38170945e-316, 1.77863633e-322, 0.00000000e+000],
       [0.00000000e+000, 8.48798316e-313, 8.77837657e-071],
       [9.04747669e-043, 5.25080487e-090, 3.95360910e+179]])
```

Для создания последовательностей чисел, в NumPy имеется функция arange(), аналогичная встроенной в Python range(), только вместо списков она возвращает массивы, и принимает не только целые значения:

```
np.arange(10, 30, 5)
```

```
array([10, 15, 20, 25])
```

```
np.arange(0, 1, 0.1)
```

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

Вообще, при использовании arange() с аргументами типа float, сложно быть уверенным в том, сколько элементов будет получено (из-за ограничения точности чисел с плавающей запятой). Поэтому, в таких случаях обычно лучше использовать функцию linspace(), которая вместо шага в качестве одного из аргументов принимает число, равное количеству нужных элементов:

```
np.linspace(0, 2, 9)  # 9 чисел от 0 до 2 включительно
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
```

fromfunction(): применяет функцию ко всем комбинациям индексов

```
def f1(i, j):
  return 100 * i + 10*j

np.fromfunction(f1, (5, 5))
```

```
array([[  0.,  10.,  20.,  30.,  40.],
       [100., 110., 120., 130., 140.],
       [200., 210., 220., 230., 240.],
       [300., 310., 320., 330., 340.],
       [400., 410., 420., 430., 440.]])
```

```
f2 = lambda x,y : (x%2 == 0) & (y%2 == 0);
np.fromfunction(f2, (5, 5))
```

```
array([[ True, False,  True, False,  True],
       [False, False, False, False, False],
       [ True, False,  True, False,  True],
       [False, False, False, False, False],
       [ True, False,  True, False,  True]])
```

Базовые операции Математические операции над массивами выполняются поэлементно. Создается новый массив, который заполняется результатами действия оператора.

```
a = np.array([20, 30, 40, 50])
b = np.arange(4)
a + b
```

```
array([20, 31, 42, 53])
```

```
a - b
```

```
array([20, 29, 38, 47])
```

```
a * b
```

```
array([  0,  30,  80, 150])
```

```
a / b  # При делении на 0 возвращается inf (бесконечность)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

```
array([       inf, 30.        , 20.        , 16.66666667])
```

```
a ** b
```

```
array([     1,     30,   1600, 125000])
```

```
a % b  # При взятии остатка от деления на 0 возвращается 0
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
RuntimeWarning: divide by zero encountered in remainder
  """Entry point for launching an IPython kernel.
```

```
array([0, 0, 0, 2])
```

Для этого, естественно, массивы должны быть одинаковых размеров.

Также можно производить математические операции между массивом и числом. В этом случае к каждому элементу прибавляется (или что вы там делаете) это число.

```
 a + 1
```

```
array([21, 31, 41, 51])
```

```
a ** 3
```

```
array([  8000,  27000,  64000, 125000])
```

```
a < 35  # И фильтрацию можно проводить
```

```
array([ True,  True, False, False])
```

NumPy также предоставляет множество математических операций для обработки массивов:

```
np.cos(a)
```

```
array([ 0.40808206,  0.15425145, -0.66693806,  0.96496603])
```

```
np.arctan(a)
```

```
array([1.52083793, 1.53747533, 1.54580153, 1.55079899])
```

```
np.sinh(a)
```

```
array([2.42582598e+08, 5.34323729e+12, 1.17692633e+17, 2.59235276e+21])
```

Многие агрегирующие унарные операции, такие как, например, вычисление суммы всех элементов массива, представлены также и в виде методов класса ndarray.

```
a = np.array([[1, 2, 3], [4, 5, 6]])
np.sum(a)
```

```
21
```

```
a.sum()
```

```
21
```

```
a.min()
```

```
1
```

```
a.max()
```

```
6
```

По умолчанию, эти операции применяются к массиву, как если бы он был списком чисел, независимо от его формы. Однако, указав параметр axis, можно применить операцию для указанной оси массива:

```
a.max(axis=0)  # Наименьшее число в каждом столбце
print("Result: ")
print(a.max(axis=0))
a
```

```
Result:
[4 5 6]
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
a.min(axis=1)  # Наименьшее число в каждой строке
```

```
array([1, 4])
```

Печать массивов Если массив слишком большой, чтобы его печатать, NumPy автоматически скрывает центральную часть массива и выводит только его уголки.

```
print(np.arange(0, 3000, 1))
```

```
[   0    1    2 ... 2997 2998 2999]
```

```
for i in np.arange(0, 30, 1):
  if i != 15:
    print(i)
  else:
    print("something else")
```

```
0
1
2
```

```
3
4
5
6
7
8
9
10
11
12
13
14
something else
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```

Выводы

1. Работу с числами, матрицами, тензорами в python лучше всего производить с использованием библиотеки NumPy. Многие функции и свойства сложных математических объектов (например скалярное и векторное произведение для матриц) уже реализованы.
2. Для эффективной работы с библиотекой требуется тщательно изучить функционал.
3. Документация у NumPy.

2.2. Первичный анализ данных с Pandas

Pandas — это библиотека Python, предоставляющая широкие возможности для анализа данных. Данные, с которыми работают датасаентисты, часто хранятся в форме табличек — например, в форматах .csv, .tsv или .xlsx. С помощью библиотеки Pandas такие табличные данные очень удобно загружать, обрабатывать и анализировать с помощью SQL-подобных запросов. А в связке с библиотеками Matplotlib и Seaborn Pandas предоставляет широкие возможности визуального анализа табличных данных.

Основными структурами данных в Pandas являются классы **Series** и **DataFrame**. Первый из них представляет собой одномерный индексированный массив данных некоторого фиксированного типа. Второй – это двухмерная структура данных, представляющая собой таблицу, каждый столбец которой содержит данные одного типа. Можно представлять её как словарь объектов типа Series. Структура DataFrame

отлично подходит для представления реальных данных: строки соответствуют признаковым описаниям отдельных объектов, а столбцы соответствуют признакам.

```
!wget https://datahub.io/core/airport-codes/r/airport-codes.csv -O
/content/sample_data/airport-codes.csv -q
```

```
!head -n 10 /content/sample_data/airport-codes.csv
```

```
ident,type,name,elevation_ft,continent,iso_country,iso_region,municipality,gp
s_code,iata_code,local_code,coordinates
00A,heliport,Total Rf
Heliport,11,NA,US,US-PA,Bensalem,00A,,00A,"-74.93360137939453,
40.07080078125"
00AA,small_airport,Aero B Ranch
Airport,3435,NA,US,US-KS,Leoti,00AA,,00AA,"-101.473911, 38.704022"
00AK,small_airport,Lowell Field,450,NA,US,US-AK,Anchor
Point,00AK,,00AK,"-151.695999146, 59.94919968"
00AL,small_airport,Epps
Airpark,820,NA,US,US-AL,Harvest,00AL,,00AL,"-86.77030181884766,
34.86479949951172"
00AR,closed,Newport Hospital & Clinic
Heliport,237,NA,US,US-AR,Newport,,,,"-91.254898, 35.6087"
00AS,small_airport,Fulton
Airport,1100,NA,US,US-OK,Alex,00AS,,00AS,"-97.8180194, 34.9428028"
00AZ,small_airport,Cordes
Airport,3810,NA,US,US-AZ,Cordes,00AZ,,00AZ,"-112.16500091552734,
34.305599212646484"
00CA,small_airport,Goldstone /Gts/
Airport,3038,NA,US,US-CA,Barstow,00CA,,00CA,"-116.888000488,
35.350498199499995"
00CL,small_airport,Williams Ag
Airport,87,NA,US,US-CA,Biggs,00CL,,00CL,"-121.763427, 39.427188"
```

Базовые операции в Pandas

```
# импортируем Pandas и Numpy
import pandas as pd
import numpy as np

airports = pd.read_csv('sample_data/airport-codes.csv')
airports.head(200)
```

```
    ident           type                                        name  \
0     00A       heliport                         Total Rf Heliport
1    00AA  small_airport                       Aero B Ranch Airport
2    00AK  small_airport                              Lowell Field
3    00AL  small_airport                              Epps Airpark
4    00AR         closed          Newport Hospital & Clinic Heliport
..    ...            ...                                       ...
195  03AZ  small_airport  Thompson International Aviation Airport
196  03CA       heliport                Grossmont Hospital Heliport
197  03CO  small_airport                       Kugel-Strong Airport
198  03FA  small_airport                   Lake Persimmon Airstrip
199  03FD         closed                              Tharpe Airport

     elevation_ft continent iso_country iso_region  municipality gps_code  \
0            11.0       NaN          US      US-PA      Bensalem      00A
```

```
1          3435.0       NaN          US      US-KS         Leoti     00AA
2           450.0       NaN          US      US-AK  Anchor Point     00AK
3           820.0       NaN          US      US-AL       Harvest     00AL
4           237.0       NaN          US      US-AR       Newport      NaN
..            ...       ...         ...        ...           ...      ...
195        4275.0       NaN          US      US-AZ      Hereford     03AZ
196         634.0       NaN          US      US-CA       La Mesa     03CA
197        4950.0       NaN          US      US-CO   Platteville     03CO
198          70.0       NaN          US      US-FL   Lake Placid     03FA
199         115.0       NaN          US      US-FL       Bonifay      NaN

    iata_code local_code                             coordinates
0         NaN        00A       -74.93360137939453, 40.07080078125
1         NaN       00AA                  -101.473911, 38.704022
2         NaN       00AK               -151.695999146, 59.94919968
3         NaN       00AL    -86.77030181884766, 34.86479949951172
4         NaN        NaN                      -91.254898, 35.6087
..        ...        ...                                      ...
195       NaN       03AZ  -110.08399963378906, 31.433399200439453
196       NaN       03CA                  -117.006952, 32.779484
197       NaN       03CO            -104.744003296, 40.2125015259
198       NaN       03FA   -81.40809631347656, 27.353099822998047
199       NaN        NaN                      -85.731003, 30.8288

[200 rows x 12 columns]
```

В Jupyter-ноутбуках датафреймы Pandas выводятся в виде вот таких красивых табличек, и print(df.head()) выглядит хуже.

```
print(airports.head())
```

```
  ident          type                                name  elevation_ft  \
0   00A       heliport                Total Rf Heliport          11.0
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                       Lowell Field         450.0
3  00AL  small_airport                       Epps Airpark         820.0
4  00AR         closed  Newport Hospital & Clinic Heliport         237.0

  continent iso_country iso_region  municipality gps_code iata_code  \
0       NaN          US      US-PA      Bensalem      00A       NaN
1       NaN          US      US-KS         Leoti     00AA       NaN
2       NaN          US      US-AK  Anchor Point     00AK       NaN
3       NaN          US      US-AL       Harvest     00AL       NaN
4       NaN          US      US-AR       Newport      NaN       NaN

  local_code                             coordinates
0        00A       -74.93360137939453, 40.07080078125
1       00AA                    -101.473911, 38.704022
2       00AK                -151.695999146, 59.94919968
3       00AL    -86.77030181884766, 34.86479949951172
4        NaN                       -91.254898, 35.6087
```

По умолчанию Pandas выводит всего 20 столбцов и 60 строк, поэтому если ваш датафрейм больше, воспользуйтесь функцией set_option:

```
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
```

Посмотрим на размер данных, названия признаков и их типы.

```
print(airports.shape)
```

```
(57421, 12)
```

Видим, что в таблице 57421 строки и 12 столбцов. Выведем названия столбцов:

```
print(airports.columns)
```

```
Index(['ident', 'type', 'name', 'elevation_ft', 'continent', 'iso_country',
       'iso_region', 'municipality', 'gps_code', 'iata_code', 'local_code',
       'coordinates'],
      dtype='object')
```

Чтобы посмотреть общую информацию по датафрейму и всем признакам, воспользуемся методом **info**:

```
print(airports.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57421 entries, 0 to 57420
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   ident         57421 non-null  object 
 1   type          57421 non-null  object 
 2   name          57421 non-null  object 
 3   elevation_ft  49608 non-null  float64
 4   continent     28978 non-null  object 
 5   iso_country   57175 non-null  object 
 6   iso_region    57421 non-null  object 
 7   municipality  51527 non-null  object 
 8   gps_code      41561 non-null  object 
 9   iata_code     9225 non-null   object 
 10  local_code    30030 non-null  object 
 11  coordinates   57421 non-null  object 
dtypes: float64(1), object(11)
memory usage: 5.3+ MB
None
```

Применение функций к ячейкам, столбцам и строкам Как видно колонка coordinates не распозналась. Попробуем извлечь из неё данные, для этого воспользуемся методом apply и анонимными функциями. Для того, что бы удалить лишний столбец воспользуемся методом drop с параметром axis=1.

```
airports['latitude'] = airports['coordinates'].apply(lambda x :
float(x.split(',')[0]))
airports['altitude'] = airports['coordinates'].apply(lambda x :
float(x.split(',')[1]))
airports = airports.drop(['coordinates'], axis=1)
airports.head()
```

```
  ident           type                                  name  elevation_ft  \
0   00A       heliport                 Total Rf Heliport          11.0
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                       Lowell Field         450.0
3  00AL  small_airport                       Epps Airpark         820.0
4  00AR         closed  Newport Hospital & Clinic Heliport         237.0

  continent iso_country iso_region  municipality gps_code iata_code  \
0       NaN          US      US-PA      Bensalem      00A       NaN
1       NaN          US      US-KS         Leoti     00AA       NaN
2       NaN          US      US-AK  Anchor Point     00AK       NaN
3       NaN          US      US-AL       Harvest     00AL       NaN
4       NaN          US      US-AR       Newport      NaN       NaN

  local_code    latitude   altitude
0        00A  -74.933601  40.070801
1       00AA -101.473911  38.704022
2       00AK -151.695999  59.949200
3       00AL  -86.770302  34.864799
4        NaN  -91.254898  35.608700
```

Метод apply можно использовать и для того, чтобы применить функцию к каждой строке. Для этого нужно указать `axis=0`. Колонки **continent** и **iata_code** заполнились не очень хорошо. Примененим функции к каждой ячейке столбца с помощью `fillna` или `replace` для `np.nan`:

```
airports['continent'] = airports['continent'].fillna('')
airports['iata_code'] = airports['iata_code'].replace(np.nan, '0')
airports.head()
```

```
  ident           type                                  name  elevation_ft  \
0   00A       heliport                 Total Rf Heliport          11.0
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                       Lowell Field         450.0
3  00AL  small_airport                       Epps Airpark         820.0
4  00AR         closed  Newport Hospital & Clinic Heliport         237.0

  continent iso_country iso_region  municipality gps_code iata_code  \
0                   US      US-PA      Bensalem      00A         0
1                   US      US-KS         Leoti     00AA         0
2                   US      US-AK  Anchor Point     00AK         0
3                   US      US-AL       Harvest     00AL         0
4                   US      US-AR       Newport      NaN         0

  local_code    latitude   altitude
0        00A  -74.933601  40.070801
1       00AA -101.473911  38.704022
2       00AK -151.695999  59.949200
3       00AL  -86.770302  34.864799
4        NaN  -91.254898  35.608700
```

Для того, что бы понять какие бывают уникальные значения колонки можно воспользоваться методом `unique`

```
airports['continent'].unique()
```

```
array(['', 'OC', 'AF', 'AN', 'EU', 'AS', 'SA'], dtype=object)
```

Иногда поле требуется перекодировать. Для этого можно воспользоваться словарём и методом map. Для подсчёта частоты встречаемости различных значений можно воспользоваться методом value_counts.

```
d = {'SA':'South America', 'EU':'Europe', 'AS' : 'Asia','AF':'Africa',
'OC':'Oceania', 'AN':'Antarctida', '':'Unknown'}
airports['continent'] = airports['continent'].map(d)
airports['continent'].value_counts()
```

```
Unknown          28443
South America     8443
Europe            8404
Asia              5619
Africa            3361
Oceania           3123
Antarctida          28
Name: continent, dtype: int64
```

Фильтрация кадров данных

Часто данные требуется отфильтровать. Причём фильтрация может потребоваться как по атрибутам (колонкам) так и по кортежам (строкам) фильтры регулируются по оси. Можно фильтровать кадры данных можно без удаления filter или с удалением drop.

```
airports.filter(items=['ident', 'type', 'continent', 'name']).head() #
фильтрация колонок
```

```
  ident           type continent                                name
0   00A       heliport                                Total Rf Heliport
1  00AA  small_airport                           Aero B Ranch Airport
2  00AK  small_airport                                   Lowell Field
3  00AL  small_airport                                   Epps Airpark
4  00AR         closed            Newport Hospital & Clinic Heliport
```

```
airports.filter(like='00', axis=0).head() # фильтрация строк
```

```
    ident           type               name  elevation_ft continent  \
100  01NE  small_airport     Detour Airport        3000.0
200  03FL       heliport    Ranger Heliport          20.0
300  04TX  small_airport     Pocock Airport         565.0
400  06MO  small_airport  Noahs Ark Airport         755.0
500  08ID  small_airport      Symms Airport        2680.0

    iso_country iso_region     municipality gps_code iata_code local_code  \
100          US      US-NE        Wellfleet     01NE         0       01NE
200          US      US-FL  West Palm Beach     03FL         0       03FL
300          US      US-TX     China Spring     04TX         0       04TX
400          US      US-MO          Waldron     06MO         0       06MO
500          US      US-ID          Marsing     08ID         0       08ID

       latitude   altitude
100 -100.653000  40.843601
200  -80.187302  26.683701
```

```
300  -97.368896  31.732201
400  -94.804398  39.230598
500 -116.777000  43.569302
```

```
airports.drop(['continent'], axis=1).head()
```

```
  ident           type                                  name  elevation_ft  \
0   00A       heliport                 Total Rf Heliport          11.0
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                       Lowell Field         450.0
3  00AL  small_airport                       Epps Airpark         820.0
4  00AR         closed  Newport Hospital & Clinic Heliport         237.0

  iso_country iso_region  municipality gps_code iata_code local_code  \
0          US      US-PA      Bensalem      00A         0        00A
1          US      US-KS         Leoti     00AA         0       00AA
2          US      US-AK  Anchor Point     00AK         0       00AK
3          US      US-AL       Harvest     00AL         0       00AL
4          US      US-AR       Newport      NaN         0        NaN

     latitude   altitude
0  -74.933601  40.070801
1 -101.473911  38.704022
2 -151.695999  59.949200
3  -86.770302  34.864799
4  -91.254898  35.608700
```

```
airports.drop(index=0, axis=0).head()
```

```
  ident           type                                  name  elevation_ft  \
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                       Lowell Field         450.0
3  00AL  small_airport                       Epps Airpark         820.0
4  00AR         closed  Newport Hospital & Clinic Heliport         237.0
5  00AS  small_airport                        Fulton Airport        1100.0

  continent iso_country iso_region  municipality gps_code iata_code  \
1                    US      US-KS         Leoti     00AA         0
2                    US      US-AK  Anchor Point     00AK         0
3                    US      US-AL       Harvest     00AL         0
4                    US      US-AR       Newport      NaN         0
5                    US      US-OK          Alex     00AS         0

  local_code    latitude   altitude
1       00AA -101.473911  38.704022
2       00AK -151.695999  59.949200
3       00AL  -86.770302  34.864799
4        NaN  -91.254898  35.608700
5       00AS  -97.818019  34.942803
```

Объединение кадров данных Распространённая задача - это обогащение данных, т.е. объединение кадра с источником приносящим дополнительную информацию. Объедиенение кадров производится по ключевому атрибуту. Рассмотрим пример:

```
poles = pd.DataFrame({'continent':['Unknown', 'Oceania', 'Africa', 'Africa',
'Antarctida', 'Europe', 'Asia', 'South America'],
```

```
'pole':['Unknown','South','South','North','South','North','North','South']})
poles.head()
```

```
    continent     pole
0     Unknown  Unknown
1     Oceania    South
2      Africa    South
3      Africa    North
4  Antarctida    South
```

```
airports.set_index('continent').join(poles.set_index('continent'),
on='continent',how='left').filter(like='Africa', axis=0).head()
```

```
          ident           type              name  elevation_ft iso_country  \
continent
Africa      AAD  small_airport     Adado Airport        1001.0          SO
Africa      AAD  small_airport     Adado Airport        1001.0          SO
Africa      ADV  small_airport  El Daein Airport        1560.0          SD
Africa      ADV  small_airport  El Daein Airport        1560.0          SD
Africa      AEE  small_airport   Adareil Airport        1301.0          SS

          iso_region municipality gps_code iata_code local_code   latitude  \
continent
Africa         SO-GA        Adado      NaN       AAD        NaN  46.637500
Africa         SO-GA        Adado      NaN       AAD        NaN  46.637500
Africa         SD-DE     El Daein      NaN       ADV        NaN  26.118600
Africa         SD-DE     El Daein      NaN       ADV        NaN  26.118600
Africa         SS-23          NaN      NaN       AEE        NaN  32.959444

            altitude   pole
continent
Africa      6.095802  South
Africa      6.095802  North
Africa     11.402300  South
Africa     11.402300  North
Africa     10.053611  South
```

Таким образом можно дополнять и **обогащать** выборку данных новым информативным смыслом. На самом деле в реальных задачах это гораздо сложнее, чем в продемонстрированном примере. Основная мысль этой демонстрации - это sql подобный синтаксис работы с pandas DataFrame.

Изучение данных

Не всегда данные находятся сразу в csv. иногда их требуется предварительно распарсить, например, их html. Это можно сделать как представленно в примере:

```
import html5lib
telecom =
pd.read_html('https://github.com/Yorko/mlcourse_open/blob/master/data/telecom
_churn.csv', header = 0)[0]
telecom.head()
```

```
   Unnamed: 0 State  Account length  Area code International plan  \
0         NaN    KS             128        415                 No
1         NaN    OH             107        415                 No
```

```
2         NaN    NJ             137        415                No
3         NaN    OH              84        408               Yes
4         NaN    OK              75        415               Yes

  Voice mail plan  Number vmail messages  Total day minutes  Total day calls
\
0             Yes                     25              265.1              110
1             Yes                     26              161.6              123
2              No                      0              243.4              114
3              No                      0              299.4               71
4              No                      0              166.7              113

   Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
0             45.07              197.4               99             16.78
1             27.47              195.5              103             16.62
2             41.38              121.2              110             10.30
3             50.90               61.9               88              5.26
4             28.34              148.3              122             12.61

   Total night minutes  Total night calls  Total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   Total intl minutes  Total intl calls  Total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   Customer service calls  Churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False

telecom.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Unnamed: 0              0 non-null      float64
 1   State                   3333 non-null   object
 2   Account length          3333 non-null   int64
 3   Area code               3333 non-null   int64
 4   International plan      3333 non-null   object
 5   Voice mail plan         3333 non-null   object
 6   Number vmail messages   3333 non-null   int64
 7   Total day minutes       3333 non-null   float64
 8   Total day calls         3333 non-null   int64
```

```
 9   Total day charge        3333 non-null   float64
 10  Total eve minutes       3333 non-null   float64
 11  Total eve calls         3333 non-null   int64
 12  Total eve charge        3333 non-null   float64
 13  Total night minutes     3333 non-null   float64
 14  Total night calls       3333 non-null   int64
 15  Total night charge      3333 non-null   float64
 16  Total intl minutes      3333 non-null   float64
 17  Total intl calls        3333 non-null   int64
 18  Total intl charge       3333 non-null   float64
 19  Customer service calls  3333 non-null   int64
 20  Churn                   3333 non-null   bool
dtypes: bool(1), float64(9), int64(8), object(3)
memory usage: 524.2+ KB
```

bool, int64, float64 и object — это типы аттрибутов (колонок, или признаков). Видим, что 1 атрибут — логический (bool), 3 атрибута имеют тип object и 16 признаков — числовые. Также с помощью метода info удобно быстро посмотреть на пропуски в данных, в нашем случае их нет, в каждом столбце по 3333 наблюдения.

Изменить тип колонки можно с помощью метода astype. Применим этот метод к признаку Churn и переведём его в int64:

Отдельно замечу, что **признаки** (или переменные) могут обладать различной природой, а значит и *статистическим распределением*. Глобально отличия природы распределения переменных бывает двух типов: количественные и категориальные. Переменная, принимающая значения из некоторого ограниченного набора категорий называется **категориальной**. Обычно связана с неисчисляемыми атрибутами, такими как названия (товаров, услуг и др.), имена людей, исходы событий (да/нет), пункты выбора в меню, тарифные планы, статусы и т.д. Синонимом может быть дискретный или качественный признак. Если категорий такое множество, что нам приходится задавать диапазон значений попадающих в эту категорию, то принято считать распределение такой величины квазинепрерывным (в статистике случайные величины могут быть не только дискретными и непрырывными, а например, в ограниченной областью рассеивания). Для таких признаков говорят, что они **количественные**. Например, возраст, длина, высота, доход, задолженность и т.д.

```
telecom['Churn'] = telecom['Churn'].astype('int64')
```

```
telecom = telecom.drop(telecom.columns[0], axis=1) # уберем не
информативную колонку
telecom.head()
```

```
  State  Account length  Area code International plan Voice mail plan  \
0    KS             128        415                 No             Yes
1    OH             107        415                 No             Yes
2    NJ             137        415                 No              No
3    OH              84        408                Yes              No
4    OK              75        415                Yes              No

   Number vmail messages  Total day minutes  Total day calls  \
0                     25              265.1              110
1                     26              161.6              123
```

```
2                     0              243.4              114
3                     0              299.4               71
4                     0              166.7              113

   Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
0             45.07              197.4               99             16.78
1             27.47              195.5              103             16.62
2             41.38              121.2              110             10.30
3             50.90               61.9               88              5.26
4             28.34              148.3              122             12.61

   Total night minutes  Total night calls  Total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   Total intl minutes  Total intl calls  Total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   Customer service calls  Churn
0                       1      0
1                       1      0
2                       0      0
3                       2      0
4                       3      0
```

Метод describe показывает основные статистические характеристики данных по каждому числовому признаку (типы int64 и float64): число непропущенных значений, среднее, стандартное отклонение, диапазон, медиану, 0.25 и 0.75 квартили.

Допущение о том, что распределение данных в признаке нормальное очень ненадёжное. Вы можете вызвать метод на категориальной переменной в которой хранятся статусы, но это **первое приближение**. На самом деле после более тщательного анализа распределения данных в категориальной переменной стоит уточнить, что для неё понятия квартилей, среднего и дисперсии выдаваемые pandas не имеют никакого смысла.

```
telecom.describe()
```

```
       Account length    Area code  Number vmail messages  Total day minutes
\
count     3333.000000  3333.000000            3333.000000        3333.000000
mean       101.064806   437.182418               8.099010         179.775098
std         39.822106    42.371290              13.688365          54.467389
min          1.000000   408.000000               0.000000           0.000000
25%         74.000000   408.000000               0.000000         143.700000
50%        101.000000   415.000000               0.000000         179.400000
75%        127.000000   510.000000              20.000000         216.400000
max        243.000000   510.000000              51.000000         350.800000
```

```
       Total day calls  Total day charge  Total eve minutes  Total eve calls
\
count      3333.000000       3333.000000        3333.000000      3333.000000
mean        100.435644         30.562307         200.980348       100.114311
std          20.069084          9.259435          50.713844        19.922625
min           0.000000          0.000000           0.000000         0.000000
25%          87.000000         24.430000         166.600000        87.000000
50%         101.000000         30.500000         201.400000       100.000000
75%         114.000000         36.790000         235.300000       114.000000
max         165.000000         59.640000         363.700000       170.000000

       Total eve charge  Total night minutes  Total night calls  \
count       3333.000000          3333.000000        3333.000000
mean          17.083540           200.872037         100.107711
std            4.310668            50.573847          19.568609
min            0.000000            23.200000          33.000000
25%           14.160000           167.000000          87.000000
50%           17.120000           201.200000         100.000000
75%           20.000000           235.300000         113.000000
max           30.910000           395.000000         175.000000

       Total night charge  Total intl minutes  Total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       Total intl charge  Customer service calls        Churn
count        3333.000000             3333.000000  3333.000000
mean            2.764581                1.562856     0.144914
std             0.753773                1.315491     0.352067
min             0.000000                0.000000     0.000000
25%             2.300000                1.000000     0.000000
50%             2.780000                1.000000     0.000000
75%             3.270000                2.000000     0.000000
max             5.400000                9.000000     1.000000
```

Чтобы посмотреть статистику по нечисловым признакам, нужно явно указать интересующие нас типы в параметре include.

```
telecom.describe(include=['object', 'bool'])
```

```
       State International plan Voice mail plan
count   3333               3333            3333
unique    51                  2               2
top       WV                 No              No
freq     106               3010            2411
```

Для категориальных (тип object) и булевых (тип bool) признаков можно воспользоваться методом value_counts. Посмотрим на распределение данных по нашей целевой переменной — Churn:

```
telecom['Churn'].value_counts()
```

```
0    2850
1     483
Name: Churn, dtype: int64
```

Сортировка

DataFrame можно отсортировать по значению какого-нибудь из признаков. В нашем случае, например, по Total day charge (ascending=False для сортировки по убыванию):

```
telecom.sort_values(by='Total day charge', ascending=False).head()
```

```
     State  Account length  Area code International plan Voice mail plan  \
365     CO             154        415                 No              No
985     NY              64        415                Yes              No
2594    OH             115        510                Yes              No
156     OH              83        415                 No              No
605     MO             112        415                 No              No

      Number vmail messages  Total day minutes  Total day calls  \
365                       0              350.8               75
985                       0              346.8               55
2594                      0              345.3               81
156                       0              337.4              120
605                       0              335.5               77

      Total day charge  Total eve minutes  Total eve calls  Total eve charge
\
365              59.64              216.5               94             18.40
985              58.96              249.5               79             21.21
2594             58.70              203.4              106             17.29
156              57.36              227.4              116             19.33
605              57.04              212.5              109             18.06

      Total night minutes  Total night calls  Total night charge  \
365                 253.9                100               11.43
985                 275.4                102               12.39
2594                217.5                107                9.79
156                 153.9                114                6.93
605                 265.0                132               11.93

      Total intl minutes  Total intl calls  Total intl charge  \
365                 10.1                 9               2.73
985                 13.3                 9               3.59
2594                11.8                 8               3.19
156                 15.8                 7               4.27
605                 12.7                 8               3.43

      Customer service calls  Churn
365                        1      1
985                        1      1
2594                       1      1
156                        0      1
605                        2      1
```

Сортировать можно и по группе столбцов:

```
telecom.sort_values(by=['Churn', 'Total day charge'],
        ascending=[True, False]).head()
```

```
     State  Account length  Area code International plan Voice mail plan  \
688     MN              13        510                 No             Yes
2259    NC             210        415                 No             Yes
534     LA              67        510                 No              No
575     SD             114        415                 No             Yes
2858    AL             141        510                 No             Yes

      Number vmail messages  Total day minutes  Total day calls  \
688                      21              315.6              105
2259                     31              313.8               87
534                       0              310.4               97
575                      36              309.9               90
2858                     28              308.0              123

      Total day charge  Total eve minutes  Total eve calls  Total eve charge
\
688              53.65              208.9               71             17.76
2259             53.35              147.7              103             12.55
534              52.77               66.5              123              5.65
575              52.68              200.3               89             17.03
2858             52.36              247.8              128             21.06

      Total night minutes  Total night calls  Total night charge  \
688                 260.1                123               11.70
2259                192.7                 97                8.67
534                 246.5                 99               11.09
575                 183.5                105                8.26
2858                152.9                103                6.88

      Total intl minutes  Total intl calls  Total intl charge  \
688                 12.1                 3               3.27
2259                10.1                 7               2.73
534                  9.2                10               2.48
575                 14.2                 2               3.83
2858                 7.4                 3               2.00

      Customer service calls  Churn
688                        3      0
2259                       3      0
534                        4      0
575                        1      0
2858                       1      0
```

Группировка данных

В общем случае группировка данных в Pandas выглядит следующим образом:

```
df.groupby(by=grouping_columns)[columns_to_show].function()
```

4. К датафрейму применяется метод groupby, который разделяет данные по grouping_columns – признаку или набору признаков.

5. Выбираем нужные нам столбцы (columns_to_show).
6. К полученным группам применяется функция или несколько функций.

Группирование данных в зависимости от значения признака Churn и вывод статистик по трём столбцам в каждой группе.

```
columns_to_show = ['Total day minutes', 'Total eve minutes', 'Total night
minutes']

telecom.groupby(['Churn'])[columns_to_show].describe(percentiles=[])
```

```
      Total day minutes                                                \
                  count        mean        std  min    50%    max
Churn
0                2850.0  175.175754  50.181655  0.0  177.2  315.6
1                 483.0  206.914079  68.997792  0.0  217.6  350.8

      Total eve minutes                                               \
                  count        mean        std   min    50%    max
Churn
0                2850.0  199.043298  50.292175   0.0  199.6  361.8
1                 483.0  212.410145  51.728910  70.9  211.3  363.7

      Total night minutes
                    count        mean        std   min     50%    max
Churn
0                  2850.0  200.133193  51.105032  23.2  200.25  395.0
1                   483.0  205.231677  47.132825  47.4  204.80  354.9
```

```
columns_to_show = ['Total day minutes', 'Total eve minutes', 'Total night
minutes']

telecom.groupby(['State'])[columns_to_show].describe(percentiles=[])
```

```
      Total day minutes                                                  \
                  count        mean        std   min     50%    max
State
AK                 52.0  178.384615  49.640430  58.2  177.25  278.4
AL                 80.0  186.010000  51.466249  68.7  190.25  308.0
AR                 55.0  176.116364  50.368831  55.3  170.70  273.4
AZ                 64.0  171.604688  51.941907  58.9  171.45  281.1
CA                 34.0  183.564706  47.742484  92.8  183.20  280.0
CO                 66.0  178.712121  59.805856  30.9  180.90  350.8
CT                 74.0  175.140541  60.523424  37.8  176.50  321.6
DC                 54.0  171.379630  57.157338  51.5  169.20  306.2
DE                 61.0  174.583607  52.060645  46.5  179.90  334.3
FL                 63.0  179.533333  57.468499  47.7  181.80  288.1
GA                 54.0  185.025926  53.736275  71.2  193.30  299.5
HI                 53.0  175.962264  54.834311  41.9  181.40  291.6
IA                 44.0  177.613636  48.400925  88.1  168.80  308.6
ID                 73.0  178.619178  52.794622  55.6  181.60  274.4
IL                 58.0  173.591379  49.802932  69.1  180.35  269.6
IN                 71.0  196.525352  51.956157  49.9  203.80  300.4
KS                 70.0  191.555714  58.143148  27.0  191.25  321.3
KY                 59.0  173.754237  54.943583  73.8  170.50  314.6
LA                 51.0  178.376471  45.435139  58.4  179.30  310.4
```

```
MA                 65.0  180.103077  51.288790  58.9  178.10  293.7
MD                 70.0  197.228571  58.031576  78.1  198.15  321.1
ME                 62.0  185.262903  52.707427  58.8  193.80  322.3
MI                 73.0  180.593151  54.873206  18.9  185.30  314.1
MN                 84.0  183.354762  56.625260  50.6  178.60  317.8
MO                 63.0  170.506349  58.076573  45.0  165.90  335.5
MS                 65.0  177.929231  61.631895  70.7  166.50  313.2
MT                 68.0  174.007353  48.848851  89.8  162.30  273.2
NC                 68.0  185.145588  56.222470  54.7  189.80  322.3
ND                 62.0  187.338710  45.251481  82.5  191.25  295.3
NE                 61.0  177.465574  52.599645  34.0  180.90  272.7
NH                 56.0  177.328571  59.963106  17.6  182.85  322.4
NJ                 68.0  196.225000  48.608661  40.9  193.05  301.5
NM                 62.0  171.429032  44.931695  69.1  169.05  286.7
NV                 66.0  176.425758  56.561785  67.4  168.70  303.9
NY                 83.0  175.114458  56.786981  60.6  166.40  346.8
OH                 78.0  183.274359  55.755483   7.8  185.60  345.3
OK                 61.0  179.909836  61.730340   2.6  179.20  329.8
OR                 78.0  176.246154  56.013219  12.5  186.40  324.7
PA                 45.0  188.375556  55.137556  35.1  205.10  288.7
RI                 65.0  167.478462  55.418410  40.4  167.80  286.2
SC                 60.0  166.441667  63.585043  19.5  157.75  322.5
SD                 60.0  189.690000  55.366666   0.0  186.55  328.1
TN                 53.0  175.771698  50.608282  54.8  170.10  305.2
TX                 72.0  181.516667  57.146528  59.5  181.30  326.5
UT                 72.0  183.569444  53.796365  63.2  187.40  285.7
VA                 77.0  177.244156  49.273203  44.9  174.50  283.4
VT                 73.0  182.031507  52.048788   0.0  188.40  307.1
WA                 66.0  178.742424  56.412167  37.7  166.40  289.1
WI                 78.0  179.130769  57.867821   7.9  177.25  326.3
WV                106.0  173.950943  53.920065  58.0  176.25  312.0
WY                 77.0  180.170130  53.975375  25.9  178.30  296.0

      Total eve minutes                                              \
                   count        mean        std    min     50%    max
State
AK                 52.0  184.282692  49.160213   58.6  179.95  314.4
AL                 80.0  195.462500  50.909648   77.9  201.25  299.9
AR                 55.0  201.047273  50.957484  120.5  192.30  350.9
AZ                 64.0  187.748438  49.070513   72.9  191.10  328.7
CA                 34.0  198.970588  40.361770  114.0  195.45  281.3
CO                 66.0  206.884848  53.306802   75.3  211.20  341.3
CT                 74.0  203.828378  55.971033   66.0  209.85  335.0
DC                 54.0  196.272222  47.422479   65.2  198.10  337.1
DE                 61.0  208.247541  46.393479  132.5  206.60  328.2
FL                 63.0  210.276190  54.921782   69.2  209.00  318.8
GA                 54.0  204.140741  47.011053   73.2  199.95  304.4
HI                 53.0  191.343396  50.612090   90.0  192.40  305.8
IA                 44.0  206.400000  55.424480  102.2  200.45  329.3
ID                 73.0  194.610959  44.761049   98.3  195.70  292.7
IL                 58.0  196.798276  52.539016   48.1  198.00  319.3
IN                 71.0  202.559155  55.354715   83.9  199.40  361.8
KS                 70.0  202.512857  48.809158   95.1  197.35  310.6
KY                 59.0  196.244068  50.654953   87.6  195.70  324.8
LA                 51.0  197.819608  57.194972   31.2  205.10  351.6
```

MA	65.0	214.664615	52.439425	95.6	211.40	348.5
MD	70.0	196.061429	50.489311	90.2	194.15	354.2
ME	62.0	200.514516	44.995424	79.3	206.60	278.2
MI	73.0	208.172603	51.870258	89.1	208.60	336.0
MN	84.0	199.334524	46.627874	42.5	203.05	322.2
MO	63.0	200.141270	57.256014	60.8	208.60	332.1
MS	65.0	200.009231	50.163423	103.2	202.80	313.7
MT	68.0	201.526471	52.860757	88.1	210.65	312.6
NC	68.0	202.536765	47.720895	80.8	204.10	363.7
ND	62.0	207.775806	45.635307	89.3	209.85	289.9
NE	61.0	203.111475	54.304328	56.0	206.70	282.6
NH	56.0	198.158929	51.762225	58.9	201.40	303.2
NJ	68.0	198.289706	53.385299	88.3	198.10	303.8
NM	62.0	212.193548	46.616795	122.9	211.40	327.1
NV	66.0	202.915152	50.859687	106.1	196.45	339.9
NY	83.0	196.993976	58.252500	64.3	193.00	332.8
OH	78.0	206.441026	49.864079	61.9	210.50	301.5
OK	61.0	193.018033	57.107987	42.2	195.00	317.0
OR	78.0	201.496154	47.566428	75.9	205.85	317.5
PA	45.0	191.653333	45.323271	101.5	186.60	294.3
RI	65.0	211.038462	49.939531	102.2	207.20	350.5
SC	60.0	207.456667	49.462711	83.4	213.90	304.6
SD	60.0	202.723333	45.108319	118.7	193.15	295.7
TN	53.0	210.513208	45.766754	93.4	206.90	296.5
TX	72.0	199.787500	51.865041	49.2	194.80	327.0
UT	72.0	195.343056	51.600630	0.0	197.90	319.0
VA	77.0	204.216883	44.592478	91.2	205.20	313.4
VT	73.0	205.368493	48.275120	71.0	209.90	304.9
WA	66.0	203.810606	49.061353	52.9	203.45	285.9
WI	78.0	197.458974	53.884652	97.7	193.80	322.3
WV	106.0	188.413208	52.769371	67.0	185.60	315.4
WY	77.0	205.828571	52.124729	60.0	212.80	330.6

	Total night minutes					
	count	mean	std	min	50%	max
State						
AK	52.0	192.326923	53.663297	23.2	200.50	303.5
AL	80.0	187.285000	42.355416	94.1	185.40	287.6
AR	55.0	205.454545	56.696498	96.4	205.50	367.7
AZ	64.0	194.004687	54.387325	77.3	191.40	297.9
CA	34.0	198.508824	58.502574	71.1	204.90	345.8
CO	66.0	189.898485	50.131426	83.9	193.45	308.2
CT	74.0	205.997297	59.124952	63.6	205.95	325.6
DC	54.0	206.348148	48.437521	95.3	210.85	302.0
DE	61.0	203.900000	43.651063	111.7	200.90	313.2
FL	63.0	196.147619	50.501983	73.2	197.40	286.3
GA	54.0	193.746296	56.564876	65.8	186.15	364.9
HI	53.0	203.713208	43.058596	112.3	203.30	318.3
IA	44.0	191.490909	48.004466	43.7	189.90	271.8
ID	73.0	202.595890	49.006208	89.3	200.40	377.5
IL	58.0	197.605172	46.280500	67.7	192.70	281.9
IN	71.0	210.242254	52.449659	99.0	213.40	350.2
KS	70.0	203.970000	46.138045	50.1	208.95	289.9
KY	59.0	198.355932	53.771045	79.9	202.10	314.1
LA	51.0	201.396078	51.633872	103.7	196.00	352.2

```
MA                  65.0  204.007692  55.335701   72.4  204.80  332.7
MD                  70.0  198.614286  51.707685   76.4  194.00  294.8
ME                  62.0  198.833871  48.369012  114.2  193.85  313.4
MI                  73.0  192.657534  47.490048   63.3  195.20  302.2
MN                  84.0  209.680952  45.807222   77.2  201.30  296.3
MO                  63.0  209.146032  54.135820   50.1  210.00  349.7
MS                  65.0  200.996923  41.684953  112.9  203.40  309.6
MT                  68.0  197.205882  45.569900   87.4  194.85  328.5
NC                  68.0  197.144118  51.714106   54.5  198.40  329.3
ND                  62.0  199.796774  53.604532   61.4  207.00  281.3
NE                  61.0  206.427869  60.281739   82.4  211.60  381.9
NH                  56.0  208.614286  48.423373  107.3  214.00  334.7
NJ                  68.0  206.383824  51.385458   57.5  210.65  309.1
NM                  62.0  200.193548  53.324426   82.3  209.00  310.1
NV                  66.0  208.645455  43.134524  126.3  206.55  320.7
NY                  83.0  203.268675  47.042381   94.9  200.80  323.5
OH                  78.0  204.491026  55.067958   64.2  204.90  352.5
OK                  61.0  196.947541  50.258414   47.4  192.50  312.1
OR                  78.0  199.925641  51.042301   53.3  199.55  322.2
PA                  45.0  195.864444  53.891090   56.6  186.90  342.8
RI                  65.0  204.052308  56.367225   91.6  200.60  315.0
SC                  60.0  195.136667  46.713639   73.2  186.00  325.9
SD                  60.0  201.310000  45.437361  116.3  198.45  303.5
TN                  53.0  210.426415  51.282487  104.7  219.50  317.8
TX                  72.0  195.288889  40.665577  115.9  193.10  306.6
UT                  72.0  190.519444  51.344801   90.9  188.20  344.3
VA                  77.0  212.963636  57.260802   54.0  214.70  395.0
VT                  73.0  206.989041  48.835198   78.1  205.70  333.5
WA                  66.0  200.045455  52.908182   84.8  203.95  304.2
WI                  78.0  199.229487  56.392127   77.9  192.55  364.3
WV                 106.0  201.055660  50.275578   75.8  202.55  326.4
WY                  77.0  199.167532  45.849655   45.0  203.30  285.3
```

Источники данных поддерживаемые в Pandas

Помимо csv, текстовых файлов и html Pandas поддерживает большое количество различных источников:

- json
- excel
- sql
- parquet
- google-bigquery

```
pd.read_json('sample_data/anscombe.json').head(10)
```

```
  Series   X      Y
0      I  10   8.04
1      I   8   6.95
2      I  13   7.58
3      I   9   8.81
4      I  11   8.33
5      I  14   9.96
6      I   6   7.24
7      I   4   4.26
```

```
8       I  12  10.84
9       I   7   4.81
```

Чтение из SQL баз данных

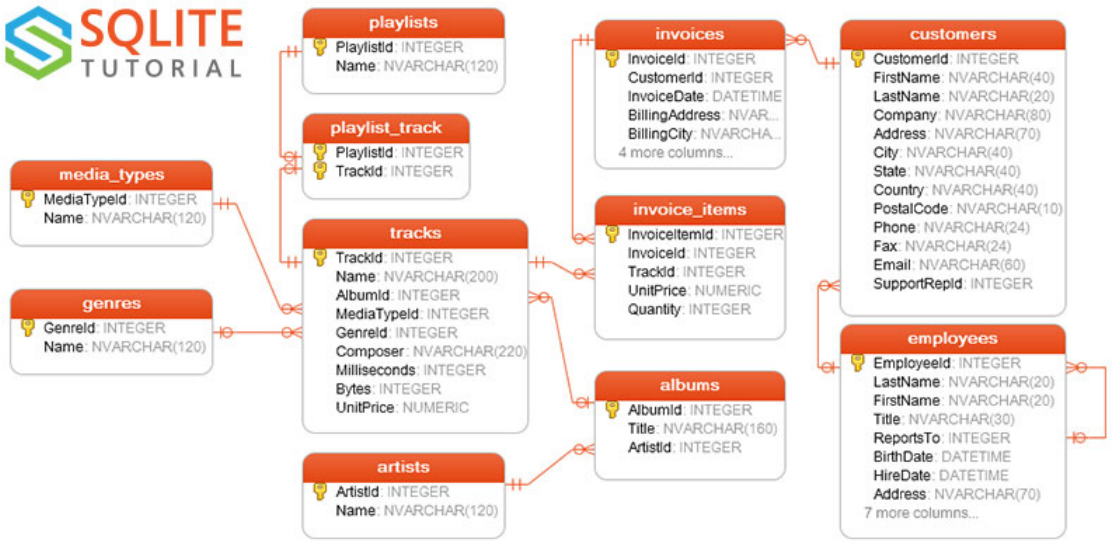
Загрузить данные из SQL базы можно с помощью функции pd.read_sql. read_sql автоматически преобразует столбцы SQL в столбцы DataFrame.

read_sql принимает 2 аргумента: запрос SELECT, и connection. Это здорово, так как это означает, что можно читать из любого вида базы данных - неважно, MySQL, SQLite, PostgreSQL, или другая.

В этом примере мы читаем из базы SQLite, но другие читаются точно также. БД возьмём из официального руководства.

```
#!wget
'https://github.com/jvns/pandas-cookbook/raw/master/data/weather_2012.sqlite'
-O /content/sample_data/weather_2012.sqlite -q
!wget https://www.sqlitetutorial.net/wp-content/uploads/2018/03/chinook.zip
-O 'sample_data/chinook.zip' -q
!unzip "sample_data/chinook.zip" -d "sample_data/"
```

```
Archive:  sample_data/chinook.zip
  inflating: sample_data/chinook.db
```




```
import pandas as pd
from sqlalchemy import create_engine

engine = create_engine("sqlite:///sample_data/chinook.db")

with engine.connect() as conn, conn.begin():
    employees = pd.read_sql_table("employees", conn)
    customers = pd.read_sql("SELECT * FROM customers LIMIT 20", conn)
employees.head()
```

```
   EmployeeId LastName FirstName                Title  ReportsTo  BirthDate
\
```

```
0           1    Adams    Andrew      General Manager        NaN 1962-02-18
1           2  Edwards     Nancy        Sales Manager        1.0 1958-12-08
2           3  Peacock      Jane  Sales Support Agent        2.0 1973-08-29
3           4     Park  Margaret  Sales Support Agent        2.0 1947-09-19
4           5  Johnson     Steve  Sales Support Agent        2.0 1965-03-03

    HireDate              Address      City State Country PostalCode  \
0 2002-08-14  11120 Jasper Ave NW  Edmonton    AB  Canada    T5K 2N1
1 2002-05-01          825 8 Ave SW   Calgary    AB  Canada    T2P 2T3
2 2002-04-01         1111 6 Ave SW   Calgary    AB  Canada    T2P 5M5
3 2003-05-03     683 10 Street SW   Calgary    AB  Canada    T2P 5G3
4 2003-10-17          7727B 41 Ave   Calgary    AB  Canada    T3B 1Y7

               Phone                Fax                    Email
0  +1 (780) 428-9482  +1 (780) 428-3457    andrew@chinookcorp.com
1  +1 (403) 262-3443  +1 (403) 262-3322     nancy@chinookcorp.com
2  +1 (403) 262-3443  +1 (403) 262-6712      jane@chinookcorp.com
3  +1 (403) 263-4423  +1 (403) 263-4289  margaret@chinookcorp.com
4   1 (780) 836-9987   1 (780) 836-9543     steve@chinookcorp.com

customers.head()

   CustomerId  FirstName     LastName  \
0           1       Luís    Gonçalves
1           2     Leonie       Köhler
2           3   François     Tremblay
3           4      Bjørn       Hansen
4           5  František  Wichterlová

                                          Company  \
0  Embraer - Empresa Brasileira de Aeronáutica S.A.
1                                              None
2                                              None
3                                              None
4                                JetBrains s.r.o.

                          Address                 City State         Country
\
0  Av. Brigadeiro Faria Lima, 2170  São José dos Campos    SP          Brazil
1          Theodor-Heuss-Straße 34            Stuttgart  None         Germany
2                1498 rue Bélanger             Montréal    QC          Canada
3                 Ullevålsveien 14                 Oslo  None          Norway
4                    Klanova 9/506               Prague  None  Czech Republic

  PostalCode               Phone                 Fax  \
0  12227-000  +55 (12) 3923-5555  +55 (12) 3923-5566
1      70174    +49 0711 2842222                None
2    H2G 1A7   +1 (514) 721-4711                None
3       0171     +47 22 44 22 22                None
4      14700    +420 2 4172 5555    +420 2 4172 5555

                      Email  SupportRepId
0      luisg@embraer.com.br             3
1     leonekohler@surfeu.de             5
2       ftremblay@gmail.com             3
```

```
3     bjorn.hansen@yahoo.no             4
4  frantisekw@jetbrains.com             4
```

read_sql не устанавливает первичный ключ (id) в качестве индекса. Можно это сделать вручную, добавив аргумент index_col к read_sql.

Если вы много использовали read_csv, вы могли заметить, что у него также есть аргумент index_col. И ведёт он себя точно так же.

```
with engine.connect() as conn, conn.begin():
    employees = pd.read_sql_table("employees", conn, index_col='EmployeeId')
    customers = pd.read_sql("SELECT * FROM customers LIMIT 20", conn,
index_col='SupportRepId')
customers.head()
```

```
              CustomerId  FirstName     LastName  \
SupportRepId
3                      1       Luís    Gonçalves
5                      2     Leonie       Köhler
3                      3   François     Tremblay
4                      4      Bjørn       Hansen
4                      5  František  Wichterlová

                                                 Company  \
SupportRepId
3             Embraer - Empresa Brasileira de Aeronáutica S.A.
5                                                     None
3                                                     None
4                                                     None
4                                          JetBrains s.r.o.

                                    Address                 City State  \
SupportRepId
3             Av. Brigadeiro Faria Lima, 2170  São José dos Campos    SP
5                    Theodor-Heuss-Straße 34            Stuttgart  None
3                          1498 rue Bélanger             Montréal    QC
4                           Ullevålsveien 14                 Oslo  None
4                              Klanova 9/506               Prague  None

                     Country PostalCode               Phone  \
SupportRepId
3                     Brazil  12227-000  +55 (12) 3923-5555
5                    Germany      70174    +49 0711 2842222
3                     Canada    H2G 1A7   +1 (514) 721-4711
4                     Norway       0171     +47 22 44 22 22
4             Czech Republic      14700    +420 2 4172 5555

                           Fax                    Email
SupportRepId
3             +55 (12) 3923-5566      luisg@embraer.com.br
5                           None     leonekohler@surfeu.de
3                           None       ftremblay@gmail.com
4                           None     bjorn.hansen@yahoo.no
4               +420 2 4172 5555  frantisekw@jetbrains.com
```

Запись в базу

Поддерживается использование sqlite без использования SQLAlchemy. Для этого режима требуется адаптер базы данных Python, который поддерживает Python DB-API. Вы можете создавать такие подключения: Запись производится с помощью метода to_sql (по аналогии с CSV):

```
import sqlite3
con = sqlite3.connect("sample_data/test_db.sqlite")
con.execute("DROP TABLE IF EXISTS employees")
employees.to_sql("employees", con)
```

```
!ls sample_data/test_db.sqlite
```

```
sample_data/test_db.sqlite
```

Теперь мы можем загрузить записанные данные:

```
con = sqlite3.connect("sample_data/test_db.sqlite")
df = pd.read_sql("SELECT * FROM employees LIMIT 5", con)
df
```

```
   EmployeeId LastName FirstName               Title  ReportsTo  \
0           1    Adams    Andrew     General Manager        NaN
1           2  Edwards     Nancy       Sales Manager        1.0
2           3  Peacock      Jane  Sales Support Agent        2.0
3           4     Park  Margaret  Sales Support Agent        2.0
4           5  Johnson     Steve  Sales Support Agent        2.0

             BirthDate             HireDate              Address      City  \
0  1962-02-18 00:00:00  2002-08-14 00:00:00  11120 Jasper Ave NW  Edmonton
1  1958-12-08 00:00:00  2002-05-01 00:00:00         825 8 Ave SW   Calgary
2  1973-08-29 00:00:00  2002-04-01 00:00:00        1111 6 Ave SW   Calgary
3  1947-09-19 00:00:00  2003-05-03 00:00:00     683 10 Street SW   Calgary
4  1965-03-03 00:00:00  2003-10-17 00:00:00         7727B 41 Ave   Calgary

  State Country PostalCode              Phone                Fax  \
0    AB  Canada    T5K 2N1  +1 (780) 428-9482  +1 (780) 428-3457
1    AB  Canada    T2P 2T3  +1 (403) 262-3443  +1 (403) 262-3322
2    AB  Canada    T2P 5M5  +1 (403) 262-3443  +1 (403) 262-6712
3    AB  Canada    T2P 5G3  +1 (403) 263-4423  +1 (403) 263-4289
4    AB  Canada    T3B 1Y7   1 (780) 836-9987   1 (780) 836-9543

                     Email
0    andrew@chinookcorp.com
1     nancy@chinookcorp.com
2      jane@chinookcorp.com
3  margaret@chinookcorp.com
4     steve@chinookcorp.com
```

Главное преимущество хранения данных в базе в том, что можно напрямую делать SQL запросы. Это особенно хорошо, если SQL для вас более родной язык. Например, можно отсортировать по колонке 'Weather' с помощью лишь SQL:

```
df = pd.read_sql("SELECT * FROM employees ORDER BY HireDate LIMIT 5", con)
df
```

```
   EmployeeId LastName FirstName               Title  ReportsTo  \
0           3  Peacock      Jane  Sales Support Agent        2.0
```

```
1           2  Edwards     Nancy        Sales Manager        1.0
2           1    Adams    Andrew      General Manager        NaN
3           4     Park  Margaret  Sales Support Agent        2.0
4           5  Johnson     Steve  Sales Support Agent        2.0

             BirthDate             HireDate              Address      City  \
0  1973-08-29 00:00:00  2002-04-01 00:00:00        1111 6 Ave SW   Calgary
1  1958-12-08 00:00:00  2002-05-01 00:00:00         825 8 Ave SW   Calgary
2  1962-02-18 00:00:00  2002-08-14 00:00:00  11120 Jasper Ave NW  Edmonton
3  1947-09-19 00:00:00  2003-05-03 00:00:00     683 10 Street SW   Calgary
4  1965-03-03 00:00:00  2003-10-17 00:00:00         7727B 41 Ave   Calgary

  State Country PostalCode              Phone                Fax  \
0    AB  Canada    T2P 5M5  +1 (403) 262-3443  +1 (403) 262-6712
1    AB  Canada    T2P 2T3  +1 (403) 262-3443  +1 (403) 262-3322
2    AB  Canada    T5K 2N1  +1 (780) 428-9482  +1 (780) 428-3457
3    AB  Canada    T2P 5G3  +1 (403) 263-4423  +1 (403) 263-4289
4    AB  Canada    T3B 1Y7   1 (780) 836-9987   1 (780) 836-9543

                     Email
0      jane@chinookcorp.com
1     nancy@chinookcorp.com
2    andrew@chinookcorp.com
3  margaret@chinookcorp.com
4     steve@chinookcorp.com
```

Не забываем отключаться от БД.

```
con.close()
```

Выводы

1. Pandas мощная библиотека для обработки данных из стандартных форматов. Позволяющая выполнять очистку, обогащение, группировку, изучение и сохранение данных.
2. Все операции производятся в памяти над объектами кадров данных и последовательностей.
3. Если вы только начинающий инженер данных или готовите небольшую выборку для прототипирования (этап разработки модели), то знания Pandas могут очень ускорить вашу работу.

3. Основы работы с Apache Spark

3.1. Установка Apache Spark

Скачиваем и устанавливаем необходимое ПО

```
# Устанавливаем OpenJDK
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
# Закачиваем Spark
!wget -q
```

```
http://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-bin-hadoop2.7.tg
z -O spark.tgz
# Распаковываем архив со Spark
!tar xf spark.tgz
# Устанавливаем пакет findspark для работы со Spark из Python
!pip install -q findspark

!ls
```

```
sample_data  spark-3.2.0-bin-hadoop2.7    spark.tgz
```

Настраиваем переменные окружения для работы с Apache Spark

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.2.0-bin-hadoop2.7"
```

Создание сессии Spark

Находим установку Spark

```
import findspark
findspark.init()
```

Подключаем необходимые модули для работы со Spark из Python

```
from pyspark.sql import SparkSession
```

Создаем сессию Spark на локальном компьютере

```
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

```
!echo 'apple' > sample_data/fruits.txt
!echo 'banana' >> sample_data/fruits.txt
!echo 'canary melon' >> sample_data/fruits.txt
!echo 'grape' >> sample_data/fruits.txt
!echo 'lemon' >> sample_data/fruits.txt
!echo 'orange' >> sample_data/fruits.txt
!echo 'pineapple' >> sample_data/fruits.txt
!echo 'strawberry' >> sample_data/fruits.txt
!echo 'banana' > sample_data/yellowthings.txt
!echo 'bee' >> sample_data/yellowthings.txt
!echo 'butter' >> sample_data/yellowthings.txt
!echo 'canary melon' >> sample_data/yellowthings.txt
!echo 'gold' >> sample_data/yellowthings.txt
!echo 'lemon' >> sample_data/yellowthings.txt
!echo 'pineapple' >> sample_data/yellowthings.txt
!echo 'sunflower' >> sample_data/yellowthings.txt
```

3.2. Общие сведения

Apache Spark - это проект с открытым исходным кодом предоставляющий фреймворк для параллельной обработки, поддерживающий вычисления в оперативной памяти для повышенной производительности в приложениях аналитики больших данных.

Apache Spark - это:

- Платформа для построения распределнных приложений обработки данных
- Эволюция Hadoop MapReduce
- Библиотека под Python/Scala
- Один из самых популярных проектов в области обработки больших данных

Область применения

- Распределенная обработка больших данных
- Построение конвееров обработки данных (Extract Transform Load pipelines)
- Работа со структурированными данными (SQL)
- Разработка потоковых приложений (Near Real Time streaming)

Архитектура приложения

Driver Program (also Master):

- предоставляет API через SparkSession и SparkContext
- выполняет ваш код - python файл или скомпилированный .jar
- контролирует выполнение задачи

Executors (also Workers or Slaves):

- обрабатывают данные
- каждый Worker работает со своим сегментом данных - Partition
- не выполняются ваш код напрямую
- получают задачи от Driver

Cluster Manager (YARN/Mesos):

- отвечает за выделение(аллокацию) контейнеров, выполняющих код драйвера и исполнителей, на кластере
- квотирует и распределяет ресурсы между пользователями
- контролирует состояние контейнеров

3.3. Устойчивые распределенные наборы данных

Приложения для больших данных полагаются на итеративные распределенные вычисления для более быстрой обработки больших наборов данных. Чтобы распределить обработку данных по нескольким заданиям, данные обычно повторно используются или совместно используются между заданиями. Чтобы обмениваться данными между существующими распределенными вычислительными системами, вам необходимо хранить данные в некотором промежуточном стабильном распределенном хранилище, таком как HDFS. Это в целом замедляет вычисления. Устойчивые распределенные наборы данных или RDD решают эту проблему, обеспечивая отказоустойчивые, распределенные вычисления в памяти.

Resilent Distributed Dataset (RDD) - самая базовая и самая низкоуровневая структура в Spark, доступная разработчику. Представляет собой типизированную неизменяемую неупорядоченную партиционированную коллекцию данных, распределенную по узлам кластера.

RDD может быть создана из:

- локальной коллекции на драйвере
- файла (локального или на распределенной файловой системе, например HDFS)
- базы данных

```
# локальный список python
cities = ["Yekaterinburg", "Moscow", "Paris", "Madrid", "London", "New York",
"Dubai"]
print('The list has %s elements, the first one is "%s" and the last one is
"%s".' %(len(cities), cities[0], cities[-1]))
```

```
The list has 7 elements, the first one is "Yekaterinburg" and the last one is
"Dubai".
```

```
# RDD "Города"
citiesRDD = spark.sparkContext.parallelize(cities)
print('The RDD has %s elements, the first one is %s' %(citiesRDD.count(),
citiesRDD.take(2)))
```

```
The RDD has 7 elements, the first one is ['Yekaterinburg', 'Moscow']
```

```
# RDD "Фрукты"
fruits = spark.sparkContext.textFile('sample_data/fruits.txt')
print('The RDD has %s elements, the first one is %s' %(fruits.count(),
fruits.take(1)))
```

```
The RDD has 8 elements, the first one is ['apple']
```

```
# RDD "Желтые вещи"
yellowThings = spark.sparkContext.textFile('sample_data/yellowthings.txt')
print('The RDD has %s elements, the first one is %s' %(yellowThings.count(),
yellowThings.take(1)))
```

```
The RDD has 8 elements, the first one is ['banana']
```

3.4. Операции с RDD

1. Трансформации (e.g. map, filter)
2. Действия (e.g. reduce, collect, count, foreach)

Трансформации (Transormations):

- всегда превращают один RDD в новый RDD
- всегда являются ленивыми - создают граф вычислений, но не запускают их
- иногда (часто) неявно требуют перемешивания данных между исполнителями - **Shuffle**

Ниже приведены примеры некоторых доступных преобразований. Подробный список см. RDD Transformations

```
# Трансформация map: не запускает вычислений, не изменяет изначальный RDD
rddUpper = citiesRDD.map(lambda city: city.upper())

# Метод take (является действием) возвращает N первых элементов RDD
upperVec = rddUpper.take(3)
oldVec = citiesRDD.take(3)
```

```
# метод join позволяет сделать из любой локальной коллекции строку
print('New RDD: %s'%(', '.join(str(e) for e in upperVec)))
print('Old RDD: %s'%(', '.join(str(e) for e in oldVec)))
```

```
New RDD: YEKATERINBURG, MOSCOW, PARIS
Old RDD: Yekaterinburg, Moscow, Paris
```

```
def rdd_show(rdd): #обьявим вспомогательную функцию по выводу на экран
    for i in rdd.collect(): # Метод collect является действием
        print(i)
def rdd_showF(rdd): #обьявим вспомогательную функцию по выводу на экран
    for i in rdd.take(10): # Метод take является действием
        print(i)

# map (Отображение)
def reverse(inp):
  return inp[::-1]
fruitsReversed = fruits.map(reverse)
rdd_show(fruitsReversed)
```

```
elppa
ananab
nolem yranac
eparg
nomel
egnaro
elppaenip
yrrebwarts
```

```
# filter (Фильтрация)
startsWithM = rddUpper.filter(lambda x: x.startswith("M"))
print('The following city names starts with M: %s' %(', '.join(str(e) for e
in startsWithM.take(2))))
```

```
The following city names starts with M: MOSCOW, MADRID
```

```
# filter (Фильтрация)
shortFruits = fruits.filter(lambda fruit: len(fruit) <= 5)
rdd_show(shortFruits)
```

```
apple
grape
lemon
```

```
# flatMap (Спрямление вложенных коллекций)
mappedRdd = citiesRDD.map(lambda x: list(x))
rdd_showF(mappedRdd)
print("####")
flatMappedRdd = citiesRDD.flatMap(lambda x: x.lower())
rdd_showF(flatMappedRdd)
```

```
['Y', 'e', 'k', 'a', 't', 'e', 'r', 'i', 'n', 'b', 'u', 'r', 'g']
['M', 'o', 's', 'c', 'o', 'w']
['P', 'a', 'r', 'i', 's']
['M', 'a', 'd', 'r', 'i', 'd']
['L', 'o', 'n', 'd', 'o', 'n']
['N', 'e', 'w', ' ', 'Y', 'o', 'r', 'k']
```

```
['D', 'u', 'b', 'a', 'i']
####
y
e
k
a
t
e
r
i
n
b
```

```
# flatMap (Спрямление вложенных коллекций)
characters = fruits.flatMap(lambda fruit: list(fruit))
rdd_showF(characters)
```

```
a
p
p
l
e
b
a
n
a
n
```

```
# union (Объединение)
fruitsAndYellowThings = fruits.union(yellowThings)
rdd_show(fruitsAndYellowThings)
```

```
apple
banana
canary melon
grape
lemon
orange
pineapple
strawberry
banana
bee
butter
canary melon
gold
lemon
pineapple
sunflower
```

```
# intersection (Пересечение)
yellowFruits = fruits.intersection(yellowThings)
rdd_show(yellowFruits)
```

```
pineapple
canary melon
lemon
banana
```

```
# distinct (Удаление дубликатов)
distinctFruitsAndYellowThings = fruitsAndYellowThings.distinct()
rdd_show(distinctFruitsAndYellowThings)
```

```
orange
pineapple
canary melon
grape
lemon
bee
banana
butter
gold
sunflower
apple
strawberry
```

```
letters = flatMappedRdd \
    .distinct() \
    .filter(lambda x : x != ' ') \
    .collect()
letters.sort()
uniqueLetters = ', '.join(str(e) for e in letters)

print('Letters in the RDD are: %s' %(uniqueLetters))
```

```
Letters in the RDD are: a, b, c, d, e, g, i, k, l, m, n, o, p, r, s, t, u, w,
y
```

```
# reduceByKey (Редукция по ключам)
numFruitsByLength = fruits.map(lambda fruit: (len(fruit),
1)).reduceByKey(lambda x, y: 10*x + y)
rdd_show(numFruitsByLength)
```

```
(6, 11)
(12, 1)
(10, 1)
(5, 111)
(9, 1)
```

```
# groupByKey (Группировка по ключам)
yellowThingsByFirstLetter = yellowThings.map(lambda thing: (thing[0],
thing)).groupByKey()
rdd_show(yellowThingsByFirstLetter.map(lambda x:(x[0],list(x[1]))))
```

```
('b', ['banana', 'bee', 'butter'])
('c', ['canary melon'])
('g', ['gold'])
('l', ['lemon'])
('p', ['pineapple'])
('s', ['sunflower'])
```

Действия (Actions):

- выполняют действие над RDD
- запускают вычисления

Ниже приведены примеры некоторых доступных общих действий. Подробный список см. RDD Actions.

```
# Действие reduce применяет функцию f к промежуточному результату
# от предыдущей итерации со следующим элементом коллекции
count = citiesRDD.map(lambda x: len(x) ).reduce(lambda x,y: x+y)
print('The RDD contains %s letters' %(count))
```

```
The RDD contains 49 letters
```

```
# reduce (Редукция)
letterSet = fruits.map(lambda fruit: set(fruit)).reduce(lambda x, y:
x.union(y))
letterSet
```

```
{' ',
 'a',
 'b',
 'c',
 'e',
 'g',
 'i',
 'l',
 'm',
 'n',
 'o',
 'p',
 'r',
 's',
 't',
 'w',
 'y'}
```

```
# collect (Передача ВСЕХ элементов RDD на драйвер)
fruitsArray = fruits.collect()
yellowThingsArray = yellowThings.collect()
fruitsArray
```

```
['apple',
 'banana',
 'canary melon',
 'grape',
 'lemon',
 'orange',
 'pineapple',
 'strawberry']
```

```
# collect (Передача ВСЕХ элементов RDD на драйвер)
localArray = startsWithM.collect()
containsMoscow = "MOSCOW" in localArray
print('The array contains MOSCOW: %s'%(containsMoscow))
```

```
The array contains MOSCOW: True
```

```
# count (Подсчет количества элементов в RDD)
countM = startsWithM.count()
print('The RDD contains %s elements'%(countM))
```

```
The RDD contains 2 elements
```

```
# count (Подсчет количества элементов в RDD)
numFruits = fruits.count()
numFruits
```

```
8
```

```
# take (Передача N элементов по сети на драйвер)
twoElements = startsWithM.take(2)
print('Two elements of the RDD are: %s'%(', '.join(str(e) for e in
twoElements)))
```

```
Two elements of the RDD are: MOSCOW, MADRID
```

```
# take (Передача N элементов по сети на драйвер)
first3Fruits = fruits.take(3)
first3Fruits
```

```
['apple', 'banana', 'canary melon']
```

```
# takeOrdered (Передача ВСЕХ элементов RDD на драйвер, сортировка и выборка
из N первых элементов)
twoElementsSorted = startsWithM.takeOrdered(2)
print('Two elements of the RDD are: %s'%(', '.join(str(e) for e in
twoElementsSorted)))
```

```
Two elements of the RDD are: MADRID, MOSCOW
```

Выводы

- RDD - это неизменяемый распределенный набор данных
- Трансформации (map, filter, flatMap) создают новый RDD из существующего и не изменяют существующий
- Любые трансформации являются ленивыми и не запускают вычислений
- Действия (count, reduce, collect, take) запускают вычисления

Полезные ссылки:

- RDD API Reference
- RDD Programming guide
- Scala 2.11.12 API

3.5. PairRDD функции

Во всех вышестоящих экспериментах мы создавали RDD, состоящие из элементов базовых типов - числовых, срок и символов. На самом деле RDD не имеют как таковых органичений на тип элементов. Ими могут выступать коллекции, кейс классы, кортежи и т.д.

PairRDD - расширенный класс функций, доступных для RDD, где элементы - это кортеж (key, value)

```
pairRdd = citiesRDD.flatMap(lambda x : x.lower()).map(lambda x : (x, 1))
rdd_showF(pairRdd)
```

```
('y', 1)
('e', 1)
('k', 1)
('a', 1)
('t', 1)
('e', 1)
('r', 1)
('i', 1)
('n', 1)
('b', 1)
```

countByKey подсчитывает количество кортежей по каждому ключу и возвращает локальный Map

```
letterCount = pairRdd.countByKey()
for (i,j) in letterCount.items():
    print('%s -> %s'%(i,j))
```

```
y -> 2
e -> 3
k -> 2
a -> 4
t -> 1
r -> 5
i -> 4
n -> 4
b -> 2
u -> 2
g -> 1
m -> 2
o -> 5
s -> 2
c -> 1
w -> 2
p -> 1
d -> 4
l -> 1
  -> 1
```

reduceByKey работает аналогично обычному reduce, но промежуточный итог накапливается по каждому ключу независимо

```
letterCount = pairRdd.reduceByKey(lambda x,y: x+y)

print('%s'%(', '.join(str(e) for e in letterCount.take(3))))
```

```
('y', 2), ('r', 5), ('i', 4)
```

Join позволяет соединить два RDD по ключу. Поддерживаются join, leftOuterJoin и fullOuterJoin

```
favouriteLetters = ['a', 'd', 'o']
favLetRdd = spark.sparkContext.parallelize(favouriteLetters).map(lambda x :
(x,1))

joined = letterCount.leftOuterJoin(favLetRdd)
for (letter, (cnt, k)) in joined.collect():
```

```
    if k is not None:
        print('The letter %s is my favourite and it appears in the RDD %s
times'%(letter, cnt))
    else:
        print('The letter %s is not my favourite!'%(letter))
```

```
The letter y is not my favourite!
The letter r is not my favourite!
The letter i is not my favourite!
The letter b is not my favourite!
The letter g is not my favourite!
The letter s is not my favourite!
The letter c is not my favourite!
The letter p is not my favourite!
The letter l is not my favourite!
The letter e is not my favourite!
The letter a is my favourite and it appears in the RDD 4 times
The letter m is not my favourite!
The letter w is not my favourite!
The letter d is my favourite and it appears in the RDD 4 times
The letter k is not my favourite!
The letter t is not my favourite!
The letter n is not my favourite!
The letter u is not my favourite!
The letter o is my favourite and it appears in the RDD 5 times
The letter   is not my favourite!
```

Выводы

- PairRDD функции - расширенный список функций, доступный для RDD, элементы которых являются кортежем (K, V)
- PairRDD позволяют соединять два RDD по ключу K

Полезные ссылки:

- PairRDD API Reference

3.6. Работа с данными

Для изучения структуры и вычислений RDD проведем анализ датасета Airport Codes

```
!wget https://datahub.io/core/airport-codes/r/airport-codes.csv -O
/content/sample_data/airport-codes.csv
rdd = spark.sparkContext.textFile("sample_data/airport-codes.csv")
```

```
--2022-09-26 13:59:52--
https://datahub.io/core/airport-codes/r/airport-codes.csv
Resolving datahub.io (datahub.io)... 188.114.96.0, 188.114.97.0,
2a06:98c1:3121::, ...
Connecting to datahub.io (datahub.io)|188.114.96.0|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:
https://pkgstore.datahub.io/core/airport-codes/airport-codes_csv/data/e07739e
49300d125989ee543d5598c4b/airport-codes_csv.csv [following]
--2022-09-26 13:59:54--
https://pkgstore.datahub.io/core/airport-codes/airport-codes_csv/data/e07739e
```

```
49300d125989ee543d5598c4b/airport-codes_csv.csv
Resolving pkgstore.datahub.io (pkgstore.datahub.io)... 188.114.97.0,
188.114.96.0, 2a06:98c1:3121::, ...
Connecting to pkgstore.datahub.io (pkgstore.datahub.io)|188.114.97.0|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 6232459 (5.9M) [text/csv]
Saving to: ‘/content/sample_data/airport-codes.csv’

/content/sample_dat 100%[===================>]   5.94M  13.4MB/s    in 0.4s

2022-09-26 13:59:55 (13.4 MB/s) - ‘/content/sample_data/airport-codes.csv’
saved [6232459/6232459]
```

Выведем первые 3 строки на экран:

```
for i in rdd.take(3):
    print(i)
```

```
ident,type,name,elevation_ft,continent,iso_country,iso_region,municipality,gp
s_code,iata_code,local_code,coordinates
00A,heliport,Total Rf
Heliport,11,NA,US,US-PA,Bensalem,00A,,00A,"-74.93360137939453,
40.07080078125"
00AA,small_airport,Aero B Ranch
Airport,3435,NA,US,US-KS,Leoti,00AA,,00AA,"-101.473911, 38.704022"
```

Подготовим частичный класс для парсинга данных

```
from pyspark.sql import Row
Airport = Row("ident",
              "type",
              "name",
              "elevationFt",
              "continent",
              "isoCountry",
              "isoRegion",
              "municipality",
              "gpsCode",
              "iataCode",
              "localCode",
              "longitude",
              "latitude")
```

Уберем шапку и ненужные кавычки

```
firstElem = rdd.first()

noHeader = rdd \
    .filter(lambda x : x != firstElem) \
    .map(lambda x : x.replace('"', ''))
print(noHeader.first())
```

```
00A,heliport,Total Rf
Heliport,11,NA,US,US-PA,Bensalem,00A,,00A,-74.93360137939453, 40.07080078125
```

Напишем функцию, которая преобразует RDD[String] => RDD[Airport]

```
def toAirport(data):
    r = Airport(*data.split(","))
    return r
```

Выполним преобразование RDD

```
airportRdd = noHeader.map(lambda x : toAirport(x))
```

Поскольку любые трансформации являются ленивыми, отсутствие ошибок при выполнении предыдущей ячейки еще не означает, что данная функция отрабатывает корректно на всем датасете. Проверим это с помощью операции count:

```
rdd_showF(airportRdd)
```

```
Row(ident='00A', type='heliport', name='Total Rf Heliport', elevationFt='11',
continent='NA', isoCountry='US', isoRegion='US-PA', municipality='Bensalem',
gpsCode='00A', iataCode='', localCode='00A', longitude='-74.93360137939453',
latitude=' 40.07080078125')
Row(ident='00AA', type='small_airport', name='Aero B Ranch Airport',
elevationFt='3435', continent='NA', isoCountry='US', isoRegion='US-KS',
municipality='Leoti', gpsCode='00AA', iataCode='', localCode='00AA',
longitude='-101.473911', latitude=' 38.704022')
Row(ident='00AK', type='small_airport', name='Lowell Field',
elevationFt='450', continent='NA', isoCountry='US', isoRegion='US-AK',
municipality='Anchor Point', gpsCode='00AK', iataCode='', localCode='00AK',
longitude='-151.695999146', latitude=' 59.94919968')
Row(ident='00AL', type='small_airport', name='Epps Airpark',
elevationFt='820', continent='NA', isoCountry='US', isoRegion='US-AL',
municipality='Harvest', gpsCode='00AL', iataCode='', localCode='00AL',
longitude='-86.77030181884766', latitude=' 34.86479949951172')
Row(ident='00AR', type='closed', name='Newport Hospital & Clinic Heliport',
elevationFt='237', continent='NA', isoCountry='US', isoRegion='US-AR',
municipality='Newport', gpsCode='', iataCode='', localCode='',
longitude='-91.254898', latitude=' 35.6087')
Row(ident='00AS', type='small_airport', name='Fulton Airport',
elevationFt='1100', continent='NA', isoCountry='US', isoRegion='US-OK',
municipality='Alex', gpsCode='00AS', iataCode='', localCode='00AS',
longitude='-97.8180194', latitude=' 34.9428028')
Row(ident='00AZ', type='small_airport', name='Cordes Airport',
elevationFt='3810', continent='NA', isoCountry='US', isoRegion='US-AZ',
municipality='Cordes', gpsCode='00AZ', iataCode='', localCode='00AZ',
longitude='-112.16500091552734', latitude=' 34.305599212646484')
Row(ident='00CA', type='small_airport', name='Goldstone /Gts/ Airport',
elevationFt='3038', continent='NA', isoCountry='US', isoRegion='US-CA',
municipality='Barstow', gpsCode='00CA', iataCode='', localCode='00CA',
longitude='-116.888000488', latitude=' 35.350498199499995')
Row(ident='00CL', type='small_airport', name='Williams Ag Airport',
elevationFt='87', continent='NA', isoCountry='US', isoRegion='US-CA',
municipality='Biggs', gpsCode='00CL', iataCode='', localCode='00CL',
longitude='-121.763427', latitude=' 39.427188')
Row(ident='00CN', type='heliport', name='Kitchen Creek Helibase Heliport',
elevationFt='3350', continent='NA', isoCountry='US', isoRegion='US-CA',
```

```
municipality='Pine Valley', gpsCode='00CN', iataCode='', localCode='00CN',
longitude='-116.4597417', latitude=' 32.7273736')

print(airportRdd.count())

---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-51-acac0166ae7b> in <module>
----> 1 print(airportRdd.count())

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py in count(self)
   1235         3
   1236         """
-> 1237         return self.mapPartitions(lambda i: [sum(1 for _ in
i)]).sum()
   1238
   1239     def stats(self):

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py in sum(self)
   1224         6.0
   1225         """
-> 1226         return self.mapPartitions(lambda x: [sum(x)]).fold(0,
operator.add)
   1227
   1228     def count(self):

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py in fold(self,
zeroValue, op)
   1078         # zeroValue provided to each partition is unique from the one
provided
   1079         # to the final reduce call
-> 1080         vals = self.mapPartitions(func).collect()
   1081         return reduce(op, vals, zeroValue)
   1082

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py in collect(self)
    948         """
    949         with SCCallSiteSync(self.context) as css:
--> 950             sock_info =
self.ctx._jvm.PythonRDD.collectAndServe(self._jrdd.rdd())
    951         return list(_load_from_socket(sock_info,
self._jrdd_deserializer))
    952

/content/spark-3.2.0-bin-hadoop2.7/python/lib/py4j-0.10.9.2-src.zip/py4j/java
_gateway.py in __call__(self, *args)
   1308         answer = self.gateway_client.send_command(command)
   1309         return_value = get_return_value(
-> 1310             answer, self.gateway_client, self.target_id, self.name)
   1311
   1312         for temp_arg in temp_args:

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/sql/utils.py in deco(*a,
**kw)
    109     def deco(*a, **kw):
    110         try:
```

```
--> 111             return f(*a, **kw)
    112         except py4j.protocol.Py4JJavaError as e:
    113             converted = convert_exception(e.java_exception)

/content/spark-3.2.0-bin-hadoop2.7/python/lib/py4j-0.10.9.2-src.zip/py4j/prot
ocol.py in get_return_value(answer, gateway_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling
z:org.apache.spark.api.python.PythonRDD.collectAndServe.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 0
in stage 57.0 failed 1 times, most recent failure: Lost task 0.0 in stage
57.0 (TID 105) (ab7ec29723d5 executor driver):
org.apache.spark.api.python.PythonException: Traceback (most recent call
last):
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/worker.py"
, line 619, in main
    process()
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/worker.py"
, line 609, in process
    out_iter = func(split_index, iterator)
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 417,
in func
    return f(iterator)
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 1237,
in <lambda>
    return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 1237,
in <genexpr>
    return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/util.py",
line 74, in wrapper
    return f(*args, **kwargs)
  File "<ipython-input-49-d0d658a9733f>", line 1, in <lambda>
  File "<ipython-input-48-a08f1e6365b8>", line 2, in toAirport
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/sql/types.
py", line 1551, in __call__
    "but got %s" % (self, len(self), args))
```

```
ValueError: Can not create Row with fields <Row('ident', 'type', 'name',
'elevationFt', 'continent', 'isoCountry', 'isoRegion', 'municipality',
'gpsCode', 'iataCode', 'localCode', 'longitude', 'latitude')>, expected 13
values but got ('03OI', 'heliport', 'Cleveland Clinic', ' Marymount Hospital
Heliport', '890', 'NA', 'US', 'US-OH', 'Garfield Heights', '03OI', '',
'03OI', '-81.599552', ' 41.420312')

      at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonExcep
tion(PythonRunner.scala:545)
      at
org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:703)
      at
org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:685)
      at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.hasNext(PythonRun
ner.scala:498)
      at
org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37
)
      at scala.collection.Iterator.foreach(Iterator.scala:943)
      at scala.collection.Iterator.foreach$(Iterator.scala:943)
      at
org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator.scala:28
)
      at scala.collection.generic.Growable.$plus$plus$eq(Growable.scala:62)
      at scala.collection.generic.Growable.$plus$plus$eq$(Growable.scala:53)
      at
scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:105)
      at
scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:49)
      at scala.collection.TraversableOnce.to(TraversableOnce.scala:366)
      at scala.collection.TraversableOnce.to$(TraversableOnce.scala:364)
      at
org.apache.spark.InterruptibleIterator.to(InterruptibleIterator.scala:28)
      at scala.collection.TraversableOnce.toBuffer(TraversableOnce.scala:358)
      at
scala.collection.TraversableOnce.toBuffer$(TraversableOnce.scala:358)
      at
org.apache.spark.InterruptibleIterator.toBuffer(InterruptibleIterator.scala:2
8)
      at scala.collection.TraversableOnce.toArray(TraversableOnce.scala:345)
      at scala.collection.TraversableOnce.toArray$(TraversableOnce.scala:339)
      at
org.apache.spark.InterruptibleIterator.toArray(InterruptibleIterator.scala:28
)
      at org.apache.spark.rdd.RDD.$anonfun$collect$2(RDD.scala:1030)
      at
org.apache.spark.SparkContext.$anonfun$runJob$5(SparkContext.scala:2254)
      at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
      at org.apache.spark.scheduler.Task.run(Task.scala:131)
      at
org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:5
06)
      at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1462)
```

```
	at
org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:509)
	at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:114
9)
	at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:62
4)
	at java.lang.Thread.run(Thread.java:750)

Driver stacktrace:
	at
org.apache.spark.scheduler.DAGScheduler.failJobAndIndependentStages(DAGSchedu
ler.scala:2403)
	at
org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2(DAGScheduler.sc
ala:2352)
	at
org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2$adapted(DAGSche
duler.scala:2351)
	at
scala.collection.mutable.ResizableArray.foreach(ResizableArray.scala:62)
	at
scala.collection.mutable.ResizableArray.foreach$(ResizableArray.scala:55)
	at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffer.scala:49)
	at
org.apache.spark.scheduler.DAGScheduler.abortStage(DAGScheduler.scala:2351)
	at
org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1(DAGSch
eduler.scala:1109)
	at
org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1$adapte
d(DAGScheduler.scala:1109)
	at scala.Option.foreach(Option.scala:407)
	at
org.apache.spark.scheduler.DAGScheduler.handleTaskSetFailed(DAGScheduler.scal
a:1109)
	at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.doOnReceive(DAGSchedu
ler.scala:2591)
	at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGSchedule
r.scala:2533)
	at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGSchedule
r.scala:2522)
	at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:49)
	at
org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:898)
	at org.apache.spark.SparkContext.runJob(SparkContext.scala:2214)
	at org.apache.spark.SparkContext.runJob(SparkContext.scala:2235)
	at org.apache.spark.SparkContext.runJob(SparkContext.scala:2254)
	at org.apache.spark.SparkContext.runJob(SparkContext.scala:2279)
	at org.apache.spark.rdd.RDD.$anonfun$collect$1(RDD.scala:1030)
	at
```

```
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151
)
      at
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112
)
      at org.apache.spark.rdd.RDD.withScope(RDD.scala:414)
      at org.apache.spark.rdd.RDD.collect(RDD.scala:1029)
      at
org.apache.spark.api.python.PythonRDD$.collectAndServe(PythonRDD.scala:180)
      at
org.apache.spark.api.python.PythonRDD.collectAndServe(PythonRDD.scala)
      at sun.reflect.GeneratedMethodAccessor58.invoke(Unknown Source)
      at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
java:43)
      at java.lang.reflect.Method.invoke(Method.java:498)
      at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
      at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
      at py4j.Gateway.invoke(Gateway.java:282)
      at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
      at py4j.commands.CallCommand.execute(CallCommand.java:79)
      at
py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:182)
      at py4j.ClientServerConnection.run(ClientServerConnection.java:106)
      at java.lang.Thread.run(Thread.java:750)
Caused by: org.apache.spark.api.python.PythonException: Traceback (most
recent call last):
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/worker.py"
, line 619, in main
    process()
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/worker.py"
, line 609, in process
    out_iter = func(split_index, iterator)
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 417,
in func
    return f(iterator)
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 1237,
in <lambda>
    return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 1237,
in <genexpr>
    return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/util.py",
```

```
line 74, in wrapper
    return f(*args, **kwargs)
  File "<ipython-input-49-d0d658a9733f>", line 1, in <lambda>
  File "<ipython-input-48-a08f1e6365b8>", line 2, in toAirport
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/sql/types.
py", line 1551, in __call__
    "but got %s" % (self, len(self), args))
ValueError: Can not create Row with fields <Row('ident', 'type', 'name',
'elevationFt', 'continent', 'isoCountry', 'isoRegion', 'municipality',
'gpsCode', 'iataCode', 'localCode', 'longitude', 'latitude')>, expected 13
values but got ('03OI', 'heliport', 'Cleveland Clinic', ' Marymount Hospital
Heliport', '890', 'NA', 'US', 'US-OH', 'Garfield Heights', '03OI', '',
'03OI', '-81.599552', ' 41.420312')

      at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonExcep
tion(PythonRunner.scala:545)
      at
org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:703)
      at
org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:685)
      at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.hasNext(PythonRun
ner.scala:498)
      at
org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37
)
      at scala.collection.Iterator.foreach(Iterator.scala:943)
      at scala.collection.Iterator.foreach$(Iterator.scala:943)
      at
org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator.scala:28
)
      at scala.collection.generic.Growable.$plus$plus$eq(Growable.scala:62)
      at scala.collection.generic.Growable.$plus$plus$eq$(Growable.scala:53)
      at
scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:105)
      at
scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:49)
      at scala.collection.TraversableOnce.to(TraversableOnce.scala:366)
      at scala.collection.TraversableOnce.to$(TraversableOnce.scala:364)
      at
org.apache.spark.InterruptibleIterator.to(InterruptibleIterator.scala:28)
      at scala.collection.TraversableOnce.toBuffer(TraversableOnce.scala:358)
      at
scala.collection.TraversableOnce.toBuffer$(TraversableOnce.scala:358)
      at
org.apache.spark.InterruptibleIterator.toBuffer(InterruptibleIterator.scala:2
8)
      at scala.collection.TraversableOnce.toArray(TraversableOnce.scala:345)
      at scala.collection.TraversableOnce.toArray$(TraversableOnce.scala:339)
      at
org.apache.spark.InterruptibleIterator.toArray(InterruptibleIterator.scala:28
)
      at org.apache.spark.rdd.RDD.$anonfun$collect$2(RDD.scala:1030)
```

```
	at
org.apache.spark.SparkContext.$anonfun$runJob$5(SparkContext.scala:2254)
	at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
	at org.apache.spark.scheduler.Task.run(Task.scala:131)
	at
org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:5
06)
	at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1462)
	at
org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:509)
	at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:114
9)
	at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:62
4)
	... 1 more
```

Что произошло? count, как и любой action, запускает вычисление всех элементов в RDD. Если посмотреть стектрейс, мы увидим причину возникновения ошибки:

```
"but got %s" % (self, len(self), args))
```

Это означает, что размер массива, полученного после операции split, меньше количества переменных, которые мы указали в данной операции:

```
ValueError: Can not create Row with fields
<Row(ident, type, name, elevationFt, continent, isoCountry, isoRegion,
municipality, gpsCode, iataCode, localCode, longitude, latitude)>,
expected 13 values but got
(u'03OI', u'heliport', u'Cleveland Clinic', u' Marymount Hospital Heliport',
u'890', u'NA', u'US', u'US-OH', u'Garfield Heights', u'03OI', u'', u'03OI',
u'-81.599552', u' 41.420312')
```

Изменим код функции toAirport, чтобы решить данную проблему. Для простоты будем считать, что если в строке недостаточно элементов, то эту строку следует выкинуть (сделать None)

```
def toAirportOpt(data):
    lst = data.split(",")
    if (len(lst) == 13):
        r = Airport(*lst)
    else:
        r = None
    return r
```

Применим новую функцию к RDD:

```
airportOptRdd = noHeader.map(toAirportOpt)
```

Проверим корректность выполнения функции на первых трех элементах и на всем датасете:

```
rdd_showF(airportOptRdd)
airportOptRdd.count()
```

```
Row(ident='00A', type='heliport', name='Total Rf Heliport', elevationFt='11',
continent='NA', isoCountry='US', isoRegion='US-PA', municipality='Bensalem',
gpsCode='00A', iataCode='', localCode='00A', longitude='-74.93360137939453',
latitude=' 40.07080078125')
Row(ident='00AA', type='small_airport', name='Aero B Ranch Airport',
elevationFt='3435', continent='NA', isoCountry='US', isoRegion='US-KS',
municipality='Leoti', gpsCode='00AA', iataCode='', localCode='00AA',
longitude='-101.473911', latitude=' 38.704022')
Row(ident='00AK', type='small_airport', name='Lowell Field',
elevationFt='450', continent='NA', isoCountry='US', isoRegion='US-AK',
municipality='Anchor Point', gpsCode='00AK', iataCode='', localCode='00AK',
longitude='-151.695999146', latitude=' 59.94919968')
Row(ident='00AL', type='small_airport', name='Epps Airpark',
elevationFt='820', continent='NA', isoCountry='US', isoRegion='US-AL',
municipality='Harvest', gpsCode='00AL', iataCode='', localCode='00AL',
longitude='-86.77030181884766', latitude=' 34.86479949951172')
Row(ident='00AR', type='closed', name='Newport Hospital & Clinic Heliport',
elevationFt='237', continent='NA', isoCountry='US', isoRegion='US-AR',
municipality='Newport', gpsCode='', iataCode='', localCode='',
longitude='-91.254898', latitude=' 35.6087')
Row(ident='00AS', type='small_airport', name='Fulton Airport',
elevationFt='1100', continent='NA', isoCountry='US', isoRegion='US-OK',
municipality='Alex', gpsCode='00AS', iataCode='', localCode='00AS',
longitude='-97.8180194', latitude=' 34.9428028')
Row(ident='00AZ', type='small_airport', name='Cordes Airport',
elevationFt='3810', continent='NA', isoCountry='US', isoRegion='US-AZ',
municipality='Cordes', gpsCode='00AZ', iataCode='', localCode='00AZ',
longitude='-112.16500091552734', latitude=' 34.305599212646484')
Row(ident='00CA', type='small_airport', name='Goldstone /Gts/ Airport',
elevationFt='3038', continent='NA', isoCountry='US', isoRegion='US-CA',
municipality='Barstow', gpsCode='00CA', iataCode='', localCode='00CA',
longitude='-116.888000488', latitude=' 35.350498199499995')
Row(ident='00CL', type='small_airport', name='Williams Ag Airport',
elevationFt='87', continent='NA', isoCountry='US', isoRegion='US-CA',
municipality='Biggs', gpsCode='00CL', iataCode='', localCode='00CL',
longitude='-121.763427', latitude=' 39.427188')
Row(ident='00CN', type='heliport', name='Kitchen Creek Helibase Heliport',
elevationFt='3350', continent='NA', isoCountry='US', isoRegion='US-CA',
municipality='Pine Valley', gpsCode='00CN', iataCode='', localCode='00CN',
longitude='-116.4597417', latitude=' 32.7273736')

57421

airportRdd = noHeader.flatMap(toAirportOpt)

airportRdd.count()

---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-56-8993195ad010> in <module>
----> 1 airportRdd.count()

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py in count(self)
   1235         3
   1236         """
-> 1237         return self.mapPartitions(lambda i: [sum(1 for _ in
```

```
i)]).sum()
   1238
   1239     def stats(self):

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py in sum(self)
   1224         6.0
   1225         """
-> 1226         return self.mapPartitions(lambda x: [sum(x)]).fold(0,
operator.add)
   1227
   1228     def count(self):

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py in fold(self,
zeroValue, op)
   1078         # zeroValue provided to each partition is unique from the one
provided
   1079         # to the final reduce call
-> 1080         vals = self.mapPartitions(func).collect()
   1081         return reduce(op, vals, zeroValue)
   1082

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py in collect(self)
    948         """
    949         with SCCallSiteSync(self.context) as css:
--> 950             sock_info =
self.ctx._jvm.PythonRDD.collectAndServe(self._jrdd.rdd())
    951         return list(_load_from_socket(sock_info,
self._jrdd_deserializer))
    952

/content/spark-3.2.0-bin-hadoop2.7/python/lib/py4j-0.10.9.2-src.zip/py4j/java
_gateway.py in __call__(self, *args)
   1308         answer = self.gateway_client.send_command(command)
   1309         return_value = get_return_value(
-> 1310             answer, self.gateway_client, self.target_id, self.name)
   1311
   1312         for temp_arg in temp_args:

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/sql/utils.py in deco(*a,
**kw)
    109     def deco(*a, **kw):
    110         try:
--> 111             return f(*a, **kw)
    112         except py4j.protocol.Py4JJavaError as e:
    113             converted = convert_exception(e.java_exception)

/content/spark-3.2.0-bin-hadoop2.7/python/lib/py4j-0.10.9.2-src.zip/py4j/prot
ocol.py in get_return_value(answer, gateway_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling
```

```
z:org.apache.spark.api.python.PythonRDD.collectAndServe.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 1
in stage 60.0 failed 1 times, most recent failure: Lost task 1.0 in stage
60.0 (TID 111) (ab7ec29723d5 executor driver):
org.apache.spark.api.python.PythonException: Traceback (most recent call
last):
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/worker.py"
, line 619, in main
    process()
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/worker.py"
, line 609, in process
    out_iter = func(split_index, iterator)
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 417,
in func
    return f(iterator)
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 1237,
in <lambda>
    return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 1237,
in <genexpr>
    return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
TypeError: 'NoneType' object is not iterable

      at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonExcep
tion(PythonRunner.scala:545)
      at
org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:703)
      at
org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:685)
      at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.hasNext(PythonRun
ner.scala:498)
      at
org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37
)
      at scala.collection.Iterator.foreach(Iterator.scala:943)
      at scala.collection.Iterator.foreach$(Iterator.scala:943)
      at
org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator.scala:28
)
      at scala.collection.generic.Growable.$plus$plus$eq(Growable.scala:62)
      at scala.collection.generic.Growable.$plus$plus$eq$(Growable.scala:53)
      at
```

```
scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:105)
      at
scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:49)
      at scala.collection.TraversableOnce.to(TraversableOnce.scala:366)
      at scala.collection.TraversableOnce.to$(TraversableOnce.scala:364)
      at
org.apache.spark.InterruptibleIterator.to(InterruptibleIterator.scala:28)
      at scala.collection.TraversableOnce.toBuffer(TraversableOnce.scala:358)
      at
scala.collection.TraversableOnce.toBuffer$(TraversableOnce.scala:358)
      at
org.apache.spark.InterruptibleIterator.toBuffer(InterruptibleIterator.scala:2
8)
      at scala.collection.TraversableOnce.toArray(TraversableOnce.scala:345)
      at scala.collection.TraversableOnce.toArray$(TraversableOnce.scala:339)
      at
org.apache.spark.InterruptibleIterator.toArray(InterruptibleIterator.scala:28
)
      at org.apache.spark.rdd.RDD.$anonfun$collect$2(RDD.scala:1030)
      at
org.apache.spark.SparkContext.$anonfun$runJob$5(SparkContext.scala:2254)
      at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
      at org.apache.spark.scheduler.Task.run(Task.scala:131)
      at
org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:5
06)
      at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1462)
      at
org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:509)
      at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:114
9)
      at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:62
4)
      at java.lang.Thread.run(Thread.java:750)

Driver stacktrace:
      at
org.apache.spark.scheduler.DAGScheduler.failJobAndIndependentStages(DAGSchedu
ler.scala:2403)
      at
org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2(DAGScheduler.sc
ala:2352)
      at
org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2$adapted(DAGSche
duler.scala:2351)
      at
scala.collection.mutable.ResizableArray.foreach(ResizableArray.scala:62)
      at
scala.collection.mutable.ResizableArray.foreach$(ResizableArray.scala:55)
      at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffer.scala:49)
      at
org.apache.spark.scheduler.DAGScheduler.abortStage(DAGScheduler.scala:2351)
      at
```

```
org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1(DAGSch
eduler.scala:1109)
      at
org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1$adapte
d(DAGScheduler.scala:1109)
      at scala.Option.foreach(Option.scala:407)
      at
org.apache.spark.scheduler.DAGScheduler.handleTaskSetFailed(DAGScheduler.scal
a:1109)
      at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.doOnReceive(DAGSchedu
ler.scala:2591)
      at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGSchedule
r.scala:2533)
      at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGSchedule
r.scala:2522)
      at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:49)
      at
org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:898)
      at org.apache.spark.SparkContext.runJob(SparkContext.scala:2214)
      at org.apache.spark.SparkContext.runJob(SparkContext.scala:2235)
      at org.apache.spark.SparkContext.runJob(SparkContext.scala:2254)
      at org.apache.spark.SparkContext.runJob(SparkContext.scala:2279)
      at org.apache.spark.rdd.RDD.$anonfun$collect$1(RDD.scala:1030)
      at
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151
)
      at
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112
)
      at org.apache.spark.rdd.RDD.withScope(RDD.scala:414)
      at org.apache.spark.rdd.RDD.collect(RDD.scala:1029)
      at
org.apache.spark.api.python.PythonRDD$.collectAndServe(PythonRDD.scala:180)
      at
org.apache.spark.api.python.PythonRDD.collectAndServe(PythonRDD.scala)
      at sun.reflect.GeneratedMethodAccessor58.invoke(Unknown Source)
      at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
java:43)
      at java.lang.reflect.Method.invoke(Method.java:498)
      at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
      at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
      at py4j.Gateway.invoke(Gateway.java:282)
      at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
      at py4j.commands.CallCommand.execute(CallCommand.java:79)
      at
py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:182)
      at py4j.ClientServerConnection.run(ClientServerConnection.java:106)
      at java.lang.Thread.run(Thread.java:750)
Caused by: org.apache.spark.api.python.PythonException: Traceback (most
recent call last):
  File
```

```
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/worker.py"
, line 619, in main
    process()
  File
"/content/spark-3.2.0-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/worker.py"
, line 609, in process
    out_iter = func(split_index, iterator)
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 2918,
in pipeline_func
    return func(split, prev_func(split, iterator))
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 417,
in func
    return f(iterator)
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 1237,
in <lambda>
    return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
  File "/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/rdd.py", line 1237,
in <genexpr>
    return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
TypeError: 'NoneType' object is not iterable

      at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonExcep
tion(PythonRunner.scala:545)
      at
org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:703)
      at
org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:685)
      at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.hasNext(PythonRun
ner.scala:498)
      at
org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37
)
      at scala.collection.Iterator.foreach(Iterator.scala:943)
      at scala.collection.Iterator.foreach$(Iterator.scala:943)
      at
org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator.scala:28
)
      at scala.collection.generic.Growable.$plus$plus$eq(Growable.scala:62)
      at scala.collection.generic.Growable.$plus$plus$eq$(Growable.scala:53)
      at
scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:105)
      at
scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:49)
      at scala.collection.TraversableOnce.to(TraversableOnce.scala:366)
      at scala.collection.TraversableOnce.to$(TraversableOnce.scala:364)
      at
org.apache.spark.InterruptibleIterator.to(InterruptibleIterator.scala:28)
```

```
	at scala.collection.TraversableOnce.toBuffer(TraversableOnce.scala:358)
	at
scala.collection.TraversableOnce.toBuffer$(TraversableOnce.scala:358)
	at
org.apache.spark.InterruptibleIterator.toBuffer(InterruptibleIterator.scala:2
8)
	at scala.collection.TraversableOnce.toArray(TraversableOnce.scala:345)
	at scala.collection.TraversableOnce.toArray$(TraversableOnce.scala:339)
	at
org.apache.spark.InterruptibleIterator.toArray(InterruptibleIterator.scala:28
)
	at org.apache.spark.rdd.RDD.$anonfun$collect$2(RDD.scala:1030)
	at
org.apache.spark.SparkContext.$anonfun$runJob$5(SparkContext.scala:2254)
	at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
	at org.apache.spark.scheduler.Task.run(Task.scala:131)
	at
org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:5
06)
	at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1462)
	at
org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:509)
	at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:114
9)
	at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:62
4)
	... 1 more
```

Теперь у нас новая ошибка:

```
return self.mapPartitions(lambda i: [sum(1 for _ in i)]).sum()
TypeError: 'NoneType' object is not iterable
```

Она возникает, когда мы пытаемся превратить пустую запись в Row. Для решения этой проблемы мы создадим новый метод toAirportOptSafe.

```
def toAirportOptSafe(data):
    lst = data.split(",")
    if (len(lst) == 13):
        r = Airport(*lst)
    else:
        r = Airport('','','',0,'','','','','','','','')
    return r
```

Применим данную функцию к нашему датасету:

```
airportSafeRdd = noHeader.flatMap(toAirportOptSafe)
rdd_showF(airportSafeRdd)
```

```
00A
heliport
Total Rf Heliport
11
NA
```

```
US
US-PA
Bensalem
00A
```

Проверим ее применимость:

```
airportSafeRdd.count()
```

```
746473
```

```
airportFinal=noHeader.map(toAirportOptSafe)
rdd_showF(airportFinal)
```

```
Row(ident='00A', type='heliport', name='Total Rf Heliport', elevationFt='11',
continent='NA', isoCountry='US', isoRegion='US-PA', municipality='Bensalem',
gpsCode='00A', iataCode='', localCode='00A', longitude='-74.93360137939453',
latitude=' 40.07080078125')
Row(ident='00AA', type='small_airport', name='Aero B Ranch Airport',
elevationFt='3435', continent='NA', isoCountry='US', isoRegion='US-KS',
municipality='Leoti', gpsCode='00AA', iataCode='', localCode='00AA',
longitude='-101.473911', latitude=' 38.704022')
Row(ident='00AK', type='small_airport', name='Lowell Field',
elevationFt='450', continent='NA', isoCountry='US', isoRegion='US-AK',
municipality='Anchor Point', gpsCode='00AK', iataCode='', localCode='00AK',
longitude='-151.695999146', latitude=' 59.94919968')
Row(ident='00AL', type='small_airport', name='Epps Airpark',
elevationFt='820', continent='NA', isoCountry='US', isoRegion='US-AL',
municipality='Harvest', gpsCode='00AL', iataCode='', localCode='00AL',
longitude='-86.77030181884766', latitude=' 34.86479949951172')
Row(ident='00AR', type='closed', name='Newport Hospital & Clinic Heliport',
elevationFt='237', continent='NA', isoCountry='US', isoRegion='US-AR',
municipality='Newport', gpsCode='', iataCode='', localCode='',
longitude='-91.254898', latitude=' 35.6087')
Row(ident='00AS', type='small_airport', name='Fulton Airport',
elevationFt='1100', continent='NA', isoCountry='US', isoRegion='US-OK',
municipality='Alex', gpsCode='00AS', iataCode='', localCode='00AS',
longitude='-97.8180194', latitude=' 34.9428028')
Row(ident='00AZ', type='small_airport', name='Cordes Airport',
elevationFt='3810', continent='NA', isoCountry='US', isoRegion='US-AZ',
municipality='Cordes', gpsCode='00AZ', iataCode='', localCode='00AZ',
longitude='-112.16500091552734', latitude=' 34.305599212646484')
Row(ident='00CA', type='small_airport', name='Goldstone /Gts/ Airport',
elevationFt='3038', continent='NA', isoCountry='US', isoRegion='US-CA',
municipality='Barstow', gpsCode='00CA', iataCode='', localCode='00CA',
longitude='-116.888000488', latitude=' 35.350498199499995')
Row(ident='00CL', type='small_airport', name='Williams Ag Airport',
elevationFt='87', continent='NA', isoCountry='US', isoRegion='US-CA',
municipality='Biggs', gpsCode='00CL', iataCode='', localCode='00CL',
longitude='-121.763427', latitude=' 39.427188')
Row(ident='00CN', type='heliport', name='Kitchen Creek Helibase Heliport',
elevationFt='3350', continent='NA', isoCountry='US', isoRegion='US-CA',
municipality='Pine Valley', gpsCode='00CN', iataCode='', localCode='00CN',
longitude='-116.4597417', latitude=' 32.7273736')
```

Получим коллекцию, содержащую максимальную высота аэропорта с разбивкой по странам. Для этого первым шагом получим PairRDD: RDD[(K,V)], где K - это страна, а V - высота

```
pairAirport = airportFinal.map(lambda Airport : (Airport['isoCountry'],
int(Airport['elevationFt']) if Airport['elevationFt'] != '' else 0))
pairAirport.first()
```

```
('US', 11)
```

Теперь нам необходимо применить функцию reduceByKey и получить нужный результат:

```
result = pairAirport.reduceByKey(lambda x, y : y if (y >= x) else
x).collect()
for i in sorted(result, key=lambda t: t[1], reverse=True):
    print(i)
```

```
('US', 29977)
('IN', 22000)
('PE', 14965)
('CN', 14472)
('BO', 14360)
('CO', 13119)
('AR', 13000)
('CL', 12468)
('NP', 12400)
('TJ', 11962)
('FR', 11647)
('IT', 11443)
('CH', 10837)
('AF', 10490)
('LS', 10400)
('KE', 10200)
('MX', 10074)
('EC', 9649)
('AQ', 9300)
('ID', 9288)
('BT', 9000)
('ET', 8490)
('PG', 8400)
('KG', 8250)
('GT', 7933)
('TZ', 7795)
('MW', 7759)
('ER', 7661)
('AT', 7522)
('IR', 7385)
('PK', 7316)
('MN', 7260)
('YE', 7216)
('SA', 6858)
('BR', 6825)
('RU', 6695)
('CD', 6562)
('OM', 6500)
```

('ZA', 6464)
('TR', 6400)
('UG', 6200)
('RW', 6102)
('NA', 6063)
('KZ', 6051)
('MM', 6000)
('AO', 5778)
('AU', 5752)
('BI', 5741)
('SO', 5720)
('HN', 5475)
('MA', 5459)
('ZM', 5454)
('ZW', 5370)
('CA', 5350)
('VE', 5269)
('AM', 5000)
('PA', 5000)
('MG', 4997)
('VN', 4937)
('KR', 4816)
('GE', 4778)
('CR', 4650)
('CM', 4593)
('KP', 4547)
('DZ', 4518)
('MZ', 4505)
('EG', 4368)
('ES', 4265)
('ME', 4252)
('PH', 4251)
('BW', 4250)
('NG', 4232)
('DO', 3950)
('LA', 3721)
('LK', 3580)
('TD', 3524)
('SZ', 3522)
('CF', 3464)
('IQ', 3455)
('AD', 3450)
('BG', 3447)
('GN', 3396)
('MY', 3350)
('PT', 3278)
('NI', 3232)
('SD', 3230)
('DE', 3215)
('GY', 3198)
('NO', 3150)
('NZ', 3100)
('LB', 3018)
('RS', 2966)
('JM', 2940)

('AZ', 2863)
('JO', 2790)
('CG', 2756)
('SY', 2727)
('IS', 2625)
('CZ', 2618)
('UZ', 2617)
('IL', 2556)
('SS', 2500)
('PS', 2485)
('HR', 2462)
('GH', 2408)
('SK', 2356)
('BA', 2349)
('GA', 2346)
('PR', 2340)
('DJ', 2313)
('MK', 2313)
('LY', 2296)
('JP', 2271)
('SV', 2229)
('SR', 2217)
('XK', 2168)
('GR', 2167)
('GQ', 2165)
('NE', 2162)
('BE', 2067)
('PL', 2060)
('PY', 1873)
('MR', 1775)
('TL', 1771)
('RO', 1740)
('BF', 1706)
('GL', 1664)
('SI', 1654)
('SE', 1640)
('LR', 1632)
('EH', 1624)
('ML', 1621)
('LI', 1585)
('FI', 1584)
('CI', 1583)
('LU', 1522)
('TG', 1515)
('BJ', 1512)
('BZ', 1500)
('PF', 1481)
('GB', 1386)
('TM', 1329)
('TH', 1312)
('SL', 1300)
('UA', 1150)
('HU', 1150)
('AL', 1120)
('TN', 1060)

('SH', 1017)
('CX', 916)
('NC', 902)
('AE', 869)
('TW', 790)
('SM', 787)
('MD', 758)
('BY', 748)
('UY', 743)
('IE', 680)
('CV', 669)
('GF', 656)
('CU', 656)
('LT', 650)
('VU', 630)
('SC', 610)
('MP', 607)
('ST', 591)
('SN', 584)
('MS', 550)
('LV', 518)
('KW', 472)
('CY', 404)
('NL', 375)
('NF', 371)
('FJ', 358)
('KH', 350)
('GG', 336)
('EE', 331)
('DK', 325)
('TO', 325)
('GU', 311)
('FO', 305)
('MT', 300)
('JE', 277)
('FK', 244)
('VA', 221)
('NU', 209)
('HT', 203)
('MU', 186)
('AS', 185)
('PW', 176)
('BD', 176)
('KN', 170)
('BB', 169)
('GW', 165)
('BH', 136)
('VC', 136)
('WS', 131)
('QA', 130)
('SB', 130)
('BQ', 129)
('AI', 127)
('VI', 125)
('HK', 107)

```
('IM', 97)
('GM', 95)
('KM', 93)
('FM', 91)
('SG', 86)
('WF', 79)
('TT', 75)
('DM', 73)
('BN', 73)
('RE', 66)
('AG', 62)
('AW', 60)
('GP', 59)
('TV', 55)
('BL', 49)
('CK', 45)
('GD', 41)
('BS', 37)
('MH', 32)
('KI', 30)
('MV', 29)
('CW', 29)
('PM', 27)
('YT', 23)
('NR', 22)
('LC', 22)
('MO', 20)
('TF', 20)
('MC', 20)
('MQ', 16)
('TC', 15)
('VG', 15)
('GI', 15)
('UM', 14)
('SX', 13)
('BM', 12)
('CC', 10)
('IO', 9)
('KY', 8)
('MF', 7)
('', 0)
```

Выводы

- RDD API - это низкоуровневый API, который позволяет применять любые функции к распределенным данным
- При использовании RDD API обработка всех исключительных ситуаций лежит на плечах разработчика

После завершения работы не забывайте останавливать SparkSession, чтобы освободить ресурсы кластера!

```
spark.stop
```

```
<bound method SparkSession.stop of <pyspark.sql.session.SparkSession object
at 0x7f1710fc6e90>>
```

4. Основы интерфейса кадров данных Apache Spark

4.1. Сравнение RDD API и DataFrame API

Типы данных

RDD: низкоуревная распределенная коллекция данных любого типа

DF: таблица со схемой, состоящей из колонок разных типов, описанных в org.apache.spark.sql.types

Обработка данных

RDD: сериализуемые функции

DF: кодогенерация SQL > Java код

Функции и алгоритмы

RDD: нет ограничений

DF: ограничен SQL операторами, функциями org.apache.spark.sql.functions и пользовательскими функциями

Источники данных

RDD: каждый источник имеет свое API

DF: единое API для всех источников

Производительность

RDD: напрямую зависит от качества кода

DF: встроенные механизмы оптимизации SQL запроса

Потоковая обработка данных

RDD: устаревший DStreams

DF: активно развивающийся Structured Streaming

Выводы:

- На текущий момент RDD является низкоуровневым API, которое постепенно уходит "под капот" Apache Spark
- DF API представляет собой библиотеку для обработки данных с использованием SQL примитивов

4.2. Базовые функции

Что такое DataFrame?

Библиотека pyspark.sql предоставляет альтернативный API для управления структурированными наборами данных, известный как «кадры данных». (Кадры данных "Датафреймы" не являются специфической концепцией Spark, но pyspark предоставляет свою собственную выделенную библиотеку кадров данных). Они отличаются от RDD, но при необходимости вы можете преобразовать RDD в кадр данных или наоборот.

См. Руководство по Spark SQL и DataFrame для получения дополнительной информации.

Как создать DataFrame? Вы можете загрузить кадр данных непосредственно из источника входных данных.

- Вы также можете создать кадр данных из файла CSV или строки, как показано ниже
- Чтение и запись данных из файлов hdfs
- Чтение и запись данных из таблиц Hive
- Чтение и запись данных из Баз данных

```
dictionary = [{'city': 'Yekat'}, {'city': 'Moscow'}, {'city': 'Paris'},
{'city': 'Madrid'}, {'city': 'London'}, {'city': 'New York'}]
cityList = spark.sparkContext.parallelize(dictionary)

df = cityList.toDF()
```

У любого DF есть схема:

```
df.printSchema()
```

```
root
 |-- city: string (nullable = true)
```

Посмотреть содержимое DF можно с помощью метода show():

```
df.show(4)
```

```
+------+
|  city|
+------+
| Yekat|
|Moscow|
| Paris|
|Madrid|
+------+
only showing top 4 rows
```

Также можно вывести содержимое в вертикальной ориентации - это удобно при большом количестве столбцов:

```
df.show(n = 20, truncate = 200, vertical=True)
```

```
-RECORD 0--------
 city | Yekat
-RECORD 1--------
```

```
 city | Moscow
-RECORD 2--------
 city | Paris
-RECORD 3--------
 city | Madrid
-RECORD 4--------
 city | London
-RECORD 5--------
 city | New York
```

Подсчет количества элементов в DF с помощью count():

```
df.count()
```

```
6
```

Отфильтровать данные можно с помощью метода filter. В отличие от RDD, он принимает SQL выражение:

```
import pyspark.sql.functions as f

df.filter(f.col('city') == "Moscow").show()
```

```
+------+
|  city|
+------+
|Moscow|
+------+
```

```
df.filter('city = "Moscow"').show()
```

```
+------+
|  city|
+------+
|Moscow|
+------+
```

```
from pyspark.sql.functions import col
df.filter(col('city') == "Moscow").show()
```

```
+------+
|  city|
+------+
|Moscow|
+------+
```

Добавить новую колонку можно с помощью метода withColumn. Необходимо помнить, что данный метод, как и другие, является трансформацией и не изменяет оригинальный DF, а создает новый.

```
df.withColumn("upperCity", f.upper(col("city"))).show()
```

```
+--------+---------+
|    city|upperCity|
```

```
+--------+---------+
|   Yekat|    YEKAT|
|  Moscow|   MOSCOW|
|   Paris|    PARIS|
|  Madrid|   MADRID|
|  London|   LONDON|
|New York| NEW YORK|
+--------+---------+
```

Аналогичный результат получить, используя метод select. Данный метод может быть использован не только для выбора определенных колонок, но и для создания новых.

```
withUpper = df.select(col("city"), f.upper(col("city")).alias("upperCity"))
withUpper.show()
```

```
+--------+---------+
|    city|upperCity|
+--------+---------+
|   Yekat|    YEKAT|
|  Moscow|   MOSCOW|
|   Paris|    PARIS|
|  Madrid|   MADRID|
|  London|   LONDON|
|New York| NEW YORK|
+--------+---------+
```

Если передать col("*") в select, то вы получите DF со всеми колонками. Это полезно, когда вы не знаете список всех колонок (например вы получили его через API), но вам нужно их все выбрать и добавить новую колонку. Это можно сделать следующим образом:

```
# методы name, as и alias часто являются взаимозаменяемыми

withUpper.select(
    col("*"),
    f.lower(col("city")).name("lowerCity"),
    (f.length(col("city")) + 1).alias("length"),
    f.lit("foo").alias("bar")).show()
```

```
+--------+---------+---------+------+---+
|    city|upperCity|lowerCity|length|bar|
+--------+---------+---------+------+---+
|   Yekat|    YEKAT|    yekat|     6|foo|
|  Moscow|   MOSCOW|   moscow|     7|foo|
|   Paris|    PARIS|    paris|     6|foo|
|  Madrid|   MADRID|   madrid|     7|foo|
|  London|   LONDON|   london|     7|foo|
|New York| NEW YORK| new york|     9|foo|
+--------+---------+---------+------+---+
```

При необходимости в select можно передать список колонок, используя обычные строки:

```
withUpper.select("city", "upperCity").show()
```

```
+--------+---------+
|    city|upperCity|
+--------+---------+
|   Yekat|    YEKAT|
|  Moscow|   MOSCOW|
|   Paris|    PARIS|
|  Madrid|   MADRID|
|  London|   LONDON|
|New York| NEW YORK|
+--------+---------+
```

Удалить колонку из DF можно с помощью метода drop:

```
# drop не будет выдавать ошибку, если будет указана несуществующая колонка

withUpper.drop("upperCity", "abraKadabra").show()
```

```
+--------+
|    city|
+--------+
|   Yekat|
|  Moscow|
|   Paris|
|  Madrid|
|  London|
|New York|
+--------+
```

Выводы:

- методы filter и select принимают в качестве аргументов колонки org.apache.spark.sql.Column. Это может быть либо ссылка на существующую колонку, либо функцию из org.apache.spark.sql.functions
- любые трансформации возвращают новый DF, не меняя существующий
- тип org.apache.spark.sql.Column играет важную роль в DF API - на его основе создаются ссылки на существующие колонки, а также функции, принимающие org.apache.spark.sql.Column и возвращающие org.apache.spark.sql.Column. По этой причине обычное сравнение == не будет работать в DF API, тк filter принимает org.apache.spark.sql.Column, а не Boolean
- Класс DataFrame в последних версиях Spark представляет собой org.apache.spark.sql.Dataset[org.apache.spark.sql.Row], поэтому его описание следует искать в org.apache.spark.sql.Dataset

4.3. Очистка данных

Одной из задач обработки данных является их очистка. DF API содержит класс функций "not available", описанный в пакете org.apache.spark.sql.DataFrameNaFunctions. В данном пакете есть три функции:

- na.drop

- na.fill
- na.replace

Для демонстрации работы данных функций создадим новый датасет:

```
from pyspark.sql.functions import lit
from pyspark.sql.functions import split
from pyspark.sql.functions import col
from pyspark.sql.functions import explode

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","""

# Создаем DF из одной строки и добавляем данные в виде новой колонки
raw = spark.range(0,1).select(lit(testData).alias("value"))
raw.show(1,150, True)

-RECORD
0-------------------------------------------------------------------------
-------------------------------------------------------------------------
 value | { "name":"Moscow", "country":"Rossiya", "continent": "Europe",
"population": 12380664}\n{ "name":"Madrid", "country":"Spain" }\n{
"name":"Paris", "...


# Создаем новую колонку, разибая наши данные по \n
jsonStrings = split(col("value"), "\n").alias("value")
jsonStrings

Column<'split(value,
, -1) AS value'>

# Используем функцию explode для того, чтобы превратить кадр данных в
коллекцию строк
# для превращения DataFrame в Dataset[String]
splited = raw.select(explode(jsonStrings).name("val"))
splited.show(n = 10, truncate = False)

+-------------------------------------------------------------------------
----------+
|val
|
+-------------------------------------------------------------------------
----------+
|{ "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",
|
+--------------------------------------------------------------------------
----------+
```

```
# Создаем новый датафре... датасет, в котором наши JSON строки будут
распарсены
df = spark.read.json(splited.rdd.map(lambda r: r.val))
df.printSchema()
df.show()
```

```
root
 |-- _corrupt_record: string (nullable = true)
 |-- continent: string (nullable = true)
 |-- country: string (nullable = true)
 |-- name: string (nullable = true)
 |-- population: long (nullable = true)
```

```
+--------------------+---------+-------+---------+----------+
|     _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 Yor...|     null|   null|     null|      null|
+--------------------+---------+-------+---------+----------+
```

Для очистки датасета:

- удалим строку с навалидным JSON, сохраним ее в отдельное место
- удалим дубликаты
- заполним nullы в колонках
- исправим Rossiya на Russia

```
corruptData = df.select(col("_corrupt_record")).na.drop("all").collect()
print(corruptData)
```

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

```
fillData = {'continent': 'Undefined', 'population': '0'} # Map("continent" ->
"Undefined", "population" -> 0)
replaceData = {'Rossiya': 'Russia'} # Map("Rossiya" -> "Russia")

cleanData = df \
    .drop(col('_corrupt_record')) \
    .na.drop('all') \
    .na.fill(fillData) \
    .na.replace(replaceData, 'country') \
    .dropDuplicates()


cleanData.show()
```

```
/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/sql/dataframe.py:2233:
UserWarning: to_replace is a dict and value is not None. value will be
ignored.
  warnings.warn("to_replace is a dict and value is not None. value will be
ignored.")

+---------+-------+---------+----------+
|continent|country|     name|population|
+---------+-------+---------+----------+
|   Europe| France|    Paris|   2196936|
|   Europe|Germany|   Berlin|   3490105|
|Undefined|  Spain|   Madrid|         0|
|   Africa|  Egypt|    Cairo|  11922948|
|   Europe|  Spain|Barselona|         0|
|   Europe| Russia|   Moscow|  12380664|
+---------+-------+---------+----------+
```

Выводы:

- DF API обладает удобным API для очистки данных, позволяющим разработчику сконцентрироваться разработчику на бизнес логике, а не на написании функций для обработки всех возможных исключительных ситуаций
- метод spark.read.json позволяет читать не только файлы, но и Dataset[String], содержащие JSON строки.

4.4. Агрегаты

Посчитаем суммарное население и количество городов с разбивкой по континентам:

```
aggCount = cleanData.groupBy(col('continent')).count()
aggCount.show()
```

```
+---------+-----+
|continent|count|
+---------+-----+
|   Europe|    4|
|   Africa|    1|
|Undefined|    1|
```

```
+---------+-----+
```

```
aggSum = cleanData.groupBy(col('continent')).sum('population')
aggSum.show()
```

```
+---------+---------------+
|continent|sum(population)|
+---------+---------------+
|   Europe|       18067705|
|   Africa|       11922948|
|Undefined|              0|
+---------+---------------+
```

Для того, чтобы совместить несколько агрегатов в одном DF, мы можем использовать метод agg(). Данный метод позволяет использовать любые Aggregate functions из пакета org.apache.spark.sql.functions

```
from pyspark.sql.functions import count
from pyspark.sql.functions import sum
agg = cleanData.groupBy(col('continent')).agg(count(col("*")).alias('cnt'),
sum(col('population')).alias('sumPop'))
agg.show()
```

```
+---------+---+--------+
|continent|cnt|  sumPop|
+---------+---+--------+
|   Europe|  4|18067705|
|   Africa|  1|11922948|
|Undefined|  1|       0|
+---------+---+--------+
```

С помощью агрегатов мы можем выполнять такие действия, как, например, collect_list и collect_set. Стоит отметить, что колонки в Spark могут иметь не только скалярные типы, но и структуры, словари и массивы:

```
from pyspark.sql.functions import collect_list
aggList =
cleanData.groupBy(col('continent')).agg(collect_list("country").alias("countr
ies"))
aggList.printSchema()
aggList.show(n = 10, truncate = 100, vertical = True)
```

```
root
 |-- continent: string (nullable = false)
 |-- countries: array (nullable = false)
 |    |-- element: string (containsNull = false)
```

```
-RECORD 0-------------------------------------
 continent | Europe
 countries | [France, Germany, Spain, Russia]
-RECORD 1-------------------------------------
 continent | Africa
 countries | [Egypt]
-RECORD 2-------------------------------------
```

```
 continent | Undefined
 countries | [Spain]
```

Используя методы struct и to_json, мы можем превратить произвольный набор колонок в JSON строку. Этот методы часто используется перед отправкой данных в Kafka

```
from pyspark.sql.functions import struct
withStruct = aggList.select(struct(col('continent'),
col('countries')).alias("s"))
withStruct.printSchema()

withStruct.show(10, False)
```

```
root
 |-- s: struct (nullable = false)
 |    |-- continent: string (nullable = false)
 |    |-- countries: array (nullable = false)
 |    |    |-- element: string (containsNull = false)

+----------------------------------------+
|s                                       |
+----------------------------------------+
|{Europe, [France, Germany, Spain, Russia]}|
|{Africa, [Egypt]}                       |
|{Undefined, [Spain]}                    |
+----------------------------------------+
```

```
from pyspark.sql.functions import to_json
withStruct.withColumn("s", to_json(col('s'))).show(10, False)
```

```
+------------------------------------------------------------------------+
|s                                                                       |
+------------------------------------------------------------------------+
|{"continent":"Europe","countries":["France","Germany","Spain","Russia"]}|
|{"continent":"Africa","countries":["Egypt"]}                            |
|{"continent":"Undefined","countries":["Spain"]}                         |
+------------------------------------------------------------------------+
```

Если необходимо превратить все колонки DF в JSON String, можно воспользоваться функций toJSON:

```
jString = aggList.toJSON()
#Поскольку мы получили RDD show больше не сработает
for i in jString.take(5):
    print(i)
```

```
{"continent":"Europe","countries":["France","Germany","Spain","Russia"]}
{"continent":"Africa","countries":["Egypt"]}
{"continent":"Undefined","countries":["Spain"]}
```

Если нам необходимо создать колонки из значений текущих колонок, мы можем воспользоваться функцией pivot

```
cleanData.groupBy(col("country")).pivot("continent").agg(sum("population")).s
how()
```

```
+-------+--------+--------+---------+
|country|  Africa|  Europe|Undefined|
+-------+--------+--------+---------+
| Russia|    null|12380664|     null|
|Germany|    null| 3490105|     null|
| France|    null| 2196936|     null|
|  Spain|    null|       0|        0|
|  Egypt|11922948|    null|     null|
+-------+--------+--------+---------+
```

Выводы:

- DF API позволяет строить большое количество агрегатов. При этом необходимо помнить, что операции groupBy, cube, rollup возвращают org.apache.spark.sql.RelationalGroupedDataset, к которому затем необходимо применить одну из функций агрегации - count, sum, agg и т.п.
- При вычислении агрегатов необходимо помнить, что эта операция требует перемешивания данных между воркерами, что, в случае перекошенных данных, может привести к OOM на **воркере**.

```
spark.stop
```

```
<bound method SparkSession.stop of <pyspark.sql.session.SparkSession object
at 0x7f1d8568d690>>
```

5. Особенности работы с интерфейсом кадров данных Apache Spark

5.1. Кеширование

По умолчанию при применении каждого действия Spark пересчитывает весь граф, что может негативно сказаться на производительности приложения. Для демонстрации возьмем датасет Airport Codes

```
airports = spark.read.options(header='True',
inferSchema='True').csv("sample_data/airport-codes.csv")
airports.printSchema()
airports.show(n = 1, truncate = 100, vertical = True)
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)

-RECORD 0-----------------------------------------
 ident        | 00A
 type         | heliport
 name         | Total Rf Heliport
 elevation_ft | 11
 continent    | NA
 iso_country  | US
 iso_region   | US-PA
 municipality | Bensalem
 gps_code     | 00A
 iata_code    | null
 local_code   | 00A
 coordinates  | -74.93360137939453, 40.07080078125
only showing top 1 row
```

Посчитаем несколько агрегатов. Несмотря на то, что onlyRuAndHigh является общим для всех действий, он пересчитывается при вызове каждого действия.

```
from pyspark.sql.functions import col, lit
onlyRuAndHigh = airports.filter('iso_country == "RU" and elevation_ft >
1000')
onlyRuAndHigh.show(n = 1, truncate = 100, vertical = True)

onlyRuAndHigh.count()
onlyRuAndHigh.collect()
onlyRuAndHigh.groupBy(col('municipality')) \
    .count() \
    .orderBy(col("count").desc()) \
    .na.drop("any") \
    .show()
```

```
-RECORD 0-----------------------------
 ident        | RU-0006
 type         | closed
 name         | Arabatuk Air Base
 elevation_ft | 2280
 continent    | EU
 iso_country  | RU
 iso_region   | RU-CHI
 municipality | Daurija
 gps_code     | null
 iata_code    | null
 local_code   | ZA2N
 coordinates  | 117.098999, 50.223801
only showing top 1 row

+---------------+-----+
|   municipality|count|
+---------------+-----+
|          Chita|    3|
```

```
|         Borzya|    2|
|       Ulan Ude|    2|
|  Nizhneangarsk|    2|
|   Nizhneudinsk|    2|
|        Irkutsk|    2|
|        Mirnaya|    1|
|   Karachayevsk|    1|
|       Barguzin|    1|
|         Amazar|    1|
|          Baley|    1|
|          Kyren|    1|
|    Olovyannaya|    1|
|   Snezhnogorsk|    1|
|       Karakhun|    1|
|      Priboynyy|    1|
|       Tlyarata|    1|
|        Chistyy|    1|
|Sherlovaya Gora|    1|
|     Kislovodsk|    1|
+---------------+-----+
only showing top 20 rows
```

Для решения этой проблемы следует использовать методы cache, либо persist. Данные методы сохраняют состояние графа после первого действия, и следующие обращаются к нему. Разница между методами заключается в том, что persist позволяет выбрать, куда сохранить данные, а cache использует значение по умолчанию. В текущей версии Spark это StorageLevel.MEMORY_ONLY. Важно помнить, что данный кеш не предназначен для обмена данными между разными Spark приложения - он является внутренним для приложения. После того, как работа с данными окончена, необходимо выполнить unpersist для очистки памяти.

```
onlyRuAndHigh.cache()
onlyRuAndHigh.count()
# при вычислении count данные будут помещены в cache
onlyRuAndHigh.show(n = 1, truncate = 100, vertical = True)
onlyRuAndHigh.collect()
onlyRuAndHigh.groupBy(col('municipality')).count().orderBy(col("count").desc(
)).na.drop("any").show()

onlyRuAndHigh.unpersist()
```

```
-RECORD 0-----------------------------
 ident        | RU-0006
 type         | closed
 name         | Arabatuk Air Base
 elevation_ft | 2280
 continent    | EU
 iso_country  | RU
 iso_region   | RU-CHI
 municipality | Daurija
 gps_code     | null
 iata_code    | null
 local_code   | ZA2N
 coordinates  | 117.098999, 50.223801
```

```
only showing top 1 row

+---------------+-----+
|   municipality|count|
+---------------+-----+
|          Chita|    3|
|         Borzya|    2|
|       Ulan Ude|    2|
|  Nizhneangarsk|    2|
|   Nizhneudinsk|    2|
|        Irkutsk|    2|
|        Mirnaya|    1|
|   Karachayevsk|    1|
|       Barguzin|    1|
|         Amazar|    1|
|          Baley|    1|
|          Kyren|    1|
|    Olovyannaya|    1|
|   Snezhnogorsk|    1|
|       Karakhun|    1|
|      Priboynyy|    1|
|       Tlyarata|    1|
|        Chistyy|    1|
|Sherlovaya Gora|    1|
|     Kislovodsk|    1|
+---------------+-----+
only showing top 20 rows


DataFrame[ident: string, type: string, name: string, elevation_ft: int,
continent: string, iso_country: string, iso_region: string, municipality:
string, gps_code: string, iata_code: string, local_code: string, coordinates:
string]
```

Выводы:

- Использование cache и persist позволяет существенно сократить время обработки данных, однако следует помнить и об увеличении потребляемой памяти на воркерах

5.2. Репартиционирование

RDD и DF являются представляют собой классы, описывающие распределенные коллекции данных. Они (коллекции) разбиты на крупные блоки, которые называются партициями. В графе вычисления, который называется в Spark DAG (Direct Acyclic Graph), есть три основных компонента - job, stage, task.

- job представляет собой весь граф целиком, от момента создания DF, до применения action к нему. Состоит из одной или более stage. Когда возникает необходимость сделать shuffle данных, Spark создает новый stage.
- Каждый stage состоит из большого количества task.
- task это базовая операция над данными. Одновременно Spark выполняет N task, которые обрабатывают N партиций, где N - это суммарное число доступных потоков на всех воркерах.

Исходя из этого, важно обеспечивать:

- достаточное количество партиций для распределения нагрузки по всем воркерам
- равномерное распределение данных между партициями

Создадим датасет с перекосом данных:

```
from pyspark.sql.functions import spark_partition_id, asc, desc, when

skewColumn = when(col("id") < 900, lit(0)).otherwise(lit(1))

skewDf = spark.range(0,1000).repartition(10, skewColumn)

def printItemPerPartition(ds):
    ds \
        .withColumn("partitionId", spark_partition_id()) \
        .groupBy("partitionId") \
        .count() \
        .orderBy(asc("count")) \
        .show()

printItemPerPartition(skewDf)
```

```
+-----------+-----+
|partitionId|count|
+-----------+-----+
|          3|  100|
|          1|  900|
+-----------+-----+
```

Любые операции с таким датасетом будут работать медленно, т.к.

- если суммарное количество потоков на всех воркерах больше 10, то в один момент времени работать будут максимум 10, остальные будут простаивать
- из 10 партицийи только в 2 есть данные и это означает, что только 2 потока будут обрабатывать данные, при этом из-за перекоса данных между ними (900 vs 100) первый станет узким горлышком

Обычно перекошенные датасеты возникают после вычисления агрегатов, оконных функций и соединений, но также могут возникать и при чтении источников.

Для устранения проблемы перекоса данных, следует использовать метод repartition:

```
# здесь мы передаем только новое количество партиций и Spark выполнит RoundRobinPartitioning
repartitionedDf = skewDf.repartition(20)

printItemPerPartition(repartitionedDf)
```

```
+-----------+-----+
|partitionId|count|
+-----------+-----+
|          0|   50|
```

```
|           1|   50|
|           2|   50|
|           3|   50|
|           4|   50|
|           5|   50|
|           6|   50|
|           7|   50|
|           8|   50|
|           9|   50|
|          10|   50|
|          11|   50|
|          12|   50|
|          13|   50|
|          14|   50|
|          15|   50|
|          16|   50|
|          17|   50|
|          18|   50|
|          19|   50|
+------------+-----+
```

```
# здесь мы добавляем к числу партиций колонку, по которой необходимо сделать репартиционирование,
# поэтому Spark выполнит HashPartitioning
repartitionedDf = skewDf.repartition(20, col("id"))

printItemPerPartition(repartitionedDf)
```

```
+------------+-----+
|partitionId|count|
+------------+-----+
|           0|   37|
|          17|   39|
|          18|   40|
|          16|   44|
|           6|   45|
|          12|   45|
|          11|   46|
|          13|   46|
|           4|   47|
|           2|   48|
|          14|   49|
|          19|   52|
|           5|   54|
|           9|   55|
|           8|   55|
|          10|   56|
|           7|   58|
|           3|   59|
|           1|   61|
|          15|   64|
+------------+-----+
```

5.3. Соление

Часто при вычислении агрегатов приходится работать с перекошенными данными:

```
airports.printSchema()
airports.groupBy(col('type')).count().orderBy(col('count').desc()).show()
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)

+--------------+-----+
|          type|count|
+--------------+-----+
| small_airport|34808|
|      heliport|12028|
|medium_airport| 4537|
|        closed| 4378|
| seaplane_base| 1030|
| large_airport|  616|
|   balloonport|   24|
+--------------+-----+
```

Поскольку при вычислении агрегата происходит неявный HashPartitioning по ключу (ключам) агрегата, то при выполнении определенных условий происходит нехватка памяти на воркере, которую нельзя исправить, не изменив подход к построению агрегата.

Один из вариантов устранение - соление ключей:

```
from pyspark.sql.functions import rand, when, ceil

salted = airports.withColumn("salt", ceil(rand()*10))
salted.show(n = 1, truncate = 200, vertical = True)
```

```
-RECORD 0-----------------------------------------
 ident        | 00A
 type         | heliport
 name         | Total Rf Heliport
 elevation_ft | 11
 continent    | NA
 iso_country  | US
 iso_region   | US-PA
 municipality | Bensalem
 gps_code     | 00A
```

```
 iata_code    | null
 local_code   | 00A
 coordinates  | -74.93360137939453, 40.07080078125
 salt         | 5
only showing top 1 row
```

Это позволяет нам существенно снизить объем данных в каждой партиции (30к vs 3к):

```
firstStep = salted.groupBy(col('type'), col('salt')).count()

firstStep.orderBy(col('count').desc()).show(20, False)
```

```
+-------------+----+-----+
|type         |salt|count|
+-------------+----+-----+
|small_airport|10  |3595 |
|small_airport|1   |3554 |
|small_airport|8   |3541 |
|small_airport|7   |3497 |
|small_airport|6   |3457 |
|small_airport|5   |3452 |
|small_airport|2   |3444 |
|small_airport|3   |3430 |
|small_airport|9   |3429 |
|small_airport|4   |3409 |
|heliport     |2   |1288 |
|heliport     |5   |1240 |
|heliport     |7   |1235 |
|heliport     |6   |1220 |
|heliport     |9   |1212 |
|heliport     |3   |1193 |
|heliport     |8   |1189 |
|heliport     |4   |1178 |
|heliport     |10  |1146 |
|heliport     |1   |1127 |
+-------------+----+-----+
only showing top 20 rows
```

Вторым шагом мы делаем еще один агрегат, суммируя предыдущие значения count:

```
from pyspark.sql.functions import sum, count
secondStep = firstStep.groupBy(col('type')).agg(sum("count").alias("count"))

secondStep.orderBy(col('count').desc()).show(200, False)
```

```
+--------------+-----+
|type          |count|
+--------------+-----+
|small_airport |34808|
|heliport      |12028|
|medium_airport|4537 |
|closed        |4378 |
|seaplane_base |1030 |
```

```
|large_airport |616  |
|balloonport   |24   |
+--------------+-----+
```

Несмотря на то, что мы сделали две группировки вместо одной, распределение данных по воркерам было более равномерным, что позволило избежать OOM на воркерах.

Выводы:

- Партиционирование - важный аспект распределенных вычислений, от которого напрямую зависит стабильность и скорость вычислений
- В Spark всегда работает правило 1 TASK = 1 THREAD = 1 PARTITION
- Репартиционирование и соление данных позволяет решить проблему перекоса данных и вычислений
- Важно помнить, что репартиционирование использует дисковую и сетевую подсистемы - обмен данными происходит **по сети**, а результат записывается на **диск**, что может стать узким местом при выполнении репартиционирования

5.4. Встроенные функции

Помимо базовых SQL операторов, в Spark существует большой набор встроенных функций:

- API методы из org.apache.spark.sql.functions
- SQL built-in functions
- API методы из pyspark.sql.functions

```
from pyspark.sql.functions import expr
df = spark.range(0,10)

# используем pyspark.sql.functions.expr
df.withColumn("pmod", expr("id % 2")).show()
```

```
+---+----+
| id|pmod|
+---+----+
|  0|   0|
|  1|   1|
|  2|   0|
|  3|   1|
|  4|   0|
|  5|   1|
|  6|   0|
|  7|   1|
|  8|   0|
|  9|   1|
+---+----+
```

```
from pyspark.sql.functions import expr
```

```
# используем SQL built-in functions
newCol = expr("""pmod(id, 2)""")
df.withColumn("pmod", newCol).show()
```

```
+---+----+
| id|pmod|
+---+----+
|  0|   0|
|  1|   1|
|  2|   0|
|  3|   1|
|  4|   0|
|  5|   1|
|  6|   0|
|  7|   1|
|  8|   0|
|  9|   1|
+---+----+
```

Выводы

- Spark обладает широким набором функций для работы с колонками разных типов, включая простые типы - строки, числа, и т. д., а также словари, массивы и структуры
- Встроенные функции принимают колонки org.apache.spark.sql.Column и возвращают org.apache.spark.sql.Column в большинстве случаев
- Встроенные функции доступны в двух местах - org.apache.spark.sql.functions и SQL built-in functions
- Встроенные функции можно (и нужно) использовать вместе - на вход во встроенные функции могут подаваться результаты встроенной функции, тк все они возвращают sql.Column

5.5. Пользовательские функции

В том случае, если функционала встроенных функций не хватает, можно написать пользовательскую функцию - UDF. Пользовательская функция может принимать до 16 аргументов. Соответствие Spark и Scala типов описано здесь. Определять как пользовательскую можно скалярные и агрегирующие функции.

Необходимо помнить, что null в Spark превращается в null внутри UDF

```
from pyspark.sql.types import IntegerType
from pyspark.sql.functions import udf

df = spark.range(0,10)

plusOne = udf(lambda val : val + 1, IntegerType())

df.withColumn("idPlusOne", plusOne(col("id"))).show(10, False)
```

```
+---+---------+
|id |idPlusOne|
+---+---------+
|0  |1        |
```

```
|1  |2        |
|2  |3        |
|3  |4        |
|4  |5        |
|5  |6        |
|6  |7        |
|7  |8        |
|8  |9        |
|9  |10       |
+---+---------+
```

Пользовательская функция может возвращать:

- простой тип - String, Long, Float, Boolean и т.д.
- массив - любые коллекции, наследующие list[T] и т. д.
- словарь - dict[A,B]
- инстанс case class'а Row

Реализуем функцию, которая возвращает имя хоста, на котором работает воркер:

```
import socket

@udf
def hostname():
    s = socket.gethostname()
    if s is not None:
        return s.upper()

df.withColumn("hostname", hostname()).show(10, False)
```

```
+---+------------+
|id |hostname    |
+---+------------+
|0  |D8FC1F507469|
|1  |D8FC1F507469|
|2  |D8FC1F507469|
|3  |D8FC1F507469|
|4  |D8FC1F507469|
|5  |D8FC1F507469|
|6  |D8FC1F507469|
|7  |D8FC1F507469|
|8  |D8FC1F507469|
|9  |D8FC1F507469|
+---+------------+
```

```
df = spark.range(0,10)

divideTwoBy = udf(lambda x: int(2/x) if x > 0 else None)# { (inputValue:
Long) => Try(2L / inputValue).toOption }

result = df.withColumn("divideTwoBy", divideTwoBy(col("id")))
result.printSchema()
result.show(10, False)
```

```
root
 |-- id: long (nullable = false)
 |-- divideTwoBy: string (nullable = true)

+---+-----------+
|id |divideTwoBy|
+---+-----------+
|0  |null       |
|1  |2          |
|2  |1          |
|3  |0          |
|4  |0          |
|5  |0          |
|6  |0          |
|7  |0          |
|8  |0          |
|9  |0          |
+---+-----------+
```

Выводы

- Пользовательские функции позволяют реализовать произвольный алгоритм и использовать его в DF API
- Пользовательские функции работают медленнее встроенных, поскольку при использовании встроенных функций Spark использует ряд оптимизаций, например векторизацию вычислений на уровне CPU
- Пользовательские функции на python/R работают значительно медленнее чем на Scala/Java, из-за необходимости передавать обработку данных вне JVM

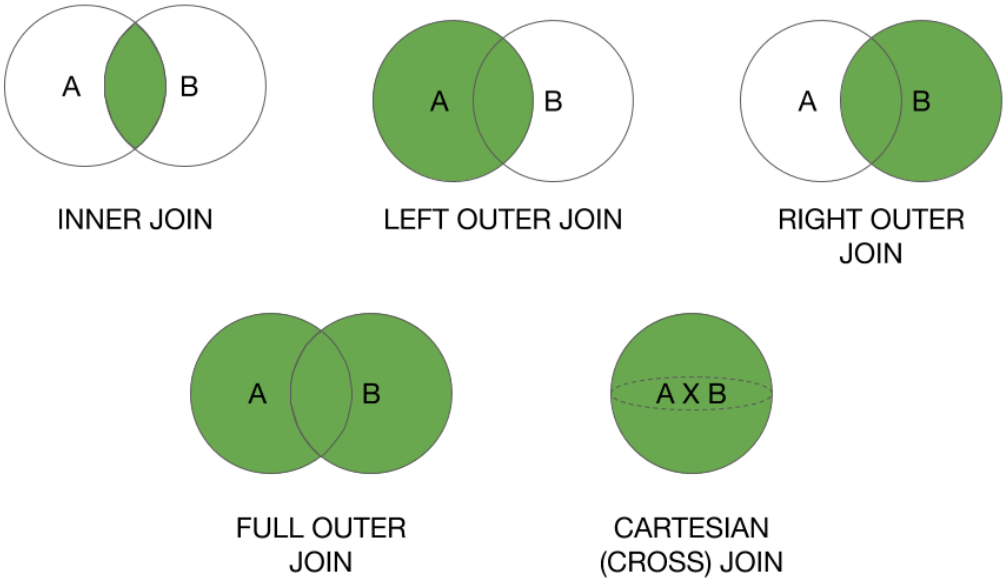
5.6. Соединения

Join'ы позволяют соединять два DF в один по заданным условиям.

По типу условия join'ы делятся на:

- equ-join - соединение по равенству одного или более ключей
- non-equ join - соединение по условию, отличному от равенства одного или более ключей

По методу соединения join'ы бывают:



Добавим новую колонку к датасету airports, в которой будет процент заданного типа аэропорта ко всем типам аэропорта по каждой стране. Первым шагом посчитаем число аэропортов каждого типа по стране:

```
aggTypeCountry = airports.groupBy(col('type'),
col('iso_country')).agg(count("*").alias("cnt_country_type"))

aggTypeCountry.show(10, 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               |
|medium_airport|EE         |5               |
|medium_airport|EG         |26              |
|medium_airport|FK         |2               |
|closed        |HN         |5               |
|small_airport |HU         |86              |
+--------------+-----------+----------------+
only showing top 10 rows
```

Теперь посчитаем количество аэропортов по каждой стране:

```
aggCountry =
airports.groupBy(col('iso_country')).agg(count("*").alias("cnt_country"))
aggCountry.show(5, False)
```

```
+-----------+-----------+
|iso_country|cnt_country|
+-----------+-----------+
|DZ         |61         |
```

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

Соединим получившиеся датасеты и получим процентное распределение типов аэропорта по стране

```
from pyspark.sql.functions import round

percent = aggTypeCountry \
        .join(aggCountry, "iso_country", "inner") \
        .select(col('iso_country'), col('type'),
round(col('cnt_country_type') / col('cnt_country') * 100,0).alias("percent"))
percent.show(10, False)
```

```
+-----------+-------------+-------+
|iso_country|type         |percent|
+-----------+-------------+-------+
|DZ         |small_airport |36.0   |
|DZ         |medium_airport|59.0   |
|DZ         |large_airport |2.0    |
|DZ         |closed        |3.0    |
|LT         |medium_airport|10.0   |
|LT         |large_airport |2.0    |
|LT         |small_airport |78.0   |
|LT         |heliport      |3.0    |
|LT         |closed        |7.0    |
|MM         |closed        |1.0    |
+-----------+-------------+-------+
only showing top 10 rows
```

Соединим полученный датасет с изначальным:

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

```
+-------+-----------+-----------+-------+
|ident  |iso_country|type       |percent|
+-------+-----------+-----------+-------+
|GB-0609|GB         |balloonport|0.0    |
|35JY   |US         |balloonport|0.0    |
|55NJ   |US         |balloonport|0.0    |
|RI16   |US         |balloonport|0.0    |
|AL-0001|AL         |closed     |45.0   |
```

```
|AL-0003|AL         |closed     |45.0   |
|LASK   |AL         |closed     |45.0   |
|AM-0006|AM         |closed     |31.0   |
|AO-0009|AO         |closed     |10.0   |
|AO-0014|AO         |closed     |10.0   |
|AO-0026|AO         |closed     |10.0   |
|AO-0031|AO         |closed     |10.0   |
|AR-0016|AR         |closed     |4.0    |
|AR-0648|AR         |closed     |4.0    |
|AR-0649|AR         |closed     |4.0    |
|SA30   |AR         |closed     |4.0    |
|X-VIE  |AT         |closed     |3.0    |
|AU-0042|AU         |closed     |1.0    |
|YBIK   |AU         |closed     |1.0    |
|YCAA   |AU         |closed     |1.0    |
+-------+-----------+-----------+-------+
only showing top 20 rows
```

Во всех наших джойнах присутствует массив Seq[String]. Это синтаксических сахар, позволяющий не переименовывать колонки датасетов, а просто указать, что соединение будет делаться по колонкам с именами, входящим в массив.

В общем случае условие джойна должно быть выражено в виде колонки sql.Column, например:

```
result = airports.join(percent, ["iso_country", "type"])
result \
    .select(col('*')) \
    .show(20, False)
```

```
+-----------+-------------+-------+----------------------------+-----------
+---------+----------+------------------+--------+---------+----------+-----
----------------------------------+-------+
|iso_country|type         |ident  |name
|elevation_ft|continent|iso_region|municipality
|gps_code|iata_code|local_code|coordinates
|percent|
+-----------+-------------+-------+----------------------------+-----------
+---------+----------+------------------+--------+---------+----------+-----
----------------------------------+-------+
|DZ         |small_airport|DZ-0001|Hamra Airport               |null
|AF       |DZ-30     |Hassi Bel Guebour  |null    |null     |null
|6.4936, 29.225107                      |36.0   |
|DZ         |small_airport|DATM   |Bordj Badji Mokhtar Airport  |1303
|AF       |DZ-01     |Bordj Badji Mokhtar|DATM    |BMW      |null
|0.923888981342, 21.375                 |36.0   |
|DZ         |small_airport|DATG   |In Guezzam Airport           |1312
|AF       |DZ-11     |In Guezzam         |DATG    |INF      |null      |5.75,
19.566999435424805                |36.0   |
|DZ         |small_airport|DAOY   |El Bayadh Airport            |4493
|AF       |DZ-32     |El Bayadh          |DAOY    |EBH      |null
|1.0925, 33.721666666699996             |36.0   |
|DZ         |small_airport|DAOS   |Sidi Bel Abbes Airport       |1614
|AF       |DZ-22     |Sidi Bel AbbÃ¨s    |DAOS    |BFW      |null
|-0.593275010586, 35.1717987061         |36.0   |
```

```
|DZ         |small_airport|DAOE   |Bou Sfer Airport             |187
|AF       |DZ-31     |null               |DAOE    |null     |null
|-0.8053889870643616, 35.73540115356445 |36.0   |
|DZ         |small_airport|DAOC   |Ouakda Airport               |2660
|AF       |DZ-08     |BÃ©char             |DAOC    |null     |null
|-2.1838901042938232, 31.642499923706055|36.0   |
|DZ         |small_airport|DAFI   |Tsletsi Airport              |3753
|AF       |DZ-17     |Djelfa             |DAFI    |QDJ      |null
|3.351, 34.6657                         |36.0   |
|DZ         |small_airport|DABO   |Oum el Bouaghi airport       |2980
|AF       |DZ-04     |Oum El Bouaghi     |DAEO    |QMH      |null
|7.270800113679999, 35.879699707        |36.0   |
|DZ         |small_airport|DAAX   |ChÃ©raga Airport              |396
|AF       |DZ-42     |ChÃ©raga            |DAAX    |null     |null
|2.9284, 36.7782                        |36.0   |
|DZ         |small_airport|DAAW   |Bordj Omar Driss Airport     |1207
|AF       |DZ-33     |Bordj Omar Driss   |DAAW    |null     |null
|6.8336100578308105, 28.131399154663086 |36.0   |
|DZ         |small_airport|DAAQ   |Ain Oussera Airport          |2132
|AF       |DZ-17     |null               |DAAQ    |null     |null
|2.8787100315093994, 35.52539825439453  |36.0   |
|DZ         |small_airport|DAAN   |Reggane Airport              |955
|AF       |DZ-01     |null               |DAAN    |null     |null
|0.285647, 26.7101                      |36.0   |
|DZ         |small_airport|DAAM   |Telerghma Airport            |2484
|AF       |DZ-43     |Telerghma          |DAAM    |null     |null
|6.36460018157959, 36.108699798583984   |36.0   |
|DZ         |small_airport|DAAF   |Aoulef Airport               |1017
|AF       |DZ-01     |Aoulef             |DAAF    |null     |null
|1.1111, 27.0624                        |36.0   |
|DZ         |small_airport|DA16   |Tindouf East Airport         |1425
|AF       |DZ-37     |null               |DA16    |null     |DA16
|-7.500110149383545, 27.585899353027344 |36.0   |
|DZ         |small_airport|DA15   |Saida Airport                |2444
|AF       |DZ-20     |null               |DA15    |null     |DA15
|0.15169399976730347, 34.89720153808594 |36.0   |
|DZ         |small_airport|DA14   |Mostaganem Airport           |732
|AF       |DZ-27     |null               |DA14    |MQV      |DA14
|0.149382993579, 35.9087982178          |36.0   |
|DZ         |small_airport|DA13   |Tinfouchy Airport            |1804
|AF       |DZ-37     |null               |DA13    |null     |DA13
|-5.82289981842041, 28.87929916381836   |36.0   |
|DZ         |small_airport|DA12   |El Abiodh Sidi Cheikh Airport|2965
|AF       |DZ-32     |null               |DA12    |null     |DA12
|0.52469402551651, 32.89849853515625    |36.0   |
+-----------+-------------+-------+-----------------------------+-----------
+---------+----------+------------------+--------+---------+----------+-----
-----------------------------------+-------+
only showing top 20 rows
```

При этом в данном выражении допускается использование встроенных функций, пользовательских функций и операторов сравнения. Однако следует помнить, что мы выполняем джойн двух распределенных датасетов и если условие соединения

будет плохо составлено, то Spark выполнит cross join, производительность которого будет "крайне мала" ©

Выводы:

- Spark поддерживает большое число типов соединений
- Условием соединения может быть Seq[String], либо sql.Column
- При использовании сложных условий соединения следует избегать тех, которые приведут к cross join

5.7. Оконные функции

Оконные функции позволяют делать функции над "окнами" данных.

Окно создается из класса org.apache.spark.sql.expressions.Window с указанием полей, определяющих границы окон и полей, определяющих порядок сортировки внутри окна:

```
window = Window.partitionBy("a", "b").orderBy("a")
```

Применяя окна, можно использовать такие полезные функции из org.apache.spark.sql.functions, как lag() и lead(), а также эффективно работать с данными time-series данными.

Выполним задачу с вычисление процента отношения типов аэропортов, используя оконные функции.

```
from pyspark.sql import Window

windowCountry = Window.partitionBy("iso_country")
windowTypeCountry = Window.partitionBy("type", "iso_country")

result = airports \
    .withColumn("cnt_country", count("*").over(windowCountry)) \
    .withColumn("cnt_country_type", count("*").over(windowTypeCountry)) \
    .withColumn("percent", round(lit(100) * col('cnt_country_type') /
col('cnt_country'), 2))

result.select(col('ident'), col('iso_country'), col('type'),
col('percent')).sample(0.2).show(20, False)
```

```
+-------+-----------+-----------+-------+
|ident  |iso_country|type       |percent|
+-------+-----------+-----------+-------+
|FR-0354|FR         |balloonport|0.11   |
|GB-0609|GB         |balloonport|0.25   |
|GB-0866|GB         |balloonport|0.25   |
|13M    |US         |balloonport|0.07   |
|2JY7   |US         |balloonport|0.07   |
|JY03   |US         |balloonport|0.07   |
|AL-0003|AL         |closed     |45.45  |
|AM-0007|AM         |closed     |30.77  |
|AR-0223|AR         |closed     |3.72   |
|AR-0443|AR         |closed     |3.72   |
|AR-0647|AR         |closed     |3.72   |
```

```
|AR-0649|AR         |closed     |3.72   |
|SAWA   |AR         |closed     |3.72   |
|LOAT   |AT         |closed     |2.74   |
|AU-0031|AU         |closed     |1.49   |
|LTB    |AU         |closed     |1.49   |
|YBER   |AU         |closed     |1.49   |
|YPLE   |AU         |closed     |1.49   |
|YWMD   |AU         |closed     |1.49   |
|YYCN   |AU         |closed     |1.49   |
+-------+-----------+-----------+-------+
only showing top 20 rows
```

Выводы:

- Оконные функции позволяют применять функции, применительно к окнам данных
- Окно определяется списком колонок и сортировкой
- Применение оконных функций приводит к shuffle

После завершения работы не забывайте останавливать SparkSession, чтобы освободить ресурсы кластера!

```
spark.stop
```

```
<bound method SparkSession.stop of <pyspark.sql.session.SparkSession object
at 0x7f1d8568d690>>
```

6. Настройка производительности в Apache Spark Dataframes

6.1. Планы выполнения задач

Любой job в Spark SQL имеет под собой план выполнения, кототорый генерируется на основе написанного запроса. План запроса содержит операторы, которые затем превращаются в Java код. Поскольку одну и ту же задачу в Spark SQL можно выполнить по-разному, полезно смотреть в планы выполнения, чтобы, например:

- убрать лишние shuffle
- убедиться, чтот тот или иной оператор будет выполнен на уровне источника, а не внутри Spark
- понять, как будет выполнен join

Планы выполнения доступны в двух видах:

- метод explain() у DF
- на вкладке SQL в Spark UI

Прочитаем датасет Airport Codes:

```
airports = spark.read.options(header='True',
inferSchema='True').csv("sample_data/airport-codes.csv")
```

```
airports.printSchema()
airports.show(n = 1, truncate = 100, vertical = True)
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)

-RECORD 0-----------------------------------------
 ident        | 00A
 type         | heliport
 name         | Total Rf Heliport
 elevation_ft | 11
 continent    | NA
 iso_country  | US
 iso_region   | US-PA
 municipality | Bensalem
 gps_code     | 00A
 iata_code    | null
 local_code   | 00A
 coordinates  | -74.93360137939453, 40.07080078125
only showing top 1 row
```

Используем метод explain, чтобы посмотреть план запроса. Наиболее интересным является физический план, т.к. он отражает фактически алгоритм обработки данных. В данном случае в плане присутствует единственный оператор FileScan csv:

```
airports.explain(extended = True)
```

```
== Parsed Logical Plan ==
Relation
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] csv

== Analyzed Logical Plan ==
ident: string, type: string, name: string, elevation_ft: int, continent:
string, iso_country: string, iso_region: string, municipality: string,
gps_code: string, iata_code: string, local_code: string, coordinates: string
Relation
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] csv

== Optimized Logical Plan ==
Relation
```

```
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] csv

== Physical Plan ==
FileScan csv
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] Batched: false, DataFilters: [], Format: CSV, Location:
InMemoryFileIndex(1 paths)[file:/content/sample_data/airport-codes.csv],
PartitionFilters: [], PushedFilters: [], ReadSchema:
struct<ident:string,type:string,name:string,elevation_ft:int,continent:string
,iso_country:string,...
```

Если нужно посмотреть только физический план:

```
airports.explain()
```

```
== Physical Plan ==
FileScan csv
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] Batched: false, DataFilters: [], Format: CSV, Location:
InMemoryFileIndex(1 paths)[file:/content/sample_data/airport-codes.csv],
PartitionFilters: [], PushedFilters: [], ReadSchema:
struct<ident:string,type:string,name:string,elevation_ft:int,continent:string
,iso_country:string,...
```

Выполним filter и проверим план выполнения. Читать план нужно снизу вверх. В плане появился новый оператор filter

```
from pyspark.sql.functions import col
airports.filter(col('type') == "small_airport").explain(extended = True)
```

```
== Parsed Logical Plan ==
'Filter ('type = small_airport)
+- Relation
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] csv

== Analyzed Logical Plan ==
ident: string, type: string, name: string, elevation_ft: int, continent:
string, iso_country: string, iso_region: string, municipality: string,
gps_code: string, iata_code: string, local_code: string, coordinates: string
Filter (type#706 = small_airport)
+- Relation
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] csv

== Optimized Logical Plan ==
Filter (isnotnull(type#706) AND (type#706 = small_airport))
```

```
+- Relation
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] csv

== Physical Plan ==
*(1) Filter (isnotnull(type#706) AND (type#706 = small_airport))
+- FileScan csv
[ident#705,type#706,name#707,elevation_ft#708,continent#709,iso_country#710,i
so_region#711,municipality#712,gps_code#713,iata_code#714,local_code#715,coor
dinates#716] Batched: false, DataFilters: [isnotnull(type#706), (type#706 =
small_airport)], Format: CSV, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(type), EqualTo(type,small_airport)], ReadSchema:
struct<ident:string,type:string,name:string,elevation_ft:int,continent:string
,iso_country:string,...
```

Выполним агрегацию и проверим план выполнения. В нем появляется три оператора: 2 HashAggregate и Exchange hashpartitioning.

Первый HashAggregate содержит функцию partial_count(1). Это означает, что внутри каждого воркера произойдет подсчет строк по каждому ключу. Затем происходит shuffle по ключу агрегата, после которого выполняется еще один HashAggregate с функцией count(1). Использование двух HashAggregate позволяет сократить количество передаваемых данных по сети.

```
airports.filter(col('type') ==
"small_airport").groupBy(col('iso_country')).count().explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[iso_country#710], functions=[count(1)])
   +- Exchange hashpartitioning(iso_country#710, 200), ENSURE_REQUIREMENTS,
[id=#1340]
      +- HashAggregate(keys=[iso_country#710], functions=[partial_count(1)])
         +- Project [iso_country#710]
            +- Filter (isnotnull(type#706) AND (type#706 = small_airport))
               +- FileScan csv [type#706,iso_country#710] Batched: false,
DataFilters: [isnotnull(type#706), (type#706 = small_airport)], Format: CSV,
Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(type), EqualTo(type,small_airport)], ReadSchema:
struct<type:string,iso_country:string>
```

##Выводы:

- Spark составляет физический план выполнения запроса на основании написанного вами кода
- Изучив план запроса, можно понять, какие операторы будут применены в ходе обработки ваших данных
- План выполнения запроса - один из основных инструментов оптимизации запроса

6.2. Оптимизация соединений и группировок

Оптимизация соединений и группировок При выполнении join двух DF важно следовать рекомендациям:

- фильтровать данные до join'а
- использовать equ join
- если можно путем увеличения количества данных применить equ join вместо non-equ join'а, то делать именно так
- всеми силами избегать cross-join'ов
- если правый DF помещается в памяти worker'а, использовать broadcast()

6.3. Алгоритмы соединений

- **Broadcast Hash Join**
  - equ join
  - broadcast hint or size
- **Shuffle Hash Joins**
  - equ join
- **Shuffle SortMerge Join**
  - equ join
  - sortable keys
- **Broadcast Nested Loop Join**
  - non-equ join + equ join
  - using broadcast
- **Cartesian Product**
  - non-equ join + equ join

Подготовим два датасета:

```
left = airports.select(col('type'), col('ident'), col('iso_country'))
right = airports.groupBy(col('type')).count()
```

##Broadcast Hash Join Когда один из DF мал и умещается в памяти, он будет транслироваться всем воркерам, и будет выполнено хэш-соединение.

- работает, когда условие - равенство одного или нескольких ключей
- работает, когда один из датасетов небольшой и полностью вмещается в память воркера
- может быть автоматически использован, либо явно через broadcast(df)

Если широковещательная сторона мала, BHJ может работать быстрее, чем другие алгоритмы соединения, так как не требуется перемешивание. Алгоритм:

3. оставляет левый датасет как есть
4. копирует правый датасет на каждый воркер
5. составляет hash map из правого датасета, где ключ - кортеж из колонок в условии соединения

6. итерируется по левому датасета внутри каждой партиции и проверяет наличие ключей в HashMap

Важно! На такой тип соединения сильно повлияют размеры рассылаемой таблицы и перекосы в данных.

```
from pyspark.sql.functions import broadcast

result = left.join(broadcast(right),["type"], "inner")

result.explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [type#706, ident#705, iso_country#710, count#824L]
   +- BroadcastHashJoin [type#706], [type#828], Inner, BuildRight, false
      :- Filter isnotnull(type#706)
      :  +- FileScan csv [ident#705,type#706,iso_country#710] Batched: false,
DataFilters: [isnotnull(type#706)], Format: CSV, Location:
InMemoryFileIndex(1 paths)[file:/content/sample_data/airport-codes.csv],
PartitionFilters: [], PushedFilters: [IsNotNull(type)], ReadSchema:
struct<ident:string,type:string,iso_country:string>
      +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string,
true]),false), [id=#1380]
         +- HashAggregate(keys=[type#828], functions=[count(1)])
            +- Exchange hashpartitioning(type#828, 200), ENSURE_REQUIREMENTS,
[id=#1377]
               +- HashAggregate(keys=[type#828],
functions=[partial_count(1)])
                  +- Filter isnotnull(type#828)
                     +- FileScan csv [type#828] Batched: false, DataFilters:
[isnotnull(type#828)], Format: CSV, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(type)], ReadSchema: struct<type:string>
```

##Shuffle Hash Joins Когда таблица относительно велика, использование широковещательной передачи может вызвать проблемы с памятью на стороне драйвера и исполнителя. В этом случае будет использоваться Shuffle Hash Join. Это дорогостоящее соединение, поскольку оно включает в себя как перемешивание, так и хеширование. Кроме того, для поддержки хеш-таблицы требуется память и вычисления.

Shuffle Hash Join выполняется в два этапа:

7. Перемешивание: данные из таблиц соединения разделяются на разделы на основе ключа соединения. Он перемешивает данные между разделами, чтобы те же ключи соединения записи были назначены соответствующим разделам.
8. Хеш-соединение: для данных на каждом разделе выполняется классический алгоритм хеш-соединения с одним узлом.

Производительность Shuffle Hash Join является наилучшей, когда данные распределяются равномерно с ключом, к которому вы присоединяетесь, и у вас есть достаточное количество ключей для параллелизма.

```
spark.conf.set("spark.sql.join.preferSortMergeJoin", "false")
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "-1")
result = left.join(right, ["type"], "inner")

result.explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [type#706, ident#705, iso_country#710, count#824L]
   +- SortMergeJoin [type#706], [type#1034], Inner
      :- Sort [type#706 ASC NULLS FIRST], false, 0
      :  +- Exchange hashpartitioning(type#706, 200), ENSURE_REQUIREMENTS,
[id=#1849]
      :     +- Filter isnotnull(type#706)
      :        +- FileScan csv [ident#705,type#706,iso_country#710] Batched:
false, DataFilters: [isnotnull(type#706)], Format: CSV, Location:
InMemoryFileIndex(1 paths)[file:/content/sample_data/airport-codes.csv],
PartitionFilters: [], PushedFilters: [IsNotNull(type)], ReadSchema:
struct<ident:string,type:string,iso_country:string>
      +- Sort [type#1034 ASC NULLS FIRST], false, 0
         +- HashAggregate(keys=[type#1034], functions=[count(1)])
            +- Exchange hashpartitioning(type#1034, 200),
ENSURE_REQUIREMENTS, [id=#1845]
               +- HashAggregate(keys=[type#1034],
functions=[partial_count(1)])
                  +- Filter isnotnull(type#1034)
                     +- FileScan csv [type#1034] Batched: false, DataFilters:
[isnotnull(type#1034)], Format: CSV, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(type)], ReadSchema: struct<type:string>
```

##Shuffle Sort-merge Join SMJ включает в себя перетасовку данных для получения одного и того же ключа соединения с тем же рабочим узлом, а затем выполнение операции сортировки-слияния на уровне раздела в рабочих узлах. Разделы сортируются по ключу соединения перед операцией соединения.

- работает, когда ключи соединения в обоих датасета являются сортируемыми
- репартиционирует оба датасета в 200 партиций по ключу (ключам) соединения
- сортирует партиции каждого из датасетов по ключу (ключам) соединения
- используя сравнение левого и правого ключей, обходит каждую пару партиций и соединяет строки с одинаковыми ключами

Он состроит из 3 фаз:

9. Этап перемешивания: обе большие таблицы будут перераспределены в соответствии с ключами соединения между разделами в кластере.

10. Этап сортировки: параллельная сортировка данных в каждом разделе.
11. Фаза слияния: объедините отсортированные и секционированные данные. Это объединение набора данных путем перебора элементов и объединения строк, имеющих одинаковое значение для ключей соединения.

SMJ большую часть времени работает лучше, чем другие соединения, и имеет очень масштабируемый подход, поскольку он устраняет накладные расходы на хеширование и не требует, чтобы все данные помещались в памяти.

```
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "-1")

result = left.join(right, ["type"], "inner")

result.explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [type#706, ident#705, iso_country#710, count#824L]
   +- SortMergeJoin [type#706], [type#846], Inner
      :- Sort [type#706 ASC NULLS FIRST], false, 0
      :  +- Exchange hashpartitioning(type#706, 200), ENSURE_REQUIREMENTS,
[id=#1422]
      :     +- Filter isnotnull(type#706)
      :        +- FileScan csv [ident#705,type#706,iso_country#710] Batched:
false, DataFilters: [isnotnull(type#706)], Format: CSV, Location:
InMemoryFileIndex(1 paths)[file:/content/sample_data/airport-codes.csv],
PartitionFilters: [], PushedFilters: [IsNotNull(type)], ReadSchema:
struct<ident:string,type:string,iso_country:string>
      +- Sort [type#846 ASC NULLS FIRST], false, 0
         +- HashAggregate(keys=[type#846], functions=[count(1)])
            +- Exchange hashpartitioning(type#846, 200), ENSURE_REQUIREMENTS,
[id=#1418]
               +- HashAggregate(keys=[type#846],
functions=[partial_count(1)])
                  +- Filter isnotnull(type#846)
                     +- FileScan csv [type#846] Batched: false, DataFilters:
[isnotnull(type#846)], Format: CSV, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(type)], ReadSchema: struct<type:string>
```

##Broadcast Nested Loop Join Broadcast Nested Loop Join выбирается, когда оно не превышает пороговое значение для широковещательной передачи. Он поддерживает как Equi-Joins, так и Non-Equi-Joins.

- работает, когда один из датасетов небольшой и полностью вмещается в память воркера
- оставляет левый датасет как есть
- копирует правый датасет на каждый воркер
- проходится вложенным циклом по каждой партиции левого датасета и копией правого датасета и проверяет условие

- может быть автоматически использован, либо явно через broadcast(df)

```
from pyspark.sql.functions import udf

spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "-1")

# Не смотря на то, что UDF сравнивает два ключа, Spark ничего про нее не знает
# и не может применить BroadcastHashJoin или SortMergeJoin
##compare_udf = udf(lambda leftVal, rightVal: leftVal == rightVal )

##joinExpr = compare_udf(col("left.type"), col("right.type"))

result = left.alias("left").crossJoin(broadcast(right).alias("right"))

result.explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- BroadcastNestedLoopJoin BuildRight, Cross
   :- Project [type#706, ident#705, iso_country#710]
   :  +- FileScan csv [ident#705,type#706,iso_country#710] Batched: false,
DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [], ReadSchema:
struct<ident:string,type:string,iso_country:string>
   +- BroadcastExchange IdentityBroadcastMode, [id=#1451]
      +- HashAggregate(keys=[type#862], functions=[count(1)])
         +- Exchange hashpartitioning(type#862, 200), ENSURE_REQUIREMENTS,
[id=#1448]
            +- HashAggregate(keys=[type#862], functions=[partial_count(1)])
               +- FileScan csv [type#862] Batched: false, DataFilters: [],
Format: CSV, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [], ReadSchema: struct<type:string>
```

##Cartesian Product Если тип соединения внутренний, как и нет ключей соединения, будет выбрано декартово соединение. Cross Join вычисляет декартово произведение двух таблиц.

- Создает пары из каждой партиции левого датасета с каждой партицией правого датасета, релоцирует каждую пару на один воркер и проверяет условие соединения
- на выходе создает N*M партиций
- работает медленнее остальных и часто приводит к OOM воркеров

```
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "-1")

# Не смотря на то, что UDF сравнивает два ключа, Spark ничего про нее не знает
# и не может применить BroadcastHashJoin или SortMergeJoin
#compare_udf = udf(lambda leftVal, rightVal: leftVal == rightVal )

#joinExpr = compare_udf(col("left.type"), col("right.type"))
```

```
result = left.alias("left").crossJoin(right.alias("right"))

result.explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- CartesianProduct
   :- Project [type#706, ident#705, iso_country#710]
   :  +- FileScan csv [ident#705,type#706,iso_country#710] Batched: false,
DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [], ReadSchema:
struct<ident:string,type:string,iso_country:string>
   +- HashAggregate(keys=[type#879], functions=[count(1)])
      +- Exchange hashpartitioning(type#879, 200), ENSURE_REQUIREMENTS,
[id=#1474]
         +- HashAggregate(keys=[type#879], functions=[partial_count(1)])
            +- FileScan csv [type#879] Batched: false, DataFilters: [],
Format: CSV, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airport-codes.csv], PartitionFilters: [],
PushedFilters: [], ReadSchema: struct<type:string>
```

	Cartesian Join	Broadcast Hash Join	Sort merge Join	Shuffle Hash Join	Broadcast Nested Loop Join
Inner-Join	✓	✓	✓	✓	✓
Left-Join	✗	✓	✓	✓	✓
Right-Join	✗	✓	✓	✓	✓
Cross-Join	✗	✓	✓	✓	✓
Left-Semi_Join	✗	✓	✓	✓	✓
Left-Anti-Semi-Join	✗	✓	✓	✓	✓
Outer-Join	✗	✗	✓	✓	✓
Support Equi Join	✓	✓	✓	✓	✓
Support Non Equi Join	✓	✗	✗	✗	✓
Require Sortable Join key	✗	✗	✓	✗	✗

##Снижение количества shuffle В ряде случаев можно уйти от лишних shuffle операций при выполнении соединения. Для этого оба DF должны иметь одинаковое партиционирование - одинаковое количество партиций и ключ партиционирования, совпадающий с ключом соединения.

Разница между планами выполнения будет хорошо видна в Spark UI на графе выполнения в Jobs и плане выполнения в SQL.

```
left = airports
right = airports.groupBy(col('type')).count()
joined = left.join(right, ["type"])
joined.count()
```

```
56226
```

```
airportsRep = airports.repartition(200, col("type"))
left = airportsRep
right = airports.groupBy(col('type')).count()
joined = left.join(right, ["type"])
joined.count()
```

```
56226
```

##Выводы:

- В Spark используются 5 видов соединений: Broadcast Hash Join, Shuffle Sort-merge Join, Shuffle SortMergeJoin, Broadcast NestedLoop Join, Cartesian Product
- Выбор алгоритма основывается на условии соединения и размере датасетов
- Cartesian Product обладает самой низкой вычислительной эффективностью и его по возможности стоит избегать

6.4. Управление схемой данных

В DF API каждая колонка имеет свой тип. Он может быть:

- скаляром - StringType, IntegerType и т. д.
- массивом - ArrayType(T)
- словарем MapType(K, V)
- структурой - StructType()

DF целиком также имеет схему, описанную с помощью класса StructType

Посмотреть список колонок можно с помощью атрибута columns:

```
print(airports.columns)
```

```
['ident', 'type', 'name', 'elevation_ft', 'continent', 'iso_country',
'iso_region', 'municipality', 'gps_code', 'iata_code', 'local_code',
'coordinates']
```

Схема DF доступна через атрибут schema

```
schema = airports.schema
schema
```

```
StructType(List(StructField(ident,StringType,true),StructField(type,StringTyp
e,true),StructField(name,StringType,true),StructField(elevation_ft,IntegerTyp
e,true),StructField(continent,StringType,true),StructField(iso_country,String
Type,true),StructField(iso_region,StringType,true),StructField(municipality,S
tringType,true),StructField(gps_code,StringType,true),StructField(iata_code,S
tringType,true),StructField(local_code,StringType,true),StructField(coordinat
es,StringType,true)))
```

apply() метод возвращает поле структуры по имени, как в словаре

```
field = schema["ident"]
field
```

```
StructField(ident,StringType,true)
```

StructField обладает атрибутами name и dataType:

```
from pyspark.sql.types import StringType

name = field.name
print(name)
fieldType = field.dataType
print(fieldType)

if type(fieldType) == StringType:
    print("This is string.")
else:
    print("This is not string!")
```

```
ident
StringType
This is string.
```

Метод simpleString можно использовать, чтобы получить DDL схемы в виде строки:

```
fieldType.simpleString()
```

```
{"type":"string"}
```

```
airportSchema = schema.simpleString()
airportSchema
```

```
{"type":"string"}
```

Схема может быть использована:

- при чтении источника
- при работе с JSON

```
airports = spark.read.options(header='True',
inferSchema='True').csv("sample_data/airport-codes.csv")
airports.printSchema()
airports.show(n = 1, truncate = 100, vertical = True)
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)
```

```
-RECORD 0-----------------------------------------
 ident        | 00A
 type         | heliport
 name         | Total Rf Heliport
 elevation_ft | 11
 continent    | NA
 iso_country  | US
 iso_region   | US-PA
 municipality | Bensalem
 gps_code     | 00A
 iata_code    | null
 local_code   | 00A
 coordinates  | -74.93360137939453, 40.07080078125
only showing top 1 row
```

Схема может быть создана вручную:

```
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType, BooleanType

structureData = [
    (("James","","Smith"),36636,"M",3100),
    (("Michael","Rose",""),40288,"M",4300),
    (("Robert","","Williams"),42114,"M",1400),
    (("Maria","Anne","Jones"),39192,"F",5500),
    (("Jen","Mary","Brown"),5659,"F",-1)
  ]
structureSchema = StructType([
        StructField('name', StructType([
             StructField('firstname', StringType(), True),
             StructField('middlename', StringType(), True),
             StructField('lastname', StringType(), True)
             ])),
         StructField('id', IntegerType(), True),
         StructField('gender', StringType(), True),
         StructField('salary', IntegerType(), True)
         ])

df2 = spark.createDataFrame(data=structureData,schema=structureSchema)
df2.printSchema()

root
 |-- name: struct (nullable = true)
 |    |-- firstname: string (nullable = true)
 |    |-- middlename: string (nullable = true)
 |    |-- lastname: string (nullable = true)
 |-- id: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: integer (nullable = true)
```

Схема также может быть получена из JSON строки:

```
import json
schemaFromJson = StructType.fromJson(json.loads(structureSchema.json()))
```

```
df3 =
spark.createDataFrame(spark.sparkContext.parallelize(structureData),schemaFro
mJson)
df3.printSchema()
```

```
root
 |-- name: struct (nullable = true)
 |    |-- firstname: string (nullable = true)
 |    |-- middlename: string (nullable = true)
 |    |-- lastname: string (nullable = true)
 |-- id: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: integer (nullable = true)
```

Чтобы изменить тип колонки, следует использовать метод cast. Данная операция может как возвращать null, так и бросать исключение

```
airports.select(col('elevation_ft').cast("string")).printSchema()
airports.select(col('elevation_ft').cast("string")).show(1, False)
```

```
root
 |-- elevation_ft: string (nullable = true)

+------------+
|elevation_ft|
+------------+
|11          |
+------------+
only showing top 1 row
```

```
airports.select(col('type').cast("float")).printSchema()
airports.select(col('type').cast("float")).show(1, False)
```

```
root
 |-- type: float (nullable = true)

+----+
|type|
+----+
|null|
+----+
only showing top 1 row
```

##Выводы:

- Spark использует схемы для описания типов колонок, схемы всего DF, чтения источников и для работы с JSON
- Схема представляет собой инстанс класса StructType
- Колонки в Spark могут иметь любой тип. При этом вложенность словарей, массивов и структур не ограничена

6.5. Оптимизатор запросов Catalyst

Catalyst выполняет оптимизацию запросов с целью ускорения их выполнения и применяет следующие методы:

- Column projection
- Partition pruning
- Predicate pushdown
- Simplify casts
- Constant folding
- Combine filters

Подготовим датасет для демонстрации работы Catalyst:

```
airports \
    .write \
    .format("parquet") \
    .partitionBy("iso_country") \
    .mode("overwrite") \
    .save("sample_data/airports.parquet")

airportPq = spark.read.parquet("sample_data/airports.parquet")
```

##Column projection Данный механизм позволяет избегать вычитывания ненужных колонок при работе с источниками

```
selected = airportPq
selected.cache()
selected.count()
selected.unpersist()
selected.explain()
```

```
== Physical Plan ==
*(1) ColumnarToRow
+- FileScan parquet
[ident#1333,type#1334,name#1335,elevation_ft#1336,continent#1337,iso_region#1
338,municipality#1339,gps_code#1340,iata_code#1341,local_code#1342,coordinate
s#1343,iso_country#1344] Batched: true, DataFilters: [], Format: Parquet,
Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airports.parquet], PartitionFilters: [],
PushedFilters: [], ReadSchema:
struct<ident:string,type:string,name:string,elevation_ft:int,continent:string
,iso_region:string,m...
```

```
selected = airportPq.select(col('ident'))
selected.cache()
selected.count()
selected.unpersist()
selected.explain()
```

```
== Physical Plan ==
*(1) Project [ident#1333]
```

```
+- *(1) ColumnarToRow
   +- FileScan parquet [ident#1333,iso_country#1344] Batched: true,
DataFilters: [], Format: Parquet, Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airports.parquet], PartitionFilters: [],
PushedFilters: [], ReadSchema: struct<ident:string>
```

##Partition pruning Данный механизм позволяет избежать чтения ненужных партиций

```
filtered = airportPq.filter(col('iso_country') == "RU")
filtered.count()
filtered.explain()
```

```
== Physical Plan ==
*(1) ColumnarToRow
+- FileScan parquet
[ident#1333,type#1334,name#1335,elevation_ft#1336,continent#1337,iso_region#1
338,municipality#1339,gps_code#1340,iata_code#1341,local_code#1342,coordinate
s#1343,iso_country#1344] Batched: true, DataFilters: [], Format: Parquet,
Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airports.parquet], PartitionFilters:
[isnotnull(iso_country#1344), (iso_country#1344 = RU)], PushedFilters: [],
ReadSchema:
struct<ident:string,type:string,name:string,elevation_ft:int,continent:string
,iso_region:string,m...
```

##Predicate pushdown Данный механизм позволяет "протолкнуть" условия фильтрации данных на уровень datasource

```
filtered = airportPq.filter(col('iso_region') == "RU")
filtered.count()
filtered.explain()
```

```
== Physical Plan ==
*(1) Filter (isnotnull(iso_region#1338) AND (iso_region#1338 = RU))
+- *(1) ColumnarToRow
   +- FileScan parquet
[ident#1333,type#1334,name#1335,elevation_ft#1336,continent#1337,iso_region#1
338,municipality#1339,gps_code#1340,iata_code#1341,local_code#1342,coordinate
s#1343,iso_country#1344] Batched: true, DataFilters:
[isnotnull(iso_region#1338), (iso_region#1338 = RU)], Format: Parquet,
Location: InMemoryFileIndex(1
paths)[file:/content/sample_data/airports.parquet], PartitionFilters: [],
PushedFilters: [IsNotNull(iso_region), EqualTo(iso_region,RU)], ReadSchema:
struct<ident:string,type:string,name:string,elevation_ft:int,continent:string
,iso_region:string,m...
```

##Simplify casts Данный механизм убирает ненужные cast

```
result = spark.range(0,10).select(col('id').cast("int"))
result.explain()
```

```
== Physical Plan ==
*(1) Project [cast(id#1809L as int) AS id#1811]
+- *(1) Range (0, 10, step=1, splits=2)
```

```
result = spark.range(0,10).select(col('id').cast("long"))
result.explain()
```

```
== Physical Plan ==
*(1) Range (0, 10, step=1, splits=2)
```

##Constant folding Данный механизм сокращает количество констант, используемых в физическом плане

```
from pyspark.sql.functions import lit
result = spark.range(0,10).select((lit(3) >  lit(0)).alias("foo"))
result.explain()
```

```
== Physical Plan ==
*(1) Project [true AS foo#1815]
+- *(1) Range (0, 10, step=1, splits=2)
```

```
result = spark.range(0,10).select((col('id') >  0).alias("foo"))
result.explain()
```

```
== Physical Plan ==
*(1) Project [(id#1817L > 0) AS foo#1819]
+- *(1) Range (0, 10, step=1, splits=2)
```

##Combine filters Данный механизм объединяет фильтры

```
result = spark.range(0,10).filter(col('id') > 0).filter(col('id') !=
5).filter(col('id') < 10)
result.explain()
```

```
== Physical Plan ==
*(1) Filter ((id#1821L > 0) AND (NOT (id#1821L = 5) AND (id#1821L < 10)))
+- *(1) Range (0, 10, step=1, splits=2)
```

```
spark.stop
```

```
<bound method SparkSession.stop of <pyspark.sql.session.SparkSession object
at 0x7fcf650a6690>>
```

7. Работа с источниками данных в Apache Spark

В этой записной книжке будут использоваться дополнительные драйверы, поэтому удостоверьтесь, что они прописаны в параметре `spark.jars` сеанса.

```
for i in spark.sparkContext.getConf().getAll():
  print(i)
```

```
('spark.driver.host', '3c91d9574429')
('spark.repl.local.jars',
'file:///content/sqlite-jdbc-3.34.0.jar,file:///content/postgresql-42.2.23.ja
r')
('spark.app.initial.jar.urls',
'spark://3c91d9574429:45901/jars/postgresql-42.2.23.jar,spark://3c91d9574429:
45901/jars/sqlite-jdbc-3.34.0.jar')
('spark.executor.id', 'driver')
('spark.sql.warehouse.dir', 'file:/content/spark-warehouse')
('spark.jars',
'file:///content/sqlite-jdbc-3.34.0.jar,file:///content/postgresql-42.2.23.ja
r')
('spark.app.name', 'pyspark-shell')
('spark.app.startTime', '1638888457789')
('spark.rdd.compress', 'True')
('spark.serializer.objectStreamReset', '100')
('spark.master', 'local[*]')
('spark.submit.pyFiles', '')
('spark.driver.port', '45901')
('spark.submit.deployMode', 'client')
('spark.app.id', 'local-1638888459449')
('spark.ui.showConsoleProgress', 'true')
```

7.1. Обзор источников данных

Spark - это платформа для **обработки** распределенных данных. Она не отвечает за хранение данных и не завязана на какую-либо БД или формат хранения, что позволяет разработать коннектор для работы с любым источником. Часть распространенных источников доступна "из коробки", часть - в виде сторонних библиотек.

На текущий момент Spark DF API позволяет работать (читать и писать) с большим набором источников:

- Текстовые файлы:
  - json
  - text
  - csv
- Бинарные файлы:
  - orc
  - parquet
  - avro
  - BLOB

- Базы данных:
  - jdbc
  - elastic
  - cassandra
  - redis
  - mongo
- Стриминг системы:
  - kafka

Для текстовых и бинарных файлов поддерживаются различные кодеки сжатия (например lzo, snappy, gzip)

Добавление поддержки

Чтобы добавить поддержку источника в проект, необходимо:

- найти нужный пакет на mvnrepository
  - выбрать актуальную версию для `Scala 2.11`
  - скачать jar или скопировать команду для нужной системы сборки
- добавить зависимость в `libraryDependencies` в файле `build.sbt`
  `libraryDependencies += "org.elasticsearch" %% "elasticsearch-spark-20" % "7.7.0"`
- добавить зависимость в приложение одним из способов:
  - добавить зависимость в **spark-submit**: `spark-submit --packages org.elasticsearch:elasticsearch-spark-20_2.11:7.7.0`
  - добавить jar файл в **spark-submit**: `spark-submit --jars /path/to/elasticsearch-spark-20_2.11-7.7.0.jar`
  - добавить зависимость в **spark-defaults.conf**: `spark.jars.packages org.elasticsearch:elasticsearch-spark-20_2.11:7.7.0`
  - добавить jar файл в **spark-defaults.conf**: `spark.jars /path/to/elasticsearch-spark-20_2.11-7.7.0.jar`
  - в коде через `spark.sparkContext.addJar()`

##Использование в коде Конфиги источника задаются одним из способов:

- через **spark-submit**:
  - `spark-submit --conf spark.es.nodes=localhost:9200`
- в **spark-defaults.conf**:
  - `spark.es.nodes localhost:9200`
- в коде через **SparkSession**:
  - `spark.conf.set("spark.es.nodes", "localhost:9200")`
- в коде при чтении:
  - `df = spark.read.format("elastic").option("es.nodes", "localhost:9200")...`

- df = spark.read.format("elastic").options(Map("es.nodes" -> "localhost:9200"))...

- в коде при записи:
  - df.write.format("elastic").option("es.nodes", "localhost:9200")...
  - df.write.format("elastic").options(Map("es.nodes" -> "localhost:9200"))...

##Выводы:

- Spark позволяет работать с болшим количеством источников
- Поддержка источника всегда добавлеяется на уровне JVM (даже для pyspark) путем добавления в `java classpath` нужного класса
- Добавить поддержку источника можно по-разному, однако в большинстве случаев следует избегать "хардкода"

7.2. Текстовые форматы txt, csv, json

Spark позволяет хранить данные в текстовом виде в форматах text, json, csv

- json - JSON строки (не массив JSON документов, а именно раздельные строки, разделенные \n)
- csv - плоские данные с разделителем
- text просто текстовые строки, вычитываются как DF с единственной колонкой value: String

  ##Преимущества:
- простота интеграции
- поддержка партиционирования и сжатия

  ##Недостатки: *отсутствие оптимизаций* низкая скорость чтения сжатых данных *слабая типизация* нельзя писать несколько DF в один файл *нельзя дописывать/менять данные в существующих файлах

Прочитаем датасет Airport Codes:

```
airports = spark.read.csv("sample_data/airport-codes.csv", header=True,
inferSchema=True)
airports.printSchema()
airports.show(n = 1, truncate = 100, vertical = True)
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)
```

```
-RECORD 0-----------------------------------------
 ident        | 00A
 type         | heliport
 name         | Total Rf Heliport
 elevation_ft | 11
 continent    | NA
 iso_country  | US
 iso_region   | US-PA
 municipality | Bensalem
 gps_code     | 00A
 iata_code    | null
 local_code   | 00A
 coordinates  | -74.93360137939453, 40.07080078125
only showing top 1 row
```

```
airports.write.mode("overwrite").csv("sample_data/airport-codes-csv")
```

```
!ls sample_data/airport-codes-csv -hlt
```

```
total 6.1M
-rw-r--r-- 1 root root    0 Dec  7 15:12 _SUCCESS
-rw-r--r-- 1 root root 5.1M Dec  7 15:12
part-00000-239c01f4-803c-44e2-a37e-3a0747e0f655-c000.csv
-rw-r--r-- 1 root root 1.1M Dec  7 15:12
part-00001-239c01f4-803c-44e2-a37e-3a0747e0f655-c000.csv
```

Если мы попытаемся прочитать его с помощью spark.read, используя старый код, получим ошибку - в качестве схемы Spark взял одну из строк, содержащую данные.

```
airports = spark.read.csv('sample_data/airport-codes-csv', header=True,
inferSchema=True)
airports.printSchema()
airports.show(n = 1, truncate = 100, vertical = True)
```

```
root
 |-- 00A0: string (nullable = true)
 |-- heliport: string (nullable = true)
 |-- Total Rf Heliport: string (nullable = true)
 |-- 11: integer (nullable = true)
 |-- NA: string (nullable = true)
 |-- US: string (nullable = true)
 |-- US-PA: string (nullable = true)
 |-- Bensalem: string (nullable = true)
 |-- 00A8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- 00A10: string (nullable = true)
 |-- -74.93360137939453, 40.07080078125: string (nullable = true)
```

```
-RECORD 0-----------------------------------------------------
 00A0                                | 00AA
 heliport                            | small_airport
 Total Rf Heliport                   | Aero B Ranch Airport
 11                                  | 3435
 NA                                  | NA
 US                                  | US
```

```
 US-PA                                | US-KS
 Bensalem                             | Leoti
 00A8                                 | 00AA
 _c9                                  | null
 00A10                                | 00AA
 -74.93360137939453, 40.07080078125 | -101.473911, 38.704022
only showing top 1 row
```

Поищем шапку в сырых данных - ее там не будет:

```
spark.read.text("sample_data/airport-codes-csv").show()
```

```
+--------------------+
|               value|
+--------------------+
|00A,heliport,Tota...|
|00AA,small_airpor...|
|00AK,small_airpor...|
|00AL,small_airpor...|
|00AR,closed,Newpo...|
|00AS,small_airpor...|
|00AZ,small_airpor...|
|00CA,small_airpor...|
|00CL,small_airpor...|
|00CN,heliport,Kit...|
|00CO,closed,Cass ...|
|00FA,small_airpor...|
|00FD,heliport,Rin...|
|00FL,small_airpor...|
|00GA,small_airpor...|
|00GE,heliport,Caf...|
|00HI,heliport,Kau...|
|00ID,small_airpor...|
|00IG,small_airpor...|
|00II,heliport,Bai...|
+--------------------+
only showing top 20 rows
```

Если прочитать с header=false, названия колонок будут автоматически сгенерированы:

```
airports = spark.read.csv('sample_data/airport-codes-csv', header=False,
inferSchema=True)
airports.printSchema()
airports.show(n = 1, truncate = 100, vertical = True)
```

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: integer (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
```

```
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)

-RECORD 0---------------------------------
 _c0  | 00A
 _c1  | heliport
 _c2  | Total Rf Heliport
 _c3  | 11
 _c4  | NA
 _c5  | US
 _c6  | US-PA
 _c7  | Bensalem
 _c8  | 00A
 _c9  | null
 _c10 | 00A
 _c11 | -74.93360137939453, 40.07080078125
only showing top 1 row
```

Имея шапку в виде строки, мы можем создать схему самостоятельно:

```
row = spark.read.text("sample_data/airport-codes.csv").take(1)
print(row[0].value)

from pyspark.sql.types import StructType, StructField, StringType,
IntegerType

schema = StructType(list(map(lambda x: StructField(x,IntegerType()) if (x ==
'elevation_ft') else StructField(x,StringType()), row[0].value.split(','))))

airports = spark.read.schema(schema).csv('sample_data/airport-codes-csv',
header=False, inferSchema=True)
airports.printSchema()
airports.show(n = 1, truncate = 100, vertical = True)

ident,type,name,elevation_ft,continent,iso_country,iso_region,municipality,gp
s_code,iata_code,local_code,coordinates
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)

-RECORD 0-----------------------------------------
 ident        | 00A
```

```
 type         | heliport
 name         | Total Rf Heliport
 elevation_ft | 11
 continent    | NA
 iso_country  | US
 iso_region   | US-PA
 municipality | Bensalem
 gps_code     | 00A
 iata_code    | null
 local_code   | 00A
 coordinates  | -74.93360137939453, 40.07080078125
only showing top 1 row
```

Сохраним данные в csv с включенной компрессией gzip:

```
airports.write.mode("overwrite").option("codec",
"gzip").csv("sample_data/airport-codes-gz")
```

```
!ls sample_data/airport-codes-gz -hlt
```

```
total 2.1M
-rw-r--r-- 1 root root    0 Dec  7 15:15 _SUCCESS
-rw-r--r-- 1 root root 1.8M Dec  7 15:15
part-00000-86115d5b-81cc-4566-addb-4e61647cd961-c000.csv.gz
-rw-r--r-- 1 root root 346K Dec  7 15:15
part-00001-86115d5b-81cc-4566-addb-4e61647cd961-c000.csv.gz
```

Данные стали занимать меньше места, но у этого решения есть существенный минус - при чтении каждый сжатый файл превращается ровно в 1 партицию в DF. При работе с большими датасетами это означает:

- если файлов мало и они большие, то воркерам может не хватить памяти для их чтения (тк один сжатый файл нельзя разбить на несколько партиций
- если файлов много и они маленькие - мы получаем увеличенный расход памяти в heap HDFS NameNode (память расходуется пропорционально количеству файлов на HDFS из расчета 1 ГБ памяти на 1 000 000 файлов)

Сохраним датасет в формате json с партиционирование по колонкам iso_region и iso_country:

```
airports.write.mode("overwrite").partitionBy("iso_country",
"iso_region").json("sample_data/airport-codes-json")
```

Такой формат хранения позволит использовать partition pruning и быстро фильтровать данные по колонкам iso_region и iso_country

Теперь сохраним датасет в формат text. Для этого нам необходимо подгтовить DF, в котором будет единственная колонка value: String

```
from pyspark.sql.functions import col, concat, lit
airports.select(concat(airports.ident, lit(","), col('type'), lit(","),
col("name") , lit(","), airports.continent) \
    .alias("value")) \
    .write \
    .mode("overwrite") \
```

```
    .format("text") \
    .save("sample_data/airport-codes-txt")
```

Файловые форматы не имеют автоматической валидации данных при записи, поэтому достаточно легко ошибиться и записать данные в другом формате. Такая запись пройдет без ошибок:

```
airports.write \
    .mode("append") \
    .json("sample_data/airport-codes-txt")
```

При попытке чтения данных с помощью text мы получим все данные, тк форма json сохраняет все в виде JSON строк. Однако, если прочитать данные с помощью json, часть данных будут помечены как невалидные и помещены в колонку _corrupt_record

```
airports = spark.read.json("sample_data/airport-codes-txt")
airports.printSchema()
airports.show(n = 1, truncate = 100, vertical = True)
```

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

-RECORD 0-----------------------------------------
 _corrupt_record | null
 continent       | NA
 coordinates     | -74.93360137939453, 40.07080078125
 elevation_ft    | 11
 gps_code        | 00A
 iata_code       | null
 ident           | 00A
 iso_country     | US
 iso_region      | US-PA
 local_code      | 00A
 municipality    | Bensalem
 name            | Total Rf Heliport
 type            | heliport
only showing top 1 row
```

Отобразим невалидные JSON строки:

```
# Начиная со Spark 2.3 нельзя выбирать одну колонку _corrupt_record, поэтому мы добавим к выводу ident
```

```
airports.na.drop("all", None,
["_corrupt_record"]).select(airports._corrupt_record,
airports.continent).show(20, False)
```

```
+--------------------------------------------------+---------+
|_corrupt_record                                   |continent|
+--------------------------------------------------+---------+
|00A,heliport,Total Rf Heliport,NA                 |null     |
|00AA,small_airport,Aero B Ranch Airport,NA        |null     |
|00AK,small_airport,Lowell Field,NA                |null     |
|00AL,small_airport,Epps Airpark,NA                |null     |
|00AR,closed,Newport Hospital & Clinic Heliport,NA |null     |
|00AS,small_airport,Fulton Airport,NA              |null     |
|00AZ,small_airport,Cordes Airport,NA              |null     |
|00CA,small_airport,Goldstone /Gts/ Airport,NA     |null     |
|00CL,small_airport,Williams Ag Airport,NA         |null     |
|00CN,heliport,Kitchen Creek Helibase Heliport,NA  |null     |
|00CO,closed,Cass Field,NA                         |null     |
|00FA,small_airport,Grass Patch Airport,NA         |null     |
|00FD,heliport,Ringhaver Heliport,NA               |null     |
|00FL,small_airport,River Oak Airport,NA           |null     |
|00GA,small_airport,Lt World Airport,NA            |null     |
|00GE,heliport,Caffrey Heliport,NA                 |null     |
|00HI,heliport,Kaupulehu Heliport,NA               |null     |
|00ID,small_airport,Delta Shores Airport,NA        |null     |
|00IG,small_airport,Goltl Airport,NA               |null     |
|00II,heliport,Bailey Generation Station Heliport,NA|null     |
+--------------------------------------------------+---------+
only showing top 20 rows
```

##Режимы записи Spark позволяет нам выбирать режим записи данных с помощью метода mode(). Данный метод принимает один из параметров:

- overwrite - перезаписывает всю директорию целиком (или партицию, если используется партиционирование)
- append - дописывает новые файлы к текущим
- ignore - не выполняет запись (no op режим)
- error или errorifexists - возвращает ошибку, если директория уже существует

##Семплирование Форматы csv и json позволяют автоматически выводить схему из данных. При этом по-умолчанию Spark прочитает все данные и составит подходящую схему. Однако, если мы работаем с большим датасетом, это может занять продолжительное время. Решить это можно с помощью опции samplingRatio:

```
airports = spark.read \
    .options(header=True, inferSchema=True, samplingRatio=0.1) \
    .csv("sample_data/airport-codes.csv")
airports.printSchema()
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)
```

```
airports = spark.read \
    .options(header=True, inferSchema=True, samplingRatio=1.0) \
    .csv("sample_data/airport-codes.csv")
airports.printSchema()
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)
```

##Выводы

- Spark позволяет работать с текстовыми файлами json, csv, text
- При чтении и записи поддерживаются кодеки сжатия данных, это создает дополнительные накладные расходы
- При записи данных в текстовые форматы Spark **не выполняет** валидацию схемы и формата
- При включенном выведении схемы из источника чтение из текстовых форматов происходит дольше

7.3. Parquet и ORC

В отличие от обычных текстовых форматов, ORC и Parquet изначально спроектированы под распределенные системы хранения и обработки. Они являются колоночными - в них есть колонки и схема, как в таблицах БД и бинарными - прочитать обычным текстовым редактором их не получится. Форматы имеют похожие показатели производительности и архитектуру, но Parquet используется чаще

##Преимущества

- наличие схемы данных
- блочная компрессия
- партиционирование

- для каждого блока для каждой колонки вычисляется max и min, что позволяет ускорять чтение

  ##Недостатки:
- нельзя дописывать/менять данные в существующих файлах
- нельзя писать несколько DF в один файл
- необходимо делать compaction

Подробнее о Parquet: Фёдор Лаврентьев, Moscow Spark #5: Как класть Parquet

По аналогии с текстовыми форматами, при записи, Spark создает директорию и пишет туда все непустые партиции. Обратите внимание на последовательность форматов записи - snappy.parquet вместо, скажем, json.gz. При использовании компрессии сам parquet файл не помещается в сжатый контейнер. Вместо этого, компрессии подлежат блоки с данными. Это полностью снимает ограничение, из-за которого чтение сжатых текстовых файлов происходит в 1 поток в 1 партицию.

```
airports.write.mode("overwrite").parquet("sample_data/airport-codes")
```

```
!ls sample_data/airport-codes -hlt
```

```
total 3.2M
-rw-r--r-- 1 root root    0 Dec  7 15:25 _SUCCESS
-rw-r--r-- 1 root root 2.7M Dec  7 15:25
part-00000-cdecf371-9255-462b-af20-7d9e3fa34b8d-c000.snappy.parquet
-rw-r--r-- 1 root root 536K Dec  7 15:25
part-00001-cdecf371-9255-462b-af20-7d9e3fa34b8d-c000.snappy.parquet
```

```
spark.read.parquet("sample_data/airport-codes").printSchema()
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)
```

##Schema evolution При работе с ORC/Parquet, часто возникает вопрос эволюции схемы - изменения структуры данных относительно первоначальных файлов. Создадим два DF с разными схемами и запишем их в одну директорию:

```
from pyspark.sql import Row
Apple=Row("color","size")
```

```
a1 = Apple("green",1)
data = [a1]
rdd=spark.sparkContext.parallelize(data)
rdd.toDF().write.mode("append").parquet("sample_data/apples")
```

```
PriceApple=Row("color","size", "price")

rdd=spark.sparkContext.parallelize([PriceApple("red",1,250)])
rdd.toDF().write.mode("append").parquet("sample_data/apples")
```

Несмотря на то, что файлы имеют разную схему, Spark корректно читает файлы, используя обобщенную схему:

```
df = spark.read.parquet("sample_data/apples")
df.show()
```

```
+-----+----+-----+
|color|size|price|
+-----+----+-----+
|  red|   1|  250|
|green|   1| null|
+-----+----+-----+
```

Однако, это работает только тогда, когда мы добавляем новые колонки к нашей схеме. Если мы запишем новый файл, изменив тип уже существующей колонки, мы получим ошибку:

```
AppleBase=Row("size")

rdd=spark.sparkContext.parallelize([AppleBase(3.0)])
rdd.toDF().write.mode("append").parquet("sample_data/apples")

df = spark.read.parquet("sample_data/apples")
df.show()
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-29-5f31badd0ebd> in <module>()
      1 df = spark.read.parquet("sample_data/apples")
----> 2 df.show()

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/sql/dataframe.py in
show(self, n, truncate, vertical)
    492
    493         if isinstance(truncate, bool) and truncate:
--> 494             print(self._jdf.showString(n, 20, vertical))
    495         else:
    496             try:

/content/spark-3.2.0-bin-hadoop2.7/python/lib/py4j-0.10.9.2-src.zip/py4j/java
_gateway.py in __call__(self, *args)
   1308         answer = self.gateway_client.send_command(command)
   1309         return_value = get_return_value(
-> 1310             answer, self.gateway_client, self.target_id, self.name)
   1311
   1312         for temp_arg in temp_args:

/content/spark-3.2.0-bin-hadoop2.7/python/pyspark/sql/utils.py in deco(*a,
**kw)
    109     def deco(*a, **kw):
    110         try:
```

```
--> 111             return f(*a, **kw)
    112         except py4j.protocol.Py4JJavaError as e:
    113             converted = convert_exception(e.java_exception)

/content/spark-3.2.0-bin-hadoop2.7/python/lib/py4j-0.10.9.2-src.zip/py4j/prot
ocol.py in get_return_value(answer, gateway_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling o324.showString.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 0
in stage 40.0 failed 1 times, most recent failure: Lost task 0.0 in stage
40.0 (TID 55) (3c91d9574429 executor driver):
org.apache.spark.sql.execution.QueryExecutionException: Parquet column cannot
be converted in file
file:///content/sample_data/apples/part-00001-e5cbcc27-776f-4460-871d-802e097
e1697-c000.snappy.parquet. Column: [size], Expected: double, Found: INT64
      at
org.apache.spark.sql.errors.QueryExecutionErrors$.unsupportedSchemaColumnConv
ertError(QueryExecutionErrors.scala:570)
      at
org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.nextIterator(F
ileScanRDD.scala:172)
      at
org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.hasNext(FileSc
anRDD.scala:93)
      at
org.apache.spark.sql.execution.FileSourceScanExec$$anon$1.hasNext(DataSourceS
canExec.scala:522)
      at
org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIteratorFor
CodegenStage1.columnartorow_nextBatch_0$(Unknown Source)
      at
org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIteratorFor
CodegenStage1.processNext(Unknown Source)
      at
org.apache.spark.sql.execution.BufferedRowIterator.hasNext(BufferedRowIterato
r.java:43)
      at
org.apache.spark.sql.execution.WholeStageCodegenExec$$anon$1.hasNext(WholeSta
geCodegenExec.scala:759)
      at
org.apache.spark.sql.execution.SparkPlan.$anonfun$getByteArrayRdd$1(SparkPlan
.scala:349)
      at
org.apache.spark.rdd.RDD.$anonfun$mapPartitionsInternal$2(RDD.scala:898)
      at
org.apache.spark.rdd.RDD.$anonfun$mapPartitionsInternal$2$adapted(RDD.scala:8
98)
      at
org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
      at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:373)
```

```
	at org.apache.spark.rdd.RDD.iterator(RDD.scala:337)
	at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
	at org.apache.spark.scheduler.Task.run(Task.scala:131)
	at
org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:5
06)
	at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1462)
	at
org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:509)
	at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:114
9)
	at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:62
4)
	at java.lang.Thread.run(Thread.java:748)
Caused by:
org.apache.spark.sql.execution.datasources.SchemaColumnConvertNotSupportedExc
eption
	at
org.apache.spark.sql.execution.datasources.parquet.ParquetVectorUpdaterFactor
y.constructConvertNotSupportedException(ParquetVectorUpdaterFactory.java:1077
)
	at
org.apache.spark.sql.execution.datasources.parquet.ParquetVectorUpdaterFactor
y.getUpdater(ParquetVectorUpdaterFactory.java:172)
	at
org.apache.spark.sql.execution.datasources.parquet.VectorizedColumnReader.rea
dBatch(VectorizedColumnReader.java:154)
	at
org.apache.spark.sql.execution.datasources.parquet.VectorizedParquetRecordRea
der.nextBatch(VectorizedParquetRecordReader.java:283)
	at
org.apache.spark.sql.execution.datasources.parquet.VectorizedParquetRecordRea
der.nextKeyValue(VectorizedParquetRecordReader.java:181)
	at
org.apache.spark.sql.execution.datasources.RecordReaderIterator.hasNext(Recor
dReaderIterator.scala:39)
	at
org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.hasNext(FileSc
anRDD.scala:93)
	at
org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.nextIterator(F
ileScanRDD.scala:168)
	... 20 more

Driver stacktrace:
	at
org.apache.spark.scheduler.DAGScheduler.failJobAndIndependentStages(DAGSchedu
ler.scala:2403)
	at
org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2(DAGScheduler.sc
ala:2352)
	at
org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2$adapted(DAGSche
```

```
duler.scala:2351)
      at
scala.collection.mutable.ResizableArray.foreach(ResizableArray.scala:62)
      at
scala.collection.mutable.ResizableArray.foreach$(ResizableArray.scala:55)
      at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffer.scala:49)
      at
org.apache.spark.scheduler.DAGScheduler.abortStage(DAGScheduler.scala:2351)
      at
org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1(DAGSch
eduler.scala:1109)
      at
org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1$adapte
d(DAGScheduler.scala:1109)
      at scala.Option.foreach(Option.scala:407)
      at
org.apache.spark.scheduler.DAGScheduler.handleTaskSetFailed(DAGScheduler.scal
a:1109)
      at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.doOnReceive(DAGSchedu
ler.scala:2591)
      at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGSchedule
r.scala:2533)
      at
org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGSchedule
r.scala:2522)
      at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:49)
      at
org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:898)
      at org.apache.spark.SparkContext.runJob(SparkContext.scala:2214)
      at org.apache.spark.SparkContext.runJob(SparkContext.scala:2235)
      at org.apache.spark.SparkContext.runJob(SparkContext.scala:2254)
      at
org.apache.spark.sql.execution.SparkPlan.executeTake(SparkPlan.scala:476)
      at
org.apache.spark.sql.execution.SparkPlan.executeTake(SparkPlan.scala:429)
      at
org.apache.spark.sql.execution.CollectLimitExec.executeCollect(limit.scala:48
)
      at org.apache.spark.sql.Dataset.collectFromPlan(Dataset.scala:3715)
      at org.apache.spark.sql.Dataset.$anonfun$head$1(Dataset.scala:2728)
      at
org.apache.spark.sql.Dataset.$anonfun$withAction$1(Dataset.scala:3706)
      at
org.apache.spark.sql.execution.SQLExecution$.$anonfun$withNewExecutionId$5(SQ
LExecution.scala:103)
      at
org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQLExecuti
on.scala:163)
      at
org.apache.spark.sql.execution.SQLExecution$.$anonfun$withNewExecutionId$1(SQ
LExecution.scala:90)
      at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:775)
      at
```

```
org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.
scala:64)
      at org.apache.spark.sql.Dataset.withAction(Dataset.scala:3704)
      at org.apache.spark.sql.Dataset.head(Dataset.scala:2728)
      at org.apache.spark.sql.Dataset.take(Dataset.scala:2935)
      at org.apache.spark.sql.Dataset.getRows(Dataset.scala:287)
      at org.apache.spark.sql.Dataset.showString(Dataset.scala:326)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
      at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
java:43)
      at java.lang.reflect.Method.invoke(Method.java:498)
      at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
      at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
      at py4j.Gateway.invoke(Gateway.java:282)
      at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
      at py4j.commands.CallCommand.execute(CallCommand.java:79)
      at
py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:182)
      at py4j.ClientServerConnection.run(ClientServerConnection.java:106)
      at java.lang.Thread.run(Thread.java:748)
Caused by: org.apache.spark.sql.execution.QueryExecutionException: Parquet
column cannot be converted in file
file:///content/sample_data/apples/part-00001-e5cbcc27-776f-4460-871d-802e097
e1697-c000.snappy.parquet. Column: [size], Expected: double, Found: INT64
      at
org.apache.spark.sql.errors.QueryExecutionErrors$.unsupportedSchemaColumnConv
ertError(QueryExecutionErrors.scala:570)
      at
org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.nextIterator(F
ileScanRDD.scala:172)
      at
org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.hasNext(FileSc
anRDD.scala:93)
      at
org.apache.spark.sql.execution.FileSourceScanExec$$anon$1.hasNext(DataSourceS
canExec.scala:522)
      at
org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIteratorFor
CodegenStage1.columnartorow_nextBatch_0$(Unknown Source)
      at
org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIteratorFor
CodegenStage1.processNext(Unknown Source)
      at
org.apache.spark.sql.execution.BufferedRowIterator.hasNext(BufferedRowIterato
r.java:43)
      at
org.apache.spark.sql.execution.WholeStageCodegenExec$$anon$1.hasNext(WholeSta
geCodegenExec.scala:759)
      at
org.apache.spark.sql.execution.SparkPlan.$anonfun$getByteArrayRdd$1(SparkPlan
.scala:349)
      at
```

```
org.apache.spark.rdd.RDD.$anonfun$mapPartitionsInternal$2(RDD.scala:898)
      at
org.apache.spark.rdd.RDD.$anonfun$mapPartitionsInternal$2$adapted(RDD.scala:8
98)
      at
org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
      at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:373)
      at org.apache.spark.rdd.RDD.iterator(RDD.scala:337)
      at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
      at org.apache.spark.scheduler.Task.run(Task.scala:131)
      at
org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:5
06)
      at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1462)
      at
org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:509)
      at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:114
9)
      at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:62
4)
      ... 1 more
Caused by:
org.apache.spark.sql.execution.datasources.SchemaColumnConvertNotSupportedExc
eption
      at
org.apache.spark.sql.execution.datasources.parquet.ParquetVectorUpdaterFactor
y.constructConvertNotSupportedException(ParquetVectorUpdaterFactory.java:1077
)
      at
org.apache.spark.sql.execution.datasources.parquet.ParquetVectorUpdaterFactor
y.getUpdater(ParquetVectorUpdaterFactory.java:172)
      at
org.apache.spark.sql.execution.datasources.parquet.VectorizedColumnReader.rea
dBatch(VectorizedColumnReader.java:154)
      at
org.apache.spark.sql.execution.datasources.parquet.VectorizedParquetRecordRea
der.nextBatch(VectorizedParquetRecordReader.java:283)
      at
org.apache.spark.sql.execution.datasources.parquet.VectorizedParquetRecordRea
der.nextKeyValue(VectorizedParquetRecordReader.java:181)
      at
org.apache.spark.sql.execution.datasources.RecordReaderIterator.hasNext(Recor
dReaderIterator.scala:39)
      at
org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.hasNext(FileSc
anRDD.scala:93)
      at
org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.nextIterator(F
ileScanRDD.scala:168)
      ... 20 more
```

##Parquet Tools Для диагностики и решения проблем, связанных с parquet, можно использовать утилиту parquet-tools

Она позволяет:

- получить схему файла
- вывести содержимое файла в консоль
- объединить несколько файлов в один

https://github.com/apache/parquet-mr/tree/master/parquet-tools

##Сравнение скорости обработки запросов Подготовим датасеты:

```
airports = spark.read.csv("sample_data/airport-codes.csv", header=True,
inferSchema=True)
airports.printSchema()
airports
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)
```

```
DataFrame[ident: string, type: string, name: string, elevation_ft: int,
continent: string, iso_country: string, iso_region: string, municipality:
string, gps_code: string, iata_code: string, local_code: string, coordinates:
string]
```

```
for i in range(0, 100):

airports.repartition(1).write.mode("append").parquet("sample_data/a1-parquet"
)
    airports.repartition(1).write.mode("append").json("sample_data/a1-json")
    airports.repartition(1).write.mode("append").orc("sample_data/a1-orc")
```

Создадим обертку для замеров времени

```
from time import time
from datetime import timedelta

class T():
    def __enter__(self):
        self.start = time()
    def __exit__(self, type, value, traceback):
        self.end = time()
```

```
        elapsed = self.end - self.start
        print(str(timedelta(seconds=elapsed)))
```

Сравним скорость работы фильтраци:

```
with T():
  spark.read.json("sample_data/a1-json").filter("iso_country = 'RU' and
elevation_ft > 300").count()
```

```
0:00:29.218975
```

```
with T():
  spark.read.orc("sample_data/a1-orc").filter("iso_country = 'RU' and
elevation_ft > 300").count()
```

```
0:00:02.018653
```

```
with T():
  spark.read.parquet("sample_data/a1-parquet").filter("iso_country = 'RU' and
elevation_ft > 300").count()
```

```
0:00:01.708915
```

Сравним скорость подсчета количества строк:

```
with T():
  spark.read.json("sample_data/a1-json").count()
```

```
0:00:33.340543
```

```
with T():
  spark.read.orc("sample_data/a1-orc").count()
```

```
0:00:00.442091
```

```
with T():
  spark.read.parquet("sample_data/a1-parquet").count()
```

```
0:00:00.443877
```

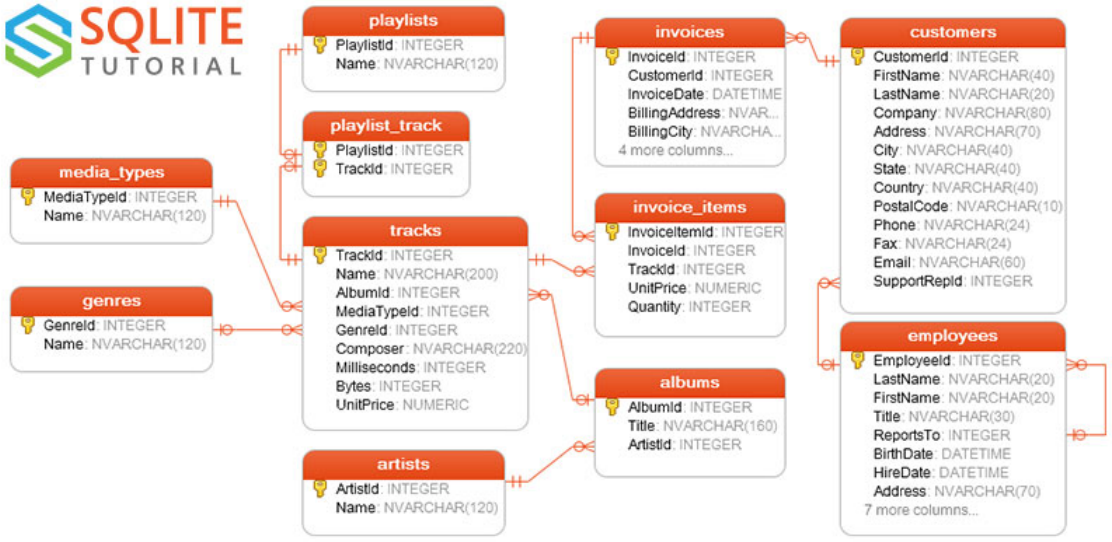
##Выводы:

- Форматы orc и parquet позволяют эффективно работать со структурированными данными
- Производительность orc и parquet на порядок выше обычных текстовых файлов
- Данные форматы поддерживают сжатие на блочном уровне, что позволяет избегать проблем с многопоточным чтением
- Форматы поддерживают добавление новых колонок в схему, но не изменение текущих

7.4. DataBases

Как было описано в самом начале БД могут быть очень разными (реляционными, документоориентированными, NoSQL). Для использования обычных РСУБД можно использовать классический jdbc драйвер. Рассмотрим настройку на примере двух БД:

1. SQLite - компактная встраиваемая система управления базами данных (СУБД).
2. PostgreSQL - свободная объектно-реляционная система управления базами данных (СУБД).



Демонстрационную БД скачаем с сайта в виде файла: https://www.sqlitetutorial.net/wp-content/uploads/2018/03/sqlite-sample-database-diagram.pdf

```
!wget https://www.sqlitetutorial.net/wp-content/uploads/2018/03/chinook.zip
-O 'sample_data/chinook.zip' -q
!unzip "sample_data/chinook.zip" -d "sample_data/"
```

```
Archive:  sample_data/chinook.zip
  inflating: sample_data/chinook.db
```

Для подключения воспользуемся мощным инструментом SQLAlchemy позволяющим подключаться к различным БД из Python.

```
from sqlalchemy import create_engine

engine = create_engine("sqlite:///sample_data/chinook.db") #движок для
подключений

with engine.connect() as conn, conn.begin(): #контекст соединения
  cursor = conn.execute('''SELECT * FROM tracks;''')
  i = 10;
  for row in cursor:
    i = i - 1;
    if i > 0:
      print(row)
    else:
      break
```

```
(1, 'For Those About To Rock (We Salute You)', 1, 1, 1, 'Angus Young, Malcolm
Young, Brian Johnson', 343719, 11170334, 0.99)
(2, 'Balls to the Wall', 2, 2, 1, None, 342562, 5510424, 0.99)
(3, 'Fast As a Shark', 3, 2, 1, 'F. Baltes, S. Kaufman, U. Dirkscneider & W.
Hoffman', 230619, 3990994, 0.99)
```

```
(4, 'Restless and Wild', 3, 2, 1, 'F. Baltes, R.A. Smith-Diesel, S. Kaufman,
U. Dirkscneider & W. Hoffman', 252051, 4331779, 0.99)
(5, 'Princess of the Dawn', 3, 2, 1, 'Deaffy & R.A. Smith-Diesel', 375418,
6290521, 0.99)
(6, 'Put The Finger On You', 1, 1, 1, 'Angus Young, Malcolm Young, Brian
Johnson', 205662, 6713451, 0.99)
(7, "Let's Get It Up", 1, 1, 1, 'Angus Young, Malcolm Young, Brian Johnson',
233926, 7636561, 0.99)
(8, 'Inject The Venom', 1, 1, 1, 'Angus Young, Malcolm Young, Brian Johnson',
210834, 6852860, 0.99)
(9, 'Snowballed', 1, 1, 1, 'Angus Young, Malcolm Young, Brian Johnson',
203102, 6599424, 0.99)
```

Для того, что бы читать из БД, выберем формат jdbc укажем соответствующий драйвер. И воспользуемся методом `load`:

```
jdbcDriver = 'org.sqlite.JDBC'
jdbcUrl = 'jdbc:sqlite:sample_data/chinook.db'
df = spark.read.format('jdbc') \
        .options(driver=jdbcDriver, dbtable='tracks', url=jdbcUrl)\
        .load()
df.show()
```

```
+-------+--------------------+-------+----------+-------+------------------
-+-----------+--------+---------+
|TrackId|                Name|AlbumId|MediaTypeId|GenreId|
Composer|Milliseconds|   Bytes|UnitPrice|
+-------+--------------------+-------+----------+-------+------------------
-+-----------+--------+---------+
|      1|For Those About T...|      1|         1|      1|Angus Young,
Malc...|      343719|11170334|     0.99|
|      2|   Balls to the Wall|      2|         2|      1|
null|      342562| 5510424|     0.99|
|      3|     Fast As a Shark|      3|         2|      1|F. Baltes, S.
Kau...|      230619| 3990994|     0.99|
|      4|   Restless and Wild|      3|         2|      1|F. Baltes, R.A.
S...|      252051| 4331779|     0.99|
|      5|Princess of the Dawn|      3|         2|      1|Deaffy & R.A.
Smi...|      375418| 6290521|     0.99|
|      6|Put The Finger On...|      1|         1|      1|Angus Young,
Malc...|      205662| 6713451|     0.99|
|      7|     Let's Get It Up|      1|         1|      1|Angus Young,
Malc...|      233926| 7636561|     0.99|
|      8|    Inject The Venom|      1|         1|      1|Angus Young,
Malc...|      210834| 6852860|     0.99|
|      9|          Snowballed|      1|         1|      1|Angus Young,
Malc...|      203102| 6599424|     0.99|
|     10|          Evil Walks|      1|         1|      1|Angus Young,
Malc...|      263497| 8611245|     0.99|
|     11|              C.O.D.|      1|         1|      1|Angus Young,
Malc...|      199836| 6566314|     0.99|
|     12|  Breaking The Rules|      1|         1|      1|Angus Young,
Malc...|      263288| 8596840|     0.99|
|     13|Night Of The Long...|      1|         1|      1|Angus Young,
Malc...|      205688| 6706347|     0.99|
|     14|          Spellbound|      1|         1|      1|Angus Young,
```

```
Malc...|      270863| 8817038|     0.99|
|     15|             Go Down|      4|          1|      1|
AC/DC|      331180|10847611|     0.99|
|     16|          Dog Eat Dog|      4|          1|      1|
AC/DC|      215196| 7032162|     0.99|
|     17|   Let There Be Rock|      4|          1|      1|
AC/DC|      366654|12021261|     0.99|
|     18|      Bad Boy Boogie|      4|          1|      1|
AC/DC|      267728| 8776140|     0.99|
|     19|       Problem Child|      4|          1|      1|
AC/DC|      325041|10617116|     0.99|
|     20|            Overdose|      4|          1|      1|
AC/DC|      369319|12066294|     0.99|
+-------+--------------------+-------+-----------+-------+------------------
-+------------+--------+---------+
only showing top 20 rows
```

Опция `dbtable` указывает имя таблицы. Но кроме того можно передать туда именованный запрос. Таким образом будет производиться **predicate pushdown** на уровень БД.

```
df = spark.read \
    .format("jdbc") \
    .options(url=jdbcUrl, dbtable="(select * from tracks join albums using
(albumid) join artists using (artistid)) tmp",driver=jdbcDriver) \
    .load()
df.show()
```

```
+-------+--------------------+-------+-----------+-------+------------------
-+------------+--------+---------+--------------------+--------+------+
|TrackId|                Name|AlbumId|MediaTypeId|GenreId|
Composer|Milliseconds|   Bytes|UnitPrice|
Title|ArtistId|Name:1|
+-------+--------------------+-------+-----------+-------+------------------
-+------------+--------+---------+--------------------+--------+------+
|      1|For Those About T...|      1|          1|      1|Angus Young,
Malc...|      343719|11170334|     0.99|For Those About T...|       1| AC/DC|
|      6|Put The Finger On...|      1|          1|      1|Angus Young,
Malc...|      205662| 6713451|     0.99|For Those About T...|       1| AC/DC|
|      7|     Let's Get It Up|      1|          1|      1|Angus Young,
Malc...|      233926| 7636561|     0.99|For Those About T...|       1| AC/DC|
|      8|    Inject The Venom|      1|          1|      1|Angus Young,
Malc...|      210834| 6852860|     0.99|For Those About T...|       1| AC/DC|
|      9|          Snowballed|      1|          1|      1|Angus Young,
Malc...|      203102| 6599424|     0.99|For Those About T...|       1| AC/DC|
|     10|          Evil Walks|      1|          1|      1|Angus Young,
Malc...|      263497| 8611245|     0.99|For Those About T...|       1| AC/DC|
|     11|              C.O.D.|      1|          1|      1|Angus Young,
Malc...|      199836| 6566314|     0.99|For Those About T...|       1| AC/DC|
|     12|  Breaking The Rules|      1|          1|      1|Angus Young,
Malc...|      263288| 8596840|     0.99|For Those About T...|       1| AC/DC|
|     13|Night Of The Long...|      1|          1|      1|Angus Young,
Malc...|      205688| 6706347|     0.99|For Those About T...|       1| AC/DC|
|     14|          Spellbound|      1|          1|      1|Angus Young,
Malc...|      270863| 8817038|     0.99|For Those About T...|       1| AC/DC|
```

```
|      2|   Balls to the Wall|      2|          2|      1|
null|      342562| 5510424|     0.99|   Balls to the Wall|       2|Accept|
|      3|     Fast As a Shark|      3|          2|      1|F. Baltes, S.
Kau...|      230619| 3990994|     0.99|   Restless and Wild|       2|Accept|
|      4|   Restless and Wild|      3|          2|      1|F. Baltes, R.A.
S...|      252051| 4331779|     0.99|   Restless and Wild|       2|Accept|
|      5|Princess of the Dawn|      3|          2|      1|Deaffy & R.A.
Smi...|      375418| 6290521|     0.99|   Restless and Wild|       2|Accept|
|     15|             Go Down|      4|          1|      1|
AC/DC|      331180|10847611|     0.99|   Let There Be Rock|       1| AC/DC|
|     16|          Dog Eat Dog|      4|          1|      1|
AC/DC|      215196| 7032162|     0.99|   Let There Be Rock|       1| AC/DC|
|     17|   Let There Be Rock|      4|          1|      1|
AC/DC|      366654|12021261|     0.99|   Let There Be Rock|       1| AC/DC|
|     18|      Bad Boy Boogie|      4|          1|      1|
AC/DC|      267728| 8776140|     0.99|   Let There Be Rock|       1| AC/DC|
|     19|       Problem Child|      4|          1|      1|
AC/DC|      325041|10617116|     0.99|   Let There Be Rock|       1| AC/DC|
|     20|            Overdose|      4|          1|      1|
AC/DC|      369319|12066294|     0.99|   Let There Be Rock|       1| AC/DC|
+-------+--------------------+-------+-----------+-------+------------------
-+------------+--------+---------+--------------------+--------+------+
only showing top 20 rows
```

На встраиваемой БД показать возможности записи проблематично, поскольку одновременно много соединений не поддерживается. Поэтому установим и настроим PostgreSQL.

```
!apt-get update &>log0 && apt install postgresql postgresql-contrib &>log1
!service postgresql start
!sudo -u postgres psql -c 'CREATE DATABASE mydatabase'
!sudo -u postgres psql -c "CREATE USER me WITH ENCRYPTED PASSWORD 'mypass'"
!sudo -u postgres psql -c 'CREATE SCHEMA AUTHORIZATION "me";'
!sudo -u postgres psql -c 'GRANT ALL PRIVILEGES ON DATABASE mydatabase TO me'
```

```
 * Starting PostgreSQL 10 database server
   ...done.
CREATE DATABASE
CREATE ROLE
CREATE SCHEMA
GRANT
```

Проверим, что БД есть и мы можем к ней подключаться:

```
from sqlalchemy import create_engine

engine = create_engine("postgresql://me:mypass@localhost/mydatabase")

with engine.connect() as conn, conn.begin():
  cursor = conn.execute('''SELECT * FROM pg_catalog.pg_tables LIMIT 10;''')
  for row in cursor:
    print(row)
  conn.close()
```

```
('pg_catalog', 'pg_statistic', 'postgres', None, True, False, False, False)
('pg_catalog', 'pg_type', 'postgres', None, True, False, False, False)
('pg_catalog', 'pg_policy', 'postgres', None, True, False, False, False)
('pg_catalog', 'pg_authid', 'postgres', 'pg_global', True, False, False,
False)
('pg_catalog', 'pg_user_mapping', 'postgres', None, True, False, False,
False)
('pg_catalog', 'pg_subscription', 'postgres', 'pg_global', True, False,
False, False)
('pg_catalog', 'pg_attribute', 'postgres', None, True, False, False, False)
('pg_catalog', 'pg_proc', 'postgres', None, True, False, False, False)
('pg_catalog', 'pg_class', 'postgres', None, True, False, False, False)
('pg_catalog', 'pg_attrdef', 'postgres', None, True, False, False, False)

/usr/local/lib/python3.7/dist-packages/psycopg2/__init__.py:144: UserWarning:
The psycopg2 wheel package will be renamed from release 2.8; in order to keep
installing from binary please use "pip install psycopg2-binary" instead. For
details see:
<http://initd.org/psycopg/docs/install.html#binary-install-from-pypi>.
  """)
```

Создадим таблицу способную вместить наш кадр данных airport-codes

```
airports = spark.read.csv("sample_data/airport-codes.csv", header=True,
inferSchema=True)
airports.printSchema()
typesMap = {"string": "VARCHAR (100)", "int": "INTEGER"}
primaryKey = "ident"
print(airports.schema.fields)
print("")
ddlColumns = map(lambda x : \
                 x.name + " " + typesMap[x.dataType.simpleString()] + "
PRIMARY KEY" \
                 if (x.name == primaryKey) \
                 else x.name + " " + typesMap[x.dataType.simpleString()],
airports.schema.fields)
dropQuery = '\nDROP TABLE IF EXISTS airports;\n'
print(dropQuery)
ddlQuery = "CREATE TABLE IF NOT EXISTS airports (\n" + ",\n".join(ddlColumns)
+ ");"
print(ddlQuery)

with engine.connect() as conn, conn.begin():
  cursor = conn.execute(dropQuery)
  cursor = conn.execute(ddlQuery)
  conn.close()

root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)

[StructField(ident,StringType,true), StructField(type,StringType,true),
StructField(name,StringType,true),
StructField(elevation_ft,IntegerType,true),
StructField(continent,StringType,true),
StructField(iso_country,StringType,true),
StructField(iso_region,StringType,true),
StructField(municipality,StringType,true),
StructField(gps_code,StringType,true),
StructField(iata_code,StringType,true),
StructField(local_code,StringType,true),
StructField(coordinates,StringType,true)]


DROP TABLE IF EXISTS airports;

CREATE TABLE IF NOT EXISTS airports (
ident VARCHAR (100) PRIMARY KEY,
type VARCHAR (100),
name VARCHAR (100),
elevation_ft INTEGER,
continent VARCHAR (100),
iso_country VARCHAR (100),
iso_region VARCHAR (100),
municipality VARCHAR (100),
gps_code VARCHAR (100),
iata_code VARCHAR (100),
local_code VARCHAR (100),
coordinates VARCHAR (100));
```

Проверим, что чтение БД возможно из Spark.

```
jdbcUrl = 'jdbc:postgresql://localhost/mydatabase?user=me&password=mypass'
jdbcDriver = 'org.postgresql.Driver'
df = spark.read.format('jdbc') \
        .options(driver=jdbcDriver, dbtable='pg_catalog.pg_user',
url=jdbcUrl)\
        .load()
df.show()

+--------+--------+-----------+--------+-------+------------+--------+-------
-+---------+
| usename|usesysid|usecreatedb|usesuper|userepl|usebypassrls|
passwd|valuntil|useconfig|
+--------+--------+-----------+--------+-------+------------+--------+-------
-+---------+
|postgres|      10|       true|    true|   true|        true|********|
null|     null|
|      me|   16385|      false|   false|  false|       false|********|
null|     null|
+--------+--------+-----------+--------+-------+------------+--------+-------
```

```
-+---------+
```

Запишем данные в БД:

```
airports.write.format("jdbc").option("url",
jdbcUrl).option("driver",jdbcDriver).option("dbtable",
"airports").mode("overwrite").save()
```

И прочитаем уже оттуда.

```
df = spark.read.format("jdbc").option("url", jdbcUrl).option("dbtable",
"airports").option("driver",jdbcDriver).load()

df.printSchema()
df.show(2, 200, True)
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)

-RECORD 0-----------------------------------------
 ident        | 00A
 type         | heliport
 name         | Total Rf Heliport
 elevation_ft | 11
 continent    | NA
 iso_country  | US
 iso_region   | US-PA
 municipality | Bensalem
 gps_code     | 00A
 iata_code    | null
 local_code   | 00A
 coordinates  | -74.93360137939453, 40.07080078125
-RECORD 1-----------------------------------------
 ident        | 00AA
 type         | small_airport
 name         | Aero B Ranch Airport
 elevation_ft | 3435
 continent    | NA
 iso_country  | US
 iso_region   | US-KS
 municipality | Leoti
 gps_code     | 00AA
 iata_code    | null
 local_code   | 00AA
```

```
 coordinates  | -101.473911, 38.704022
only showing top 2 rows
```

При использовании с параметром по умолчанию мы получаем всего 1 партицию в DF:

```
df.rdd.getNumPartitions()
```

```
1
```

Исправить это можно, используя параметры `partitionColumn`, `lowerBound`, `upperBound`, `numPartitions`. Для этого нам понадобится добавить новую колонку в нашу таблицу:

```
ddlColumns = list(map(lambda x : \
                  x.name + " " + typesMap[x.dataType.simpleString()] + "
PRIMARY KEY" \
                  if (x.name == primaryKey) \
                  else x.name + " " + typesMap[x.dataType.simpleString()],
airports.schema.fields))
ddlColumns.append('id INTEGER') # добавляем новую колонку
print(ddlColumns)
dropQuery = '\nDROP TABLE IF EXISTS airports;\n'
print(dropQuery)
ddlQuery = "CREATE TABLE IF NOT EXISTS airports (\n" + ",\n".join(ddlColumns)
+ ");"
print(ddlQuery)

with engine.connect() as conn, conn.begin():
  cursor = conn.execute(dropQuery)
  cursor = conn.execute(ddlQuery)
  conn.close()
```

```
['ident VARCHAR (100) PRIMARY KEY', 'type VARCHAR (100)', 'name VARCHAR
(100)', 'elevation_ft INTEGER', 'continent VARCHAR (100)', 'iso_country
VARCHAR (100)', 'iso_region VARCHAR (100)', 'municipality VARCHAR (100)',
'gps_code VARCHAR (100)', 'iata_code VARCHAR (100)', 'local_code VARCHAR
(100)', 'coordinates VARCHAR (100)', 'id INTEGER']

DROP TABLE IF EXISTS airports;

CREATE TABLE IF NOT EXISTS airports (
ident VARCHAR (100) PRIMARY KEY,
type VARCHAR (100),
name VARCHAR (100),
elevation_ft INTEGER,
continent VARCHAR (100),
iso_country VARCHAR (100),
iso_region VARCHAR (100),
municipality VARCHAR (100),
gps_code VARCHAR (100),
iata_code VARCHAR (100),
local_code VARCHAR (100),
coordinates VARCHAR (100),
id INTEGER);
```

Перезапишем данные в новую таблицу:

```
from pyspark.sql.functions import rand, when, ceil

airId = airports.withColumn("id", ceil(rand()*100))
airId.write \
    .format("jdbc") \
    .option("driver",jdbcDriver) \
    .option("url", jdbcUrl) \
    .option("dbtable", "airports") \
    .mode("overwrite").save()
```

Прочитаем таблицу, установив дополнительные параметры `partitionColumn`, `lowerBound`, `upperBound`, `numPartitions`:

```
df = spark.read \
    .format("jdbc") \
    .option("url", jdbcUrl) \
    .option("dbtable", "airports") \
    .option("driver",jdbcDriver) \
    .option("partitionColumn", "id") \
    .option("lowerBound", "0") \
    .option("upperBound", "100") \
    .option("numPartitions", "100") \
    .load()

df.printSchema()
df.show(2, 200, True)
```

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: integer (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)
 |-- id: long (nullable = true)

-RECORD 0-----------------------------------------------
 ident        | SSNV
 type         | small_airport
 name         | Fazenda Novo Hamburgo Airport
 elevation_ft | 285
 continent    | SA
 iso_country  | BR
 iso_region   | BR-MS
 municipality | CorumbÃ¡
 gps_code     | SSNV
 iata_code    | null
 local_code   | null
```

```
 coordinates  | -57.029720306396484, -19.233055114746094
 id           | 1
-RECORD 1-----------------------------------------------
 ident        | SSRD
 type         | small_airport
 name         | Fazenda Sol Nascente Airport
 elevation_ft | 1443
 continent    | SA
 iso_country  | BR
 iso_region   | BR-MS
 municipality | Tacuru
 gps_code     | SSRD
 iata_code    | null
 local_code   | null
 coordinates  | -55.24833297729492, -23.397499084472656
 id           | 1
only showing top 2 rows
```

Проверим, сколько партиций получилось:

```
df.rdd.getNumPartitions()
```

```
100
```

Проверим распределение данных по партициям:

```
from pyspark.sql.functions import spark_partition_id, asc

df.withColumn("partitionId",
spark_partition_id()).groupBy("partitionId").count().orderBy(asc("count")).sh
ow()
```

```
+-----------+-----+
|partitionId|count|
+-----------+-----+
|         95|  508|
|         23|  511|
|         79|  522|
|         51|  527|
|         21|  530|
|         41|  530|
|         15|  536|
|         14|  537|
|         27|  538|
|         54|  538|
|         62|  538|
|         32|  541|
|         24|  542|
|         48|  543|
|         49|  544|
|         30|  546|
|         96|  546|
|          4|  549|
|         70|  549|
|         57|  550|
+-----------+-----+
```

```
only showing top 20 rows
```

##Выводы:

- Spark позволяет работать с различными СУБД через JDBC коннектор
- При использовании `jdbc` настройка партиционирования задается вручную

8. Работа с SQL в Apache Spark

Рассматриваем, как использовать SQL в DataFrame API Apache Spark на примере анализа информации о героях комиксов.

8.1. Создание DataFrame в Spark

Обогатим данные из источников дополнительным атрибутом `universe` и создадим в кэше Spark сессии представление, для работы с кадром данных как таблицей с помощью метода `createOrReplaceTempView`. Обратите внимание, что такие представления не разделяется между сессиями spark.

```
from pyspark.sql.functions import lit, col, when

marvel_df = spark.read.csv("marvel-wikia-data.csv",  # Путь к файлу с данными
                           header =True,             # Есть ли в файле заголовки
                           inferSchema=True)         # Нужно ли автоматически определять схему данных
marvel_df = marvel_df.withColumn('universe', lit('Marvel'))
marvel_df.createOrReplaceTempView("marvel_superheroes")
dc_df = spark.read.csv("dc-wikia-data.csv",  # Путь к файлу с данными
                       header =True,         # Есть ли в файле заголовки
                       inferSchema=True)     # Нужно ли автоматически определять схему данных
dc_df = dc_df.withColumn('universe', lit('DC'))
dc_df.createOrReplaceTempView("dc_superheroes")
```

Теперь к кадрам данных можно обращаться через язык SQL. Это значительно облегчает читабельность для бизнес-аналитиков, знающих этот язык и хорошо погружённых в предметные области, которые традиционно обрабатываются в СУБД (банкинг, биллинг, учет и планирование ресурсов предприятия, системы управления взаимоотношениями с клиентами).

```
spark.sql("SELECT * FROM marvel_superheroes UNION ALL SELECT * FROM dc_superheroes").show()
```

```
+-------+--------------------+--------------------+---------------+---------
---------+----------+----------+-----------------+----+----------------+----
-------+---------------+----+--------+
|page_id|                name|             urlslug|             ID|
ALIGN|       EYE|      HAIR|              SEX| GSM|
ALIVE|APPEARANCES|FIRST APPEARANCE|Year|universe|
```

```
+-------+--------------------+--------------------+---------------+---------
---------+----------+----------+-----------------+----+-----------------+----
-------+---------------+----+--------+
|   1678|Spider-Man (Peter...|\/Spider-Man_(Pet...| Secret Identity|   Good
Characters|Hazel Eyes|Brown Hair|  Male Characters|null|Living Characters|
4043|          Aug-62|1962|  Marvel|
|   7139|Captain America (...|\/Captain_America...| Public Identity|   Good
Characters| Blue Eyes|White Hair|  Male Characters|null|Living Characters|
3360|          Mar-41|1941|  Marvel|
|  64786|"Wolverine (James...|\/Wolverine_(Jame...| Public Identity|Neutral
Characters| Blue Eyes|Black Hair|  Male Characters|null|Living Characters|
3061|          Oct-74|1974|  Marvel|
|   1868|"Iron Man (Anthon...|\/Iron_Man_(Antho...| Public Identity|   Good
Characters| Blue Eyes|Black Hair|  Male Characters|null|Living Characters|
2961|          Mar-63|1963|  Marvel|
|   2460| Thor (Thor Odinson)|\/Thor_(Thor_Odin...|No Dual Identity|   Good
Characters| Blue Eyes|Blond Hair|  Male Characters|null|Living Characters|
2258|          Nov-50|1950|  Marvel|
|   2458|Benjamin Grimm (E...|\/Benjamin_Grimm_...| Public Identity|   Good
Characters| Blue Eyes|   No Hair|  Male Characters|null|Living Characters|
2255|          Nov-61|1961|  Marvel|
|   2166|Reed Richards (Ea...|\/Reed_Richards_(...| Public Identity|   Good
Characters|Brown Eyes|Brown Hair|  Male Characters|null|Living Characters|
2072|          Nov-61|1961|  Marvel|
|   1833|Hulk (Robert Bruc...|\/Hulk_(Robert_Br...| Public Identity|   Good
Characters|Brown Eyes|Brown Hair|  Male Characters|null|Living Characters|
2017|          May-62|1962|  Marvel|
|  29481|Scott Summers (Ea...|\/Scott_Summers_(...| Public Identity|Neutral
Characters|Brown Eyes|Brown Hair|  Male Characters|null|Living Characters|
1955|          Sep-63|1963|  Marvel|
|   1837|Jonathan Storm (E...|\/Jonathan_Storm_...| Public Identity|   Good
Characters| Blue Eyes|Blond Hair|  Male Characters|null|Living Characters|
1934|          Nov-61|1961|  Marvel|
|  15725|Henry McCoy (Eart...|\/Henry_McCoy_(Ea...| Public Identity|   Good
Characters| Blue Eyes| Blue Hair|  Male Characters|null|Living Characters|
1825|          Sep-63|1963|  Marvel|
|   1863|Susan Storm (Eart...|\/Susan_Storm_(Ea...| Public Identity|   Good
Characters| Blue Eyes|Blond Hair|Female Characters|null|Living Characters|
1713|          Nov-61|1961|  Marvel|
|   7823|Namor McKenzie (E...|\/Namor_McKenzie_...|No Dual Identity|Neutral
Characters|Green Eyes|Black Hair|  Male Characters|null|Living Characters|
1528|            null|null|  Marvel|
|   2614|Ororo Munroe (Ear...|\/Ororo_Munroe_(E...| Public Identity|   Good
Characters| Blue Eyes|White Hair|Female Characters|null|Living Characters|
1512|          May-75|1975|  Marvel|
|   1803|Clinton Barton (E...|\/Clinton_Barton_...| Public Identity|   Good
Characters| Blue Eyes|Blond Hair|  Male Characters|null|Living Characters|
1394|          Sep-64|1964|  Marvel|
|   1396|Matthew Murdock (...|\/Matthew_Murdock...| Public Identity|   Good
Characters| Blue Eyes|  Red Hair|  Male Characters|null|Living Characters|
1338|          Apr-64|1964|  Marvel|
|  55534|Stephen Strange (...|\/Stephen_Strange...| Public Identity|   Good
Characters| Grey Eyes|Black Hair|  Male Characters|null|Living Characters|
1307|          Jul-63|1963|  Marvel|
|   1978|Mary Jane Watson ...|\/Mary_Jane_Watso...|No Dual Identity|   Good
```

```
Characters|Green Eyes|  Red Hair|Female Characters|null|Living Characters|
1304|          Jun-65|1965|  Marvel|
|   1872|John Jonah Jameso...|\/John_Jonah_Jame...|No Dual Identity|Neutral
Characters| Blue Eyes|Black Hair|  Male Characters|null|Living Characters|
1266|          Mar-63|1963|  Marvel|
|  35350|Robert Drake (Ear...|\/Robert_Drake_(E...| Secret Identity|   Good
Characters|Brown Eyes|Brown Hair|  Male Characters|null|Living Characters|
1265|          Sep-63|1963|  Marvel|
+-------+--------------------+--------------------+---------------+---------
---------+----------+----------+-----------------+----+-----------------+----
-------+---------------+----+--------+
only showing top 20 rows
```

Объединим все данные в одном представлении.

```
spark.sql("""
CREATE OR REPLACE TEMPORARY VIEW superheroes AS
SELECT * FROM marvel_superheroes
UNION ALL
SELECT * FROM dc_superheroes""")
```

```
DataFrame[]
```

8.2. Выборка данных с помощью SQL

Выбираем данные из представления простым запросом

```
spark.sql("SELECT * FROM superheroes").show()
```

```
+-------+--------------------+--------------------+---------------+---------
---------+----------+----------+-----------------+----+-----------------+----
-------+---------------+----+--------+
|page_id|                name|             urlslug|             ID|
ALIGN|       EYE|      HAIR|              SEX| GSM|
ALIVE|APPEARANCES|FIRST APPEARANCE|Year|universe|
+-------+--------------------+--------------------+---------------+---------
---------+----------+----------+-----------------+----+-----------------+----
-------+---------------+----+--------+
|   1678|Spider-Man (Peter...|\/Spider-Man_(Pet...| Secret Identity|   Good
Characters|Hazel Eyes|Brown Hair|  Male Characters|null|Living Characters|
4043|          Aug-62|1962|  Marvel|
|   7139|Captain America (...|\/Captain_America...| Public Identity|   Good
Characters| Blue Eyes|White Hair|  Male Characters|null|Living Characters|
3360|          Mar-41|1941|  Marvel|
|  64786|"Wolverine (James...|\/Wolverine_(Jame...| Public Identity|Neutral
Characters| Blue Eyes|Black Hair|  Male Characters|null|Living Characters|
3061|          Oct-74|1974|  Marvel|
|   1868|"Iron Man (Anthon...|\/Iron_Man_(Antho...| Public Identity|   Good
Characters| Blue Eyes|Black Hair|  Male Characters|null|Living Characters|
2961|          Mar-63|1963|  Marvel|
|   2460| Thor (Thor Odinson)|\/Thor_(Thor_Odin...|No Dual Identity|   Good
Characters| Blue Eyes|Blond Hair|  Male Characters|null|Living Characters|
2258|          Nov-50|1950|  Marvel|
|   2458|Benjamin Grimm (E...|\/Benjamin_Grimm_...| Public Identity|   Good
Characters| Blue Eyes|   No Hair|  Male Characters|null|Living Characters|
```

```
2255|          Nov-61|1961|  Marvel|
|   2166|Reed Richards (Ea...|\/Reed_Richards_(...| Public Identity|   Good
Characters|Brown Eyes|Brown Hair|  Male Characters|null|Living Characters|
2072|          Nov-61|1961|  Marvel|
|   1833|Hulk (Robert Bruc...|\/Hulk_(Robert_Br...| Public Identity|   Good
Characters|Brown Eyes|Brown Hair|  Male Characters|null|Living Characters|
2017|          May-62|1962|  Marvel|
|  29481|Scott Summers (Ea...|\/Scott_Summers_(...| Public Identity|Neutral
Characters|Brown Eyes|Brown Hair|  Male Characters|null|Living Characters|
1955|          Sep-63|1963|  Marvel|
|   1837|Jonathan Storm (E...|\/Jonathan_Storm_...| Public Identity|   Good
Characters| Blue Eyes|Blond Hair|  Male Characters|null|Living Characters|
1934|          Nov-61|1961|  Marvel|
|  15725|Henry McCoy (Eart...|\/Henry_McCoy_(Ea...| Public Identity|   Good
Characters| Blue Eyes| Blue Hair|  Male Characters|null|Living Characters|
1825|          Sep-63|1963|  Marvel|
|   1863|Susan Storm (Eart...|\/Susan_Storm_(Ea...| Public Identity|   Good
Characters| Blue Eyes|Blond Hair|Female Characters|null|Living Characters|
1713|          Nov-61|1961|  Marvel|
|   7823|Namor McKenzie (E...|\/Namor_McKenzie_...|No Dual Identity|Neutral
Characters|Green Eyes|Black Hair|  Male Characters|null|Living Characters|
1528|            null|null|  Marvel|
|   2614|Ororo Munroe (Ear...|\/Ororo_Munroe_(E...| Public Identity|   Good
Characters| Blue Eyes|White Hair|Female Characters|null|Living Characters|
1512|          May-75|1975|  Marvel|
|   1803|Clinton Barton (E...|\/Clinton_Barton_...| Public Identity|   Good
Characters| Blue Eyes|Blond Hair|  Male Characters|null|Living Characters|
1394|          Sep-64|1964|  Marvel|
|   1396|Matthew Murdock (...|\/Matthew_Murdock...| Public Identity|   Good
Characters| Blue Eyes|  Red Hair|  Male Characters|null|Living Characters|
1338|          Apr-64|1964|  Marvel|
|  55534|Stephen Strange (...|\/Stephen_Strange...| Public Identity|   Good
Characters| Grey Eyes|Black Hair|  Male Characters|null|Living Characters|
1307|          Jul-63|1963|  Marvel|
|   1978|Mary Jane Watson ...|\/Mary_Jane_Watso...|No Dual Identity|   Good
Characters|Green Eyes|  Red Hair|Female Characters|null|Living Characters|
1304|          Jun-65|1965|  Marvel|
|   1872|John Jonah Jameso...|\/John_Jonah_Jame...|No Dual Identity|Neutral
Characters| Blue Eyes|Black Hair|  Male Characters|null|Living Characters|
1266|          Mar-63|1963|  Marvel|
|  35350|Robert Drake (Ear...|\/Robert_Drake_(E...| Secret Identity|   Good
Characters|Brown Eyes|Brown Hair|  Male Characters|null|Living Characters|
1265|          Sep-63|1963|  Marvel|
+-------+--------------------+--------------------+---------------+---------
---------+----------+----------+-----------------+----+-----------------+----
-------+---------------+----+--------+
only showing top 20 rows
```

Можно выбрать только интересующие нас столбцы

```
spark.sql("SELECT name, alive FROM superheroes").show()
```

```
+--------------------+-----------------+
|                name|            alive|
+--------------------+-----------------+
```

```
|Spider-Man (Peter...|Living Characters|
|Captain America (...|Living Characters|
|"Wolverine (James...|Living Characters|
|"Iron Man (Anthon...|Living Characters|
| Thor (Thor Odinson)|Living Characters|
|Benjamin Grimm (E...|Living Characters|
|Reed Richards (Ea...|Living Characters|
|Hulk (Robert Bruc...|Living Characters|
|Scott Summers (Ea...|Living Characters|
|Jonathan Storm (E...|Living Characters|
|Henry McCoy (Eart...|Living Characters|
|Susan Storm (Eart...|Living Characters|
|Namor McKenzie (E...|Living Characters|
|Ororo Munroe (Ear...|Living Characters|
|Clinton Barton (E...|Living Characters|
|Matthew Murdock (...|Living Characters|
|Stephen Strange (...|Living Characters|
|Mary Jane Watson ...|Living Characters|
|John Jonah Jameso...|Living Characters|
|Robert Drake (Ear...|Living Characters|
+--------------------+-----------------+
only showing top 20 rows
```

Если требуется наложить фильтрацию на строки, то это делается так же как и в РСУБД, на обычном SQL:

```
spark.sql("SELECT name, hair FROM superheroes WHERE sex = 'Female
Characters'").show()
```

```
+--------------------+--------------------+
|                name|                hair|
+--------------------+--------------------+
|Susan Storm (Eart...|          Blond Hair|
|Ororo Munroe (Ear...|          White Hair|
|Mary Jane Watson ...|            Red Hair|
|Wanda Maximoff (E...|          Brown Hair|
|Janet van Dyne (E...|         Auburn Hair|
|Jean Grey (Earth-...|            Red Hair|
|Natalia Romanova ...|            Red Hair|
|May Reilly (Earth...|           Grey Hair|
|Katherine Pryde (...|          Brown Hair|
|Carol Danvers (Ea...|          Blond Hair|
|Jennifer Walters ...|          Brown Hair|
|Emma Frost (Earth...|          Brown Hair|
|Rogue (Anna Marie...|         Auburn Hair|
|Elizabeth Braddoc...|         Purple Hair|
|Elizabeth Brant (...|          Brown Hair|
|Patricia Walker (...|            Red Hair|
|Crystalia Amaquel...|Strawberry Blond ...|
|Jessica Drew (Ear...|         Auburn Hair|
|Rahne Sinclair (E...|            Red Hair|
|Elizabeth Ross (E...|          Black Hair|
+--------------------+--------------------+
only showing top 20 rows
```

8.3. Агрегация данных

```
# Супергерои мужчины
spark.sql("SELECT COUNT(*) FROM superheroes WHERE sex = 'Male
Characters'").show()
```

```
+--------+
|count(1)|
+--------+
|   16421|
+--------+
```

```
# Супергерои женщины
spark.sql("SELECT COUNT(*) FROM superheroes WHERE sex = 'Female
Characters'").show()
```

```
+--------+
|count(1)|
+--------+
|    5804|
+--------+
```

Считаем, сколько хороших, плохих и нейтральных супергероев

```
spark.sql("SELECT align, COUNT(*) FROM superheroes GROUP BY align").show()
```

```
+------------------+--------+
|             align|count(1)|
+------------------+--------+
|   Good Characters|    7468|
|Neutral Characters|    2773|
|    Bad Characters|    9615|
|Reformed Criminals|       3|
|              null|    3413|
+------------------+--------+
```

Считаем, сколько хороших, плохих и нейтральных супергероев отдельно по вселенным Marvel и DC

```
spark.sql("SELECT universe, align, COUNT(*) FROM superheroes GROUP BY
universe, align").show()
```

```
+--------+------------------+--------+
|universe|             align|count(1)|
+--------+------------------+--------+
|  Marvel|   Good Characters|    4636|
|  Marvel|    Bad Characters|    6720|
|  Marvel|Neutral Characters|    2208|
|      DC|Reformed Criminals|       3|
|      DC|Neutral Characters|     565|
|      DC|    Bad Characters|    2895|
|      DC|   Good Characters|    2832|
|  Marvel|              null|    2812|
|      DC|              null|     601|
```

```
+--------+-----------------+--------+
```

8.4. Сортировка данных

Сортируем количество хороших, плохих и нейтральных супергероев отдельно по вселенным Marvel и DC

```
spark.sql("SELECT universe, align, COUNT(*) FROM superheroes GROUP BY
universe, align ORDER BY universe, COUNT(*) desc").show()
```

```
+--------+-----------------+--------+
|universe|            align|count(1)|
+--------+-----------------+--------+
|      DC|    Bad Characters|    2895|
|      DC|   Good Characters|    2832|
|      DC|             null|     601|
|      DC|Neutral Characters|     565|
|      DC|Reformed Criminals|       3|
|  Marvel|    Bad Characters|    6720|
|  Marvel|   Good Characters|    4636|
|  Marvel|             null|    2812|
|  Marvel|Neutral Characters|    2208|
+--------+-----------------+--------+
```

##5. Соединение таблиц Создадим два представления для подсчёта числа героев разного типажа, и с разным цветом глаз.

```
spark.sql("""CREATE OR REPLACE TEMPORARY VIEW aligned_eye_heroes AS
SELECT align, eye, count(*) a_e_cnt
FROM superheroes
GROUP BY align,eye""")
```

```
DataFrame[]
```

```
spark.sql("""CREATE OR REPLACE TEMPORARY VIEW eyed_heroes AS
SELECT eye, count(*) e_cnt
FROM superheroes
GROUP BY eye""")
```

```
DataFrame[]
```

Вычислим процент хороших, плохих и нейтральных героев с разным цветом глаз. Для этого используем внутреннее соединение (inner join).

```
spark.sql("""CREATE OR REPLACE TEMPORARY VIEW percent AS
SELECT eye, align, round(a_e_cnt/e_cnt*100,2) perc
FROM aligned_eye_heroes
  JOIN eyed_heroes USING(eye)
ORDER BY 1,2""")
spark.sql("SELECT * FROM percent").show()
```

```
+-------------+-----------------+-----+
|          eye|            align| perc|
+-------------+-----------------+-----+
|    Amber Eyes|             null| 20.0|
```

```
|    Amber Eyes|    Bad Characters|26.67|
|    Amber Eyes|   Good Characters|26.67|
|    Amber Eyes|Neutral Characters|26.67|
|   Auburn Hair|              null|14.29|
|   Auburn Hair|    Bad Characters|28.57|
|   Auburn Hair|   Good Characters|28.57|
|   Auburn Hair|Neutral Characters|28.57|
|Black Eyeballs|   Good Characters|66.67|
|Black Eyeballs|Neutral Characters|33.33|
|    Black Eyes|              null|12.62|
|    Black Eyes|    Bad Characters| 39.4|
|    Black Eyes|   Good Characters|32.57|
|    Black Eyes|Neutral Characters|15.41|
|     Blue Eyes|              null| 9.43|
|     Blue Eyes|    Bad Characters|31.14|
|     Blue Eyes|   Good Characters|45.76|
|     Blue Eyes|Neutral Characters|13.61|
|     Blue Eyes|Reformed Criminals| 0.07|
|    Brown Eyes|              null|12.02|
+--------------+------------------+-----+
only showing top 20 rows
```

Соединим левым внешним соединением (left outer join) по ключу (`eye,align`) таблицу с героями и таблицу с процентом таких персонажей.

```
spark.sql("""SELECT name, p.eye, p.align, perc FROM superheroes s left join
percent p on s.eye = p.eye and s.align = p.align """).show()
```

```
+--------------------+----------+---------------+-----+
|                name|       eye|          align| perc|
+--------------------+----------+---------------+-----+
|Alexander Luthor ...|Green Eyes| Bad Characters|40.38|
|Captain America (...| Blue Eyes|Good Characters|45.76|
|"Iron Man (Anthon...| Blue Eyes|Good Characters|45.76|
| Thor (Thor Odinson)| Blue Eyes|Good Characters|45.76|
|Benjamin Grimm (E...| Blue Eyes|Good Characters|45.76|
|Jonathan Storm (E...| Blue Eyes|Good Characters|45.76|
|Henry McCoy (Eart...| Blue Eyes|Good Characters|45.76|
|Susan Storm (Eart...| Blue Eyes|Good Characters|45.76|
|Ororo Munroe (Ear...| Blue Eyes|Good Characters|45.76|
|Clinton Barton (E...| Blue Eyes|Good Characters|45.76|
|Matthew Murdock (...| Blue Eyes|Good Characters|45.76|
|Henry Pym (Earth-...| Blue Eyes|Good Characters|45.76|
|Batman (Bruce Wayne)| Blue Eyes|Good Characters|45.76|
|Superman (Clark K...| Blue Eyes|Good Characters|45.76|
|Richard Grayson (...| Blue Eyes|Good Characters|45.76|
|Wonder Woman (Dia...| Blue Eyes|Good Characters|45.76|
|Aquaman (Arthur C...| Blue Eyes|Good Characters|45.76|
|Timothy Drake (Ne...| Blue Eyes|Good Characters|45.76|
|Dinah Laurel Lanc...| Blue Eyes|Good Characters|45.76|
| Flash (Barry Allen)| Blue Eyes|Good Characters|45.76|
+--------------------+----------+---------------+-----+
only showing top 20 rows
```

Сортируем супергероев по количеству появлений

```
spark.sql("SELECT name, appearances FROM superheroes ORDER BY appearances
desc").show()
```

```
+--------------------+-----------+
|                name|appearances|
+--------------------+-----------+
|Spider-Man (Peter...|       4043|
|Captain America (...|       3360|
|Batman (Bruce Wayne)|       3093|
|"Wolverine (James...|       3061|
|"Iron Man (Anthon...|       2961|
|Superman (Clark K...|       2496|
| Thor (Thor Odinson)|       2258|
|Benjamin Grimm (E...|       2255|
|Reed Richards (Ea...|       2072|
|Hulk (Robert Bruc...|       2017|
|Scott Summers (Ea...|       1955|
|Jonathan Storm (E...|       1934|
|Henry McCoy (Eart...|       1825|
|Susan Storm (Eart...|       1713|
|Green Lantern (Ha...|       1565|
|Namor McKenzie (E...|       1528|
|Ororo Munroe (Ear...|       1512|
|Clinton Barton (E...|       1394|
|Matthew Murdock (...|       1338|
|James Gordon (New...|       1316|
+--------------------+-----------+
only showing top 20 rows
```

Находим наиболее популярный цвет волос у супергероев женщин

```
spark.sql("SELECT hair, count(*) FROM superheroes WHERE sex='Female
Characters' GROUP BY hair ORDER BY count(*) desc").show()
```

```
+--------------------+--------+
|                hair|count(1)|
+--------------------+--------+
|          Black Hair|    1557|
|          Blond Hair|    1044|
|                null|     999|
|          Brown Hair|     875|
|            Red Hair|     557|
|          White Hair|     203|
|           Grey Hair|     101|
|             No Hair|      92|
|          Green Hair|      69|
|         Auburn Hair|      50|
|         Purple Hair|      50|
|Strawberry Blond ...|      46|
|                Bald|      43|
|           Blue Hair|      43|
|           Pink Hair|      30|
|         Orange Hair|      14|
```

```
|         Silver Hair|      10|
|       Variable Hair|       5|
|         Yellow Hair|       4|
|        Magenta Hair|       4|
+--------------------+--------+
only showing top 20 rows
```

Наиболее популярные цвета волос у положительных и отрицательных супергероев

```
spark.sql("""SELECT hair, count(*) FROM superheroes
                                   WHERE sex='Female Characters' and align ==
'Good Characters'
                                   GROUP BY hair
                                   ORDER BY count(*) desc""").show()
```

```
+--------------------+--------+
|                hair|count(1)|
+--------------------+--------+
|          Black Hair|     693|
|          Blond Hair|     487|
|          Brown Hair|     423|
|                null|     348|
|            Red Hair|     215|
|          White Hair|      96|
|           Grey Hair|      48|
|Strawberry Blond ...|      27|
|             No Hair|      25|
|         Purple Hair|      25|
|          Green Hair|      24|
|         Auburn Hair|      22|
|                Bald|      14|
|           Blue Hair|      11|
|           Pink Hair|      11|
|         Orange Hair|       8|
|         Silver Hair|       4|
|        Magenta Hair|       2|
|       Variable Hair|       2|
|         Violet Hair|       2|
+--------------------+--------+
only showing top 20 rows
```

```
spark.sql("""SELECT hair, count(*) FROM superheroes
                                   WHERE sex='Female Characters' and align ==
'Bad Characters'
                                   GROUP BY hair
                                   ORDER BY count(*) desc""").show()
```

```
+--------------------+--------+
|                hair|count(1)|
+--------------------+--------+
|          Black Hair|     448|
|                null|     304|
|          Blond Hair|     257|
|          Brown Hair|     176|
```

```
|            Red Hair|     153|
|          White Hair|      56|
|             No Hair|      39|
|          Green Hair|      28|
|           Grey Hair|      23|
|                Bald|      19|
|           Blue Hair|      17|
|         Purple Hair|      14|
|         Auburn Hair|      12|
|           Pink Hair|      11|
|Strawberry Blond ...|       8|
|         Yellow Hair|       2|
|        Magenta Hair|       2|
|       Variable Hair|       1|
|         Silver Hair|       1|
| Platinum Blond Hair|       1|
+--------------------+--------+
only showing top 20 rows
```

##6. Выбор уникальных значений Определяем, какого цвета глаза встречаются у супергероев

```
spark.sql("SELECT DISTINCT(eye) FROM superheroes ORDER BY 1").show(30)
```

```
+------------------+
|               eye|
+------------------+
|              null|
|        Amber Eyes|
|       Auburn Hair|
|    Black Eyeballs|
|        Black Eyes|
|         Blue Eyes|
|        Brown Eyes|
|     Compound Eyes|
|         Gold Eyes|
|        Green Eyes|
|         Grey Eyes|
|        Hazel Eyes|
|      Magenta Eyes|
|     Multiple Eyes|
|           No Eyes|
|           One Eye|
|       Orange Eyes|
|Photocellular Eyes|
|         Pink Eyes|
|       Purple Eyes|
|          Red Eyes|
|       Silver Eyes|
|     Variable Eyes|
|       Violet Eyes|
|        White Eyes|
|   Yellow Eyeballs|
|       Yellow Eyes|
```

```
+------------------+
```

Третья строка DataFrame с цветом глаз содержит ошибочное значение: 'Auburn Hair' - это не цвет глаз, а цвет волос. Давайте посмотрим, у каких героев заполнено ошибочное значение.

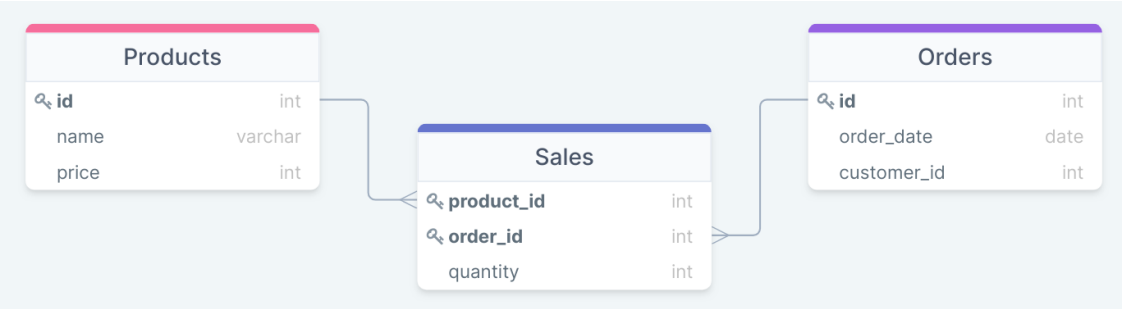
```
spark.sql("SELECT * FROM superheroes WHERE eye = 'Auburn Hair'").show()
```

```
+-------+--------------------+--------------------+--------------+---------
--------+-----------+----+----------------+----+------------------+--------
---+---------------+----+--------+
|page_id|                name|             urlslug|            ID|
ALIGN|        EYE|HAIR|              SEX| GSM|
ALIVE|APPEARANCES|FIRST APPEARANCE|Year|universe|
+-------+--------------------+--------------------+--------------+---------
--------+-----------+----+----------------+----+------------------+--------
---+---------------+----+--------+
|  80676|Marcia King (New ...|\/wiki\/Marcia_Ki...|Public Identity|   Good
Characters|Auburn Hair|null|Female Characters|null|  Living Characters|
32|     1984, April|1984|      DC|
| 146812|Anthony Angelo, J...|\/wiki\/Anthony_A...|Public Identity|Neutral
Characters|Auburn Hair|null|  Male Characters|null|  Living Characters|
14|     1989, April|1989|      DC|
| 114487|   Tawna (New Earth)|\/wiki\/Tawna_(Ne...|           null|   Good
Characters|Auburn Hair|null|Female Characters|null|  Living Characters|
5|     1998, April|1998|      DC|
| 192614|Razerkut (New Earth)|\/wiki\/Razerkut_...|Secret Identity|    Bad
Characters|Auburn Hair|null|Female Characters|null|  Living Characters|
5|    1995, August|1995|      DC|
| 130938|Carla Draper (New...|\/wiki\/Carla_Dra...|Public Identity|    Bad
Characters|Auburn Hair|null|Female Characters|null|  Living Characters|
5|  1994, November|1994|      DC|
|  71092|Madolyn Corbett (...|\/wiki\/Madolyn_C...|           null|Neutral
Characters|Auburn Hair|null|Female Characters|null|Deceased Characters|
4|     1995, April|1995|      DC|
|   4935|Sally Milton (New...|\/wiki\/Sally_Mil...|Public Identity|
null|Auburn Hair|null|Female Characters|null|  Living Characters|          2|
1988, Holiday|1988|      DC|
+-------+--------------------+--------------------+--------------+---------
--------+-----------+----+----------------+----+------------------+--------
---+---------------+----+--------+
```

9. Аналитика данных с SQL в Apache Spark

9.1. Создание кадров данных

Создадим три кадра данных (DataFrame'а), представляющие данные о продажах онлайн-школы. Схема данных выглядит следующим образом:



Онлайн-школа продает образовательные продукты: онлайн-курсы, книги, семинары и т.п. Описание и стоимость продуктов содержится в таблице **Products**. Когда клиент что-то покупает, создается заказ, который заносится в таблицу **Orders**. Заказ может содежать несколько продуктов, перечень продуктов в заказах содержится в таблице **Sales**.

Таблица **Products** - продукты онлайн-школы:

- **id** - идентификатор продукта
- **name** - название продукта
- **price** - стоимость продукта

Таблица **Oredrs** - заказы:

- **id** - идентификатор заказа
- **order_date** - дата заказа
- **customer_id** - идентификатор заказчика (таблица с заказчиками не создается для упрощения примера)

Таблица **Sales** - продажи:

- **product_id** - идентификатор продукта, ссылка на таблицу Products, поле id
- **order_date** - идентификатор заказа, ссылка на таблицу Orders, поле id
- **quantity** - количество продуктов в заказе

###Создаем кадр данных для продуктов

```
products_list = [(1, "Онлайн-курс 'Большие данные'", 7000),
                 (2, "Онлайн-курс 'Искусственный интеллект'", 10000),
                 (3, "Онлайн-курс 'Нейронные сети'", 8000),
                 (4, "Онлайн-курс 'Машинное обучение'", 5000),
                 (5, "Книга 'Программирование нейронных сетей'", 500),
                 (6, "Семинар 'Планирование карьеры в Data Science'", 1000)]
```

Для конструирования схемы данных импортируем дополнительные классы.

```
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType, DateType
from pyspark.sql.functions import to_date

products_schema = StructType([StructField('id', IntegerType(), False),
                              StructField('name', StringType(), False),
                              StructField('price', IntegerType(), False)
])
```

```
products_df = spark.createDataFrame(products_list, schema=products_schema)

products_df.show(truncate=False)
```

```
+---+-----------------------------------------------+-----+
|id |name                                           |price|
+---+-----------------------------------------------+-----+
|1  |Онлайн-курс 'Большие данные'                    |7000 |
|2  |Онлайн-курс 'Искусственный интеллект'          |10000|
|3  |Онлайн-курс 'Нейронные сети'                    |8000 |
|4  |Онлайн-курс 'Машинное обучение'                 |5000 |
|5  |Книга 'Программирование нейронных сетей'       |500  |
|6  |Семинар 'Планирование карьеры в Data Science'|1000 |
+---+-----------------------------------------------+-----+
```

```
products_df.printSchema()
```

```
root
 |-- id: integer (nullable = false)
 |-- name: string (nullable = false)
 |-- price: integer (nullable = false)
```

###Создаем кадр данных для заказов

```
orders_list = [(1, '2020-09-01', 500),
               (2, '2020-09-01', 510),
               (3, '2020-09-01', 501),
               (4, '2020-09-02', 503),
               (5, '2020-09-02', 515),
               (6, '2020-09-03', 657),
               (7, '2020-09-03', 510),
               (8, '2020-09-03', 999),
               (9, '2020-09-04', 104),
               (10, '2020-09-05', 501),
               (11, '2020-09-05', 510),
               ]

orders_schema = StructType([StructField('id', IntegerType(), False),
                            StructField('order_date', StringType(), False),
                            StructField('customer_id', IntegerType(), False)
])

orders_df = spark.createDataFrame(orders_list, schema=orders_schema)
orders_df.printSchema()
```

```
root
 |-- id: integer (nullable = false)
 |-- order_date: string (nullable = false)
 |-- customer_id: integer (nullable = false)
```

```
orders_df = orders_df.withColumn('order_date', to_date('order_date'))
orders_df.printSchema()
```

```
root
 |-- id: integer (nullable = false)
```

```
 |-- order_date: date (nullable = true)
 |-- customer_id: integer (nullable = false)


orders_df.show()

+---+----------+-----------+
| id|order_date|customer_id|
+---+----------+-----------+
|  1|2020-09-01|        500|
|  2|2020-09-01|        510|
|  3|2020-09-01|        501|
|  4|2020-09-02|        503|
|  5|2020-09-02|        515|
|  6|2020-09-03|        657|
|  7|2020-09-03|        510|
|  8|2020-09-03|        999|
|  9|2020-09-04|        104|
| 10|2020-09-05|        501|
| 11|2020-09-05|        510|
+---+----------+-----------+
```

###Создаем кадр данных для продаж

```
sales_list = [(2, 1, 10),
              (4, 1, 10),
              (3, 2, 1),
              (4, 3, 1),
              (1, 4, 1),
              (1, 5, 1),
              (2, 6, 3),
              (3, 6, 3),
              (5, 7, 5),
              (5, 8, 5),
              (2, 9, 1),
              (1, 9, 1),
              (2, 10, 1),
              (4, 11, 1),
              (7, 12, 1),
              ]

sales_schema = StructType([StructField('product_id', IntegerType(), False),
                           StructField('order_id', IntegerType(), False),
                           StructField('quantity', IntegerType(), False)
])

sales_df = spark.createDataFrame(sales_list, schema=sales_schema)

sales_df.printSchema()

root
 |-- product_id: integer (nullable = false)
 |-- order_id: integer (nullable = false)
 |-- quantity: integer (nullable = false)
```

```
sales_df.show()
```

```
+----------+--------+--------+
|product_id|order_id|quantity|
+----------+--------+--------+
|         2|       1|      10|
|         4|       1|      10|
|         3|       2|       1|
|         4|       3|       1|
|         1|       4|       1|
|         1|       5|       1|
|         2|       6|       3|
|         3|       6|       3|
|         5|       7|       5|
|         5|       8|       5|
|         2|       9|       1|
|         1|       9|       1|
|         2|      10|       1|
|         4|      11|       1|
|         7|      12|       1|
+----------+--------+--------+
```

###Регистрируем кадры данных в качестве временных представлений Кадры данных можно кэшировать прямо из SQL. Для этого используется конструкция `cache lazy table`

```
products_df.createOrReplaceTempView("products")
orders_df.createOrReplaceTempView("orders")
sales_df.createOrReplaceTempView("sales")
```

```
# memory_and_disk only
spark.sql("""cache lazy table products""")
spark.sql("""cache lazy table orders""")
spark.sql("""cache lazy table sales""")
```

```
DataFrame[]
```

9.2. Простейшая аналитика с использованием SQL

Находим продукты, которые были проданы хотя бы один раз. Объединяем представления products и sales.

```
sales_with_products = products_df.join(sales_df, products_df['id'] ==
sales_df['product_id'])
```

```
sales_with_products.show(truncate=False)
```

```
+---+---------------------------------------+-----+----------+--------+-----
---+
|id |name
|price|product_id|order_id|quantity|
+---+---------------------------------------+-----+----------+--------+-----
---+
|1  |Онлайн-курс 'Большие данные'           |7000 |1         |4       |1
|
```

```
|1  |Онлайн-курс 'Большие данные'             |7000 |1         |5       |1
|
|1  |Онлайн-курс 'Большие данные'             |7000 |1         |9       |1
|
|2  |Онлайн-курс 'Искусственный интеллект'    |10000|2         |1       |10
|
|2  |Онлайн-курс 'Искусственный интеллект'    |10000|2         |6       |3
|
|2  |Онлайн-курс 'Искусственный интеллект'    |10000|2         |9       |1
|
|2  |Онлайн-курс 'Искусственный интеллект'    |10000|2         |10      |1
|
|3  |Онлайн-курс 'Нейронные сети'             |8000 |3         |2       |1
|
|3  |Онлайн-курс 'Нейронные сети'             |8000 |3         |6       |3
|
|4  |Онлайн-курс 'Машинное обучение'          |5000 |4         |1       |10
|
|4  |Онлайн-курс 'Машинное обучение'          |5000 |4         |3       |1
|
|4  |Онлайн-курс 'Машинное обучение'          |5000 |4         |11      |1
|
|5  |Книга 'Программирование нейронных сетей'|500  |5         |7       |5
|
|5  |Книга 'Программирование нейронных сетей'|500  |5         |8       |5
|
+---+----------------------------------------+-----+----------+--------+-----
---+
```

Выбираем уникальные проданные продукты

```
sales_with_products.select('id', 'name').distinct().show(truncate=False)
```

```
+---+----------------------------------------+
|id |name                                    |
+---+----------------------------------------+
|1  |Онлайн-курс 'Большие данные'            |
|2  |Онлайн-курс 'Искусственный интеллект'   |
|3  |Онлайн-курс 'Нейронные сети'            |
|4  |Онлайн-курс 'Машинное обучение'         |
|5  |Книга 'Программирование нейронных сетей'|
+---+----------------------------------------+
```

Решение через SQL запрос с объединением таблиц

```
spark.sql("SELECT DISTINCT id, name FROM products JOIN sales ON products.id
== sales.product_id").show(truncate=False)
```

```
+---+----------------------------------------+
|id |name                                    |
+---+----------------------------------------+
|1  |Онлайн-курс 'Большие данные'            |
|2  |Онлайн-курс 'Искусственный интеллект'   |
|3  |Онлайн-курс 'Нейронные сети'            |
|4  |Онлайн-курс 'Машинное обучение'         |
```

```
|5  |Книга 'Программирование нейронных сетей'|
+---+----------------------------------------+
```

Решение через SQL запрос с вложенным запросом

```
spark.sql("SELECT DISTINCT id, name FROM products WHERE id IN (SELECT
product_id FROM SALES)").show(truncate=False)
```

```
+---+----------------------------------------+
|id |name                                    |
+---+----------------------------------------+
|1  |Онлайн-курс 'Большие данные'            |
|2  |Онлайн-курс 'Искусственный интеллект'   |
|3  |Онлайн-курс 'Нейронные сети'            |
|4  |Онлайн-курс 'Машинное обучение'         |
|5  |Книга 'Программирование нейронных сетей'|
+---+----------------------------------------+
```

Решение через SQL запрос с кореллирующим подзапросом

```
spark.sql("SELECT id, name FROM products WHERE EXISTS (SELECT 1 FROM sales
WHERE id = product_id )").show(truncate=False)
```

```
+---+----------------------------------------+
|id |name                                    |
+---+----------------------------------------+
|1  |Онлайн-курс 'Большие данные'            |
|2  |Онлайн-курс 'Искусственный интеллект'   |
|3  |Онлайн-курс 'Нейронные сети'            |
|4  |Онлайн-курс 'Машинное обучение'         |
|5  |Книга 'Программирование нейронных сетей'|
+---+----------------------------------------+
```

По-умолчанию в JOIN используется внутреннее объединение (INNER JOIN). В результат объединения попадают данные, для которых есть ключи в обеих таблицах.

Такое поведение объединения можно изменить, если использовать LEFT или RIGHT JOIN.

9.3. Демонстрация LEFT и RIGHT JOIN

LEFT JOIN в DataFrame API

Обратите внимание, что при LEFT JOIN в результат попадают данные о продукте из таблицы Products, который не был продан ни разу.

```
products_df.join(sales_df,
                 products_df['id'] == sales_df['product_id'],
                 how='left'
                 ).select('id', 'name', 'order_id',
'quantity').show(truncate=False)
```

```
+---+-----------------------------------------+--------+--------+
|id |name                                     |order_id|quantity|
+---+-----------------------------------------+--------+--------+
|1  |Онлайн-курс 'Большие данные'              |4       |1       |
|1  |Онлайн-курс 'Большие данные'              |5       |1       |
|1  |Онлайн-курс 'Большие данные'              |9       |1       |
|2  |Онлайн-курс 'Искусственный интеллект'     |1       |10      |
|2  |Онлайн-курс 'Искусственный интеллект'     |6       |3       |
|2  |Онлайн-курс 'Искусственный интеллект'     |9       |1       |
|2  |Онлайн-курс 'Искусственный интеллект'     |10      |1       |
|3  |Онлайн-курс 'Нейронные сети'              |2       |1       |
|3  |Онлайн-курс 'Нейронные сети'              |6       |3       |
|4  |Онлайн-курс 'Машинное обучение'           |1       |10      |
|4  |Онлайн-курс 'Машинное обучение'           |3       |1       |
|4  |Онлайн-курс 'Машинное обучение'           |11      |1       |
|5  |Книга 'Программирование нейронных сетей'  |7       |5       |
|5  |Книга 'Программирование нейронных сетей'  |8       |5       |
|6  |Семинар 'Планирование карьеры в Data Science'|null    |null    |
+---+-----------------------------------------+--------+--------+
```

LEFT JOIN в Spark SQL

```
spark.sql("""SELECT id, name, order_id, quantity
             FROM products
             LEFT JOIN sales
             ON products.id == sales.product_id""").show(truncate=False)
```

```
+---+-----------------------------------------+--------+--------+
|id |name                                     |order_id|quantity|
+---+-----------------------------------------+--------+--------+
|1  |Онлайн-курс 'Большие данные'              |4       |1       |
|1  |Онлайн-курс 'Большие данные'              |5       |1       |
|1  |Онлайн-курс 'Большие данные'              |9       |1       |
|2  |Онлайн-курс 'Искусственный интеллект'     |1       |10      |
|2  |Онлайн-курс 'Искусственный интеллект'     |6       |3       |
|2  |Онлайн-курс 'Искусственный интеллект'     |9       |1       |
|2  |Онлайн-курс 'Искусственный интеллект'     |10      |1       |
|3  |Онлайн-курс 'Нейронные сети'              |2       |1       |
|3  |Онлайн-курс 'Нейронные сети'              |6       |3       |
|4  |Онлайн-курс 'Машинное обучение'           |1       |10      |
|4  |Онлайн-курс 'Машинное обучение'           |3       |1       |
|4  |Онлайн-курс 'Машинное обучение'           |11      |1       |
|5  |Книга 'Программирование нейронных сетей'  |7       |5       |
|5  |Книга 'Программирование нейронных сетей'  |8       |5       |
|6  |Семинар 'Планирование карьеры в Data Science'|null    |null    |
+---+-----------------------------------------+--------+--------+
```

RIGHT JOIN в DataFrame API

```
products_df.join(sales_df,
                 products_df['id'] == sales_df['product_id'],
                 how='right'
                 ).select('id', 'name', 'order_id',
'quantity').show(truncate=False)
```

```
+----+----------------------------------------+--------+--------+
|id  |name                                    |order_id|quantity|
+----+----------------------------------------+--------+--------+
|1   |Онлайн-курс 'Большие данные'            |4       |1       |
|1   |Онлайн-курс 'Большие данные'            |5       |1       |
|1   |Онлайн-курс 'Большие данные'            |9       |1       |
|2   |Онлайн-курс 'Искусственный интеллект'   |1       |10      |
|2   |Онлайн-курс 'Искусственный интеллект'   |6       |3       |
|2   |Онлайн-курс 'Искусственный интеллект'   |9       |1       |
|2   |Онлайн-курс 'Искусственный интеллект'   |10      |1       |
|3   |Онлайн-курс 'Нейронные сети'            |2       |1       |
|3   |Онлайн-курс 'Нейронные сети'            |6       |3       |
|4   |Онлайн-курс 'Машинное обучение'         |1       |10      |
|4   |Онлайн-курс 'Машинное обучение'         |3       |1       |
|4   |Онлайн-курс 'Машинное обучение'         |11      |1       |
|5   |Книга 'Программирование нейронных сетей'|7       |5       |
|5   |Книга 'Программирование нейронных сетей'|8       |5       |
|null|null                                    |12      |1       |
+----+----------------------------------------+--------+--------+
```

RIGHT JOIN в Spark SQL

```
spark.sql("""SELECT id, name, order_id, quantity
             FROM products
             RIGHT JOIN sales
             ON products.id == sales.product_id""").show(truncate=False)
```

```
+----+----------------------------------------+--------+--------+
|id  |name                                    |order_id|quantity|
+----+----------------------------------------+--------+--------+
|1   |Онлайн-курс 'Большие данные'            |4       |1       |
|1   |Онлайн-курс 'Большие данные'            |5       |1       |
|1   |Онлайн-курс 'Большие данные'            |9       |1       |
|2   |Онлайн-курс 'Искусственный интеллект'   |1       |10      |
|2   |Онлайн-курс 'Искусственный интеллект'   |6       |3       |
|2   |Онлайн-курс 'Искусственный интеллект'   |9       |1       |
|2   |Онлайн-курс 'Искусственный интеллект'   |10      |1       |
|3   |Онлайн-курс 'Нейронные сети'            |2       |1       |
|3   |Онлайн-курс 'Нейронные сети'            |6       |3       |
|4   |Онлайн-курс 'Машинное обучение'         |1       |10      |
|4   |Онлайн-курс 'Машинное обучение'         |3       |1       |
|4   |Онлайн-курс 'Машинное обучение'         |11      |1       |
|5   |Книга 'Программирование нейронных сетей'|7       |5       |
|5   |Книга 'Программирование нейронных сетей'|8       |5       |
|null|null                                    |12      |1       |
+----+----------------------------------------+--------+--------+
```

9.4. Common Table Expression

Язык SQL для написания подзапросов позволяет писать общие табличные выражения, которые затем используются, как временные таблицы. Это повышает читабельность и сопровождаемость кода. Кроме того, так подзапросы могут быть

```
WITH cte as (
  SELECT * FROM table WHERE preticate is true
)
SELECT * FROM cte
```

```
spark.sql("""
WITH sales_agg AS (
  SELECT product_id, sum(quantity) product_quantity
  FROM sales
  GROUP BY product_id
)
SELECT *
FROM sales_agg
WHERE product_quantity > 5
""").show(truncate=False)
```

```
+----------+----------------+
|product_id|product_quantity|
+----------+----------------+
|4         |12              |
|2         |15              |
|5         |10              |
+----------+----------------+
```

9.5. Задачи

Какой продукт продали больше всего

Решение с использованием DataFrame API

```
sales_by_products =
sales_df.groupby("product_id").sum("quantity").withColumnRenamed("sum(quantit
y)", "product_quantity")
sales_by_products.show()
```

```
+----------+----------------+
|product_id|product_quantity|
+----------+----------------+
|         1|               3|
|         3|               4|
|         4|              12|
|         2|              15|
|         5|              10|
|         7|               1|
+----------+----------------+
```

```
sales_by_products.join(products_df, sales_by_products['product_id'] ==
products_df['id'])\
                 .select('id', 'name', 'product_quantity')\
                 .sort('product_quantity', ascending=False)\
                 .show(truncate=False)
```

```
+---+----------------------------------------+----------------+
|id |name                                    |product_quantity|
```

```
+---+----------------------------------------+----------------+
|2  |Онлайн-курс 'Искусственный интеллект'   |15              |
|4  |Онлайн-курс 'Машинное обучение'         |12              |
|5  |Книга 'Программирование нейронных сетей'|10              |
|3  |Онлайн-курс 'Нейронные сети'            |4               |
|1  |Онлайн-курс 'Большие данные'            |3               |
+---+----------------------------------------+----------------+
```

Решение с использованием SQL запроса

```
spark.sql("""SELECT id, name, SUM(quantity) product_quantity
             FROM sales
                JOIN products ON id = product_id
             GROUP BY id, name
             ORDER BY product_quantity DESC
             """).show(truncate=False)
```

```
+---+----------------------------------------+----------------+
|id |name                                    |product_quantity|
+---+----------------------------------------+----------------+
|2  |Онлайн-курс 'Искусственный интеллект'   |15              |
|4  |Онлайн-курс 'Машинное обучение'         |12              |
|5  |Книга 'Программирование нейронных сетей'|10              |
|3  |Онлайн-курс 'Нейронные сети'            |4               |
|1  |Онлайн-курс 'Большие данные'            |3               |
+---+----------------------------------------+----------------+
```

Какой продукт принес самую большую выручку

Решение с использованием DataFrame API

Находим количество продаж каждого продукта

```
sales_by_products =
sales_df.groupby("product_id").sum("quantity").withColumnRenamed("sum(quantit
y)", "product_quantity")

sales_by_products.show()
```

```
+----------+----------------+
|product_id|product_quantity|
+----------+----------------+
|         1|               3|
|         3|               4|
|         4|              12|
|         2|              15|
|         5|              10|
+----------+----------------+
```

Объединяем количество продаж продуктов с таблицей продуктов

```
sales_by_products = products_df.join(sales_by_products,
products_df['id']==sales_by_products['product_id'])
```

```
sales_by_products.show()
```

```
+---+--------------------+-----+----------+----------------+
| id|                name|price|product_id|product_quantity|
+---+--------------------+-----+----------+----------------+
|  1|Онлайн-курс 'Боль...| 7000|         1|               3|
|  3|Онлайн-курс 'Нейр...| 8000|         3|               4|
|  2|Онлайн-курс 'Иску...|10000|         2|              15|
|  5|Книга 'Программир...|  500|         5|              10|
|  4|Онлайн-курс 'Маши...| 5000|         4|              12|
+---+--------------------+-----+----------+----------------+
```

Расчитываем доход от продуктов

```
earnings_per_product = sales_by_products.withColumn("revenu",
sales_by_products['product_quantity'] * sales_by_products['price'])

earnings_per_product.show()
```

```
+---+--------------------+-----+----------+----------------+------+
| id|                name|price|product_id|product_quantity|revenu|
+---+--------------------+-----+----------+----------------+------+
|  1|Онлайн-курс 'Боль...| 7000|         1|               3| 21000|
|  3|Онлайн-курс 'Нейр...| 8000|         3|               4| 32000|
|  2|Онлайн-курс 'Иску...|10000|         2|              15|150000|
|  5|Книга 'Программир...|  500|         5|              10|  5000|
|  4|Онлайн-курс 'Маши...| 5000|         4|              12| 60000|
+---+--------------------+-----+----------+----------------+------+
```

Сортируем продукты в порядке убывания почтуплений от продажи

```
earnings_per_product.select('id', 'name', 'revenu').sort('revenu',
ascending=False).show(truncate=False)
```

```
+---+----------------------------------------+------+
|id |name                                    |revenu|
+---+----------------------------------------+------+
|2  |Онлайн-курс 'Искусственный интеллект'   |150000|
|4  |Онлайн-курс 'Машинное обучение'         |60000 |
|3  |Онлайн-курс 'Нейронные сети'            |32000 |
|1  |Онлайн-курс 'Большие данные'            |21000 |
|5  |Книга 'Программирование нейронных сетей'|5000  |
+---+----------------------------------------+------+
```

Решение с помощью SQL запроса

```
spark.sql("""SELECT id, name, SUM(quantity * price)  revenu
             FROM sales
                JOIN products ON id = product_id
             GROUP BY id, name
             ORDER BY revenu DESC
             """).show(truncate=False)
```

```
+---+----------------------------------------+------+
|id |name                                    |revenu|
```

```
+---+----------------------------------------+------+
|2  |Онлайн-курс 'Искусственный интеллект'   |150000|
|4  |Онлайн-курс 'Машинное обучение'         |60000 |
|3  |Онлайн-курс 'Нейронные сети'            |32000 |
|1  |Онлайн-курс 'Большие данные'            |21000 |
|5  |Книга 'Программирование нейронных сетей'|5000  |
+---+----------------------------------------+------+
```