Spark SQL

SparkSQL

- Make data look like tables regardless how they really lay out
- SQL (standard!) based query can be directly against these tables
- Generate a specific execution plan for this query

SparkSQL vs Spark

Spark RDD API

```
rdd
rdd1 = rdd.map(lambda x: x.split("\t"))
rdd2 = rdd1.map(lambda x: (x[0], x[2]))
```

VS

SQL on Spark

```
select user_id, url from access_log
```

- No data parsing
- Code optimization
- Syntax is easier
- No overhead

Write Less Code: Compute an Average

Using RDDs

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
   .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
   .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
   .collect()
```

Using SQL

```
SELECT name, avg(age)
FROM people
GROUP BY name
```

Using DataFrames

```
sqlCtx.table("people") \
   .groupBy("name") \
   .agg("name", avg("age")) \
   .collect()
```

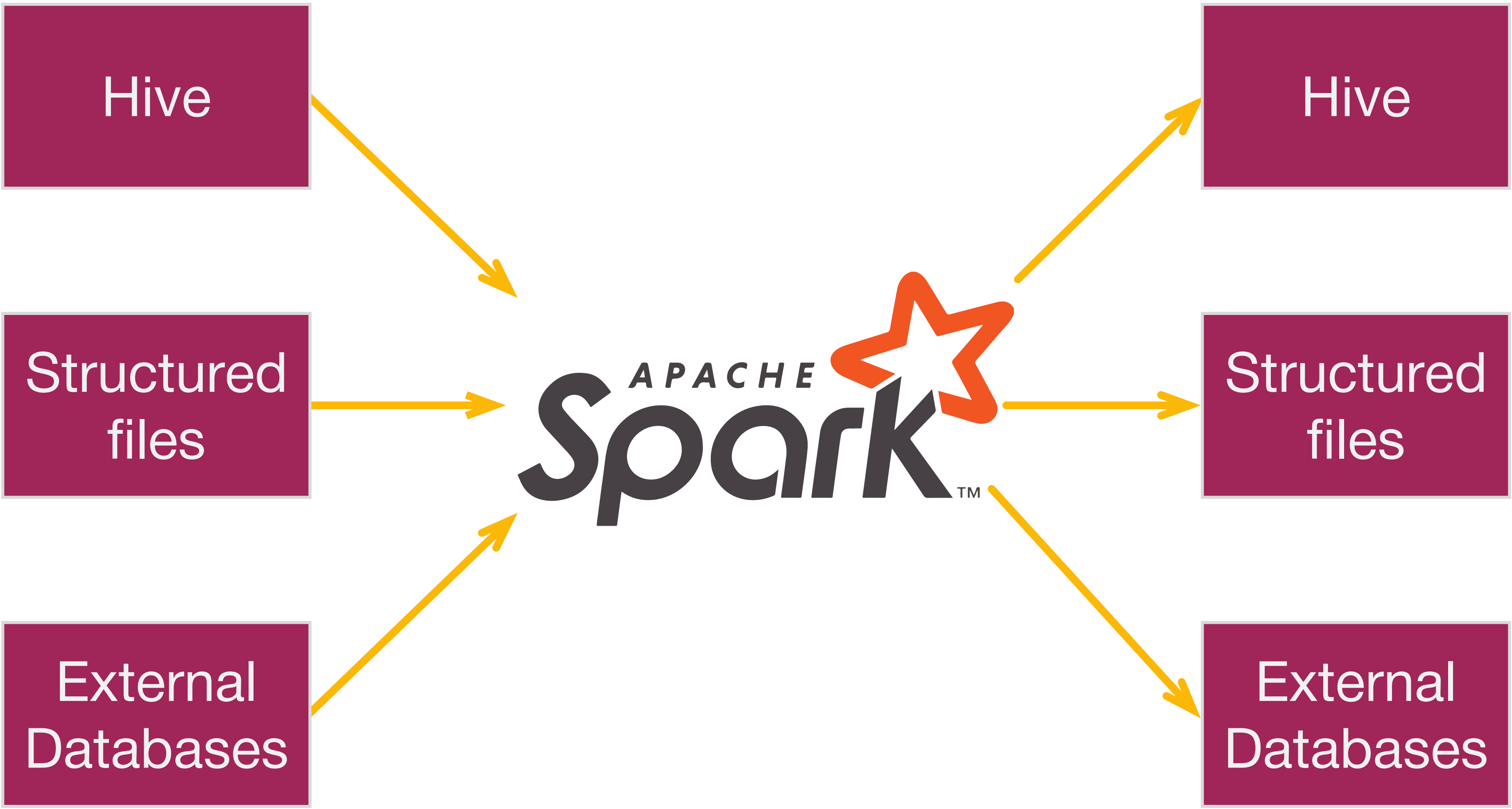
SparkSQL: Spark as a database language? - Application scenario

- No real-time queries (high latency)!
- No row level updates!
- Not designed for online transaction processing!

- Best use: batch jobs over large sets of append-only data
  - Log processing
  - Data/Text mining
  - Business Intelligence
  - …

A step back …What is Hive?

- A big data management system storing structured data on Hadoop file system
- It projects tabular schemas over folders and files in HDFS
- Enables the contents of folders in HDFS to be queried as tables, using SQL-like query semantics



Hive

Structured files

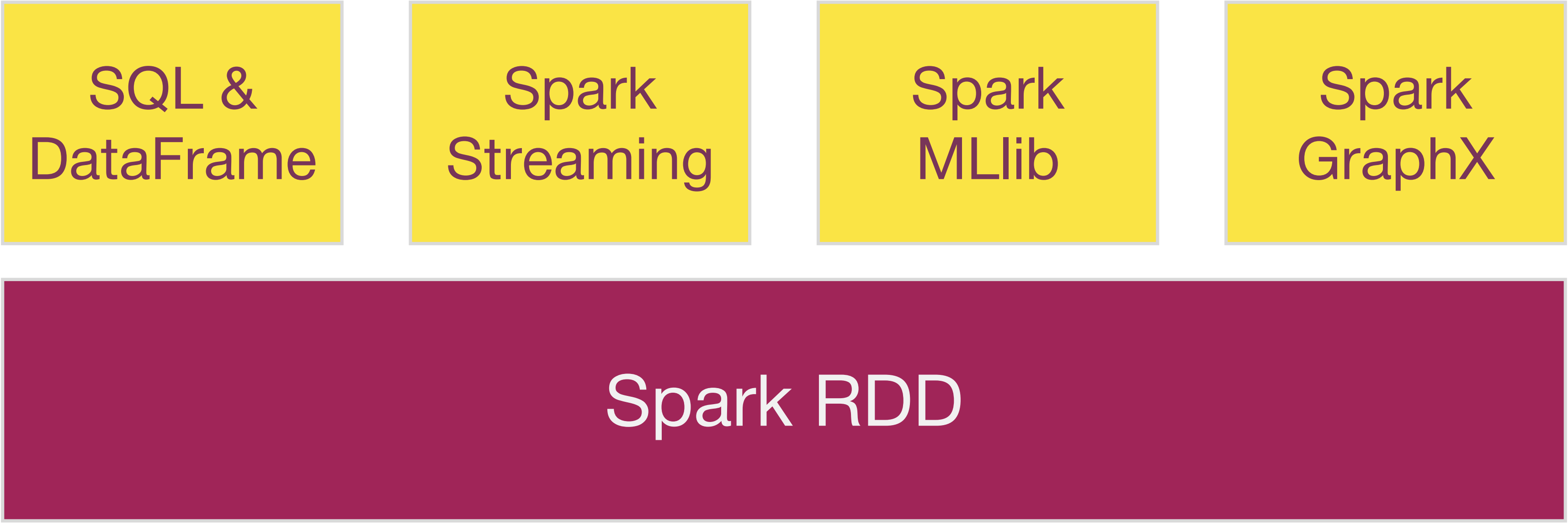
External Databases

Hive

Structured files

External Databases

Spark 1.0



SQL & DataFrame

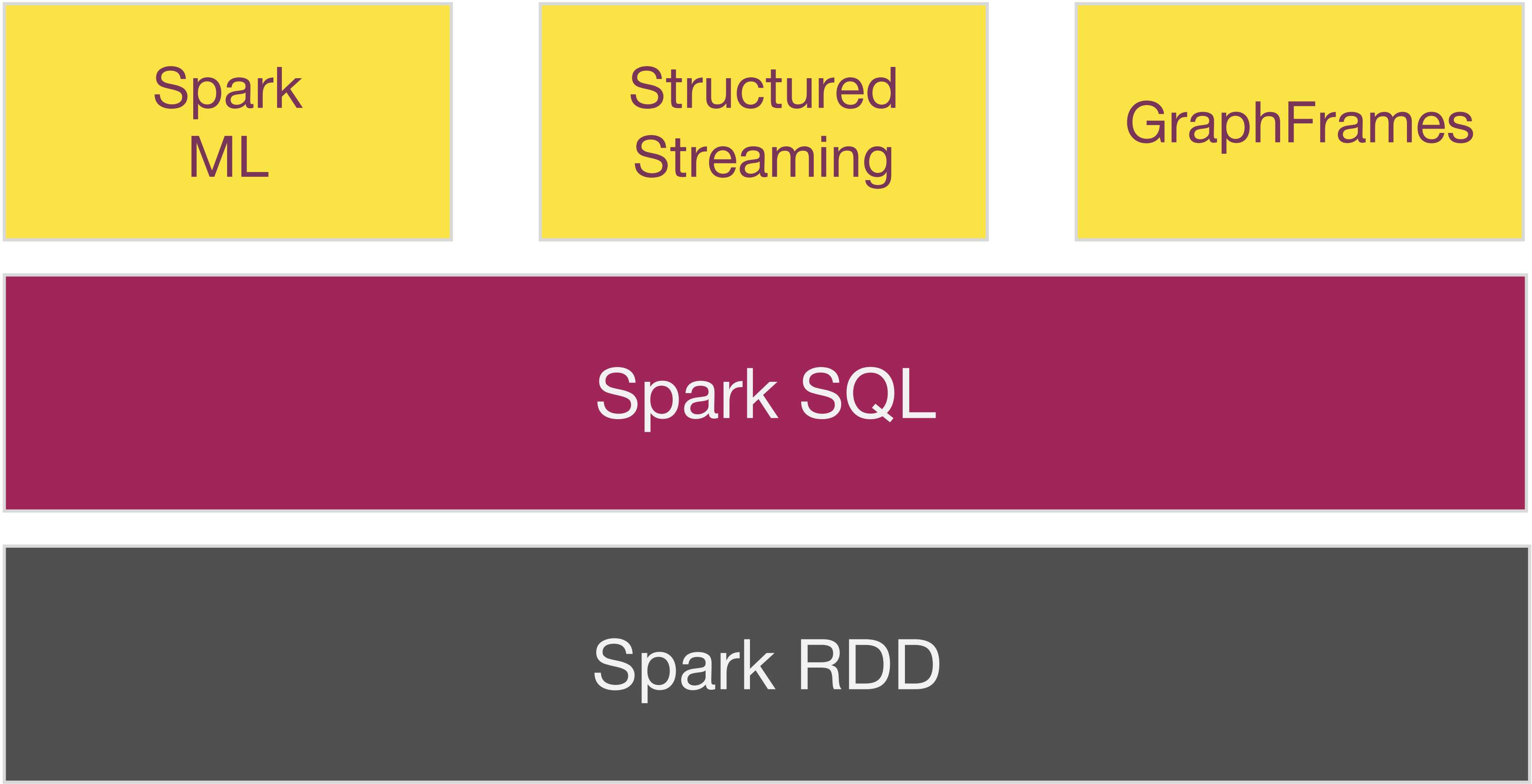
Spark Streaming

Spark MLlib

Spark GraphX

Spark RDD

Spark 2.0



Spark ML

Structured Streaming

GraphFrames

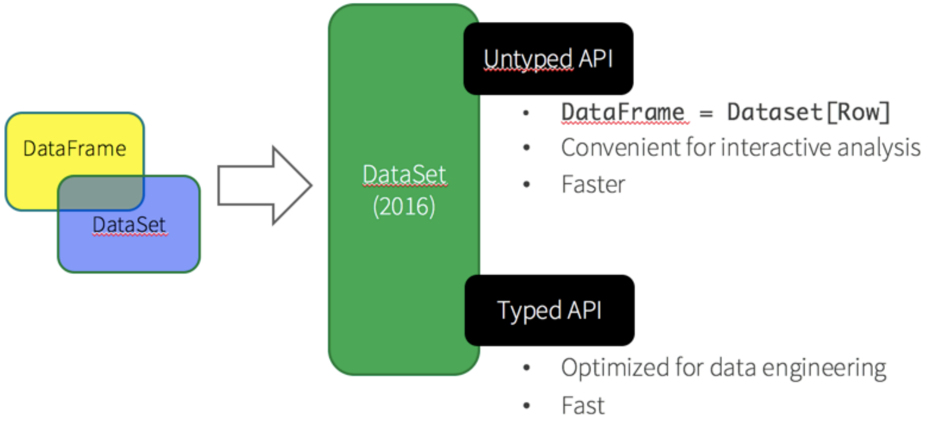
Spark SQL

Spark RDD

SparkSQL

- programming abstractions called DataFrame that can act as a distributed SQL query engine

- The input data can be queried by using
  - ad-hoc methods
  - an SQL-like language



DataFrame

DataSet

DataSet (2016)

Untyped API

- `DataFrame = Dataset[Row]`
- Convenient for interactive analysis
- Faster

Typed API

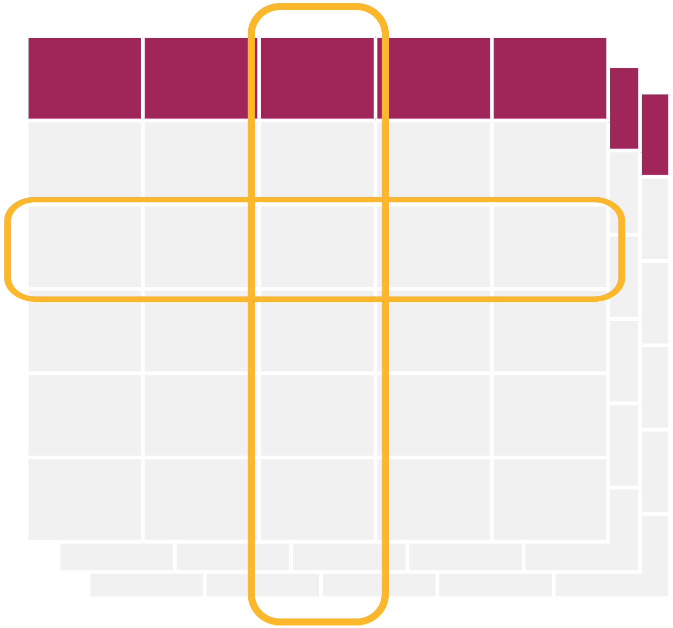
- Optimized for data engineering
- Fast

Structured API

- The interfaces provide more information about the structure of both the data and the computation being performed
- Spark uses this extra information to perform extra optimizations based on an “SQL-like” optimizer called Catalyst
  - Compute the best execution plan before executing the code
    => Programs are usually faster than standard RDD-based programs

DataFrames

DataFrames = RDD+Schema



DataFrame = RDD + schema

DataFrame

noun – [dey-tuh-freym]

1. A distributed collection of rows organized into named columns.
2. An abstraction for selecting, filtering, aggregating and plotting structured data (*cf. R, Pandas*).
3. Archaic: Previously SchemaRDD (*cf. Spark < 1.3*).

DataFrames = RDD+Schema

```
In: from pyspark.sql import SparkSession
    spark_session = SparkSession.builder\
                                .enableHiveSupport()\
                                .appName("spark sql")\
                                .master("local")\
                                .getOrCreate()
```

```
In: geoip_rdd = spark_session\
                   .sparkContext\
                   .textFile("     …           '/geoip")
```

```
In: geoip_rdd.take(3)
```

```
Out: [u'194.120.126.123, NL, Netherlands',
      u'94.126.119.173, FR, France',
      u'193.46.74.166, RU, Russian Federation']
```

DataFrames = RDD+Schema

ip **STRING**,
code **STRING**,
country **STRING**

```
In: from pyspark.sql.types import *
    schema = StructType().add("ip",      StringType())\
                         .add("code",    StringType())\
                         .add("country", StringType())
```

```
In: geoip_df = spark_session\
                  .createDataFrame(geoip_rdd1, schema)
```

```
In: geoip_df
```

```
Out: DataFrame[ip: string, code: string, country: string]
```

DataFrames = RDD+Schema

```
In: geoip_df.show(3)
```

```
+---------------+----+------------------+
|             ip|code|           country|
+---------------+----+------------------+
|194.120.126.123|  NL|       Netherlands|
| 94.126.119.173|  FR|            France|
|  193.46.74.166|  RU|Russian Federation|
+---------------+----+------------------+
only showing top 3 rows
```

DataFrames = RDD+Schema

```
 In: geoip_df.rdd.take(3)

Out: [Row(ip=u'194.120.126.123', code=u'NL',
     country=u'Netherlands'),
      Row(ip=u'94.126.119.173', code=u'FR',
     country=u'France'),
      Row(ip=u'193.46.74.166', code=u'RU',
     country=u'Russian Federation')]
```

DataFrames = RDD+Schema

```
In: geoip_df.printSchema()

    root
     |-- ip: string (nullable = true)
     |-- code: string (nullable = true)
     |-- country: string (nullable = true)
```

```
In: geoip_df
```

```
Out: DataFrame[ip: string, code: string, country: string]
```

show

```
In: geoip_df.show(3)
```

```
+---------------+----+------------------+
|             ip|code|           country|
+---------------+----+------------------+
|194.120.126.123|  NL|       Netherlands|
| 94.126.119.173|  FR|            France|
|  193.46.74.166|  RU|Russian Federation|
+---------------+----+------------------+
only showing top 3 rows
```

$$Projection$$
$$\pi$$

select

```
In: geoip_df.select("country","ip")\
            .show(3)
```

```
+-------------------+---------------+
|            country|             ip|
+-------------------+---------------+
|        Netherlands|194.120.126.123|
|             France| 94.126.119.173|
|Russian Federation|  193.46.74.166|
+-------------------+---------------+
only showing top 3 rows
```

where

Filtering (selection) σ

```
In: geoip_df\
        .select("country","ip")\
        .where("country = 'Russian Federation'")\
        .show(3)
```

```
+------------------+---------------+
|           country|             ip|
+------------------+---------------+
|Russian Federation| 193.46.74.166|
|Russian Federation| 46.235.67.202|
|Russian Federation|193.161.193.64|
+------------------+---------------+
only showing top 3 rows
```

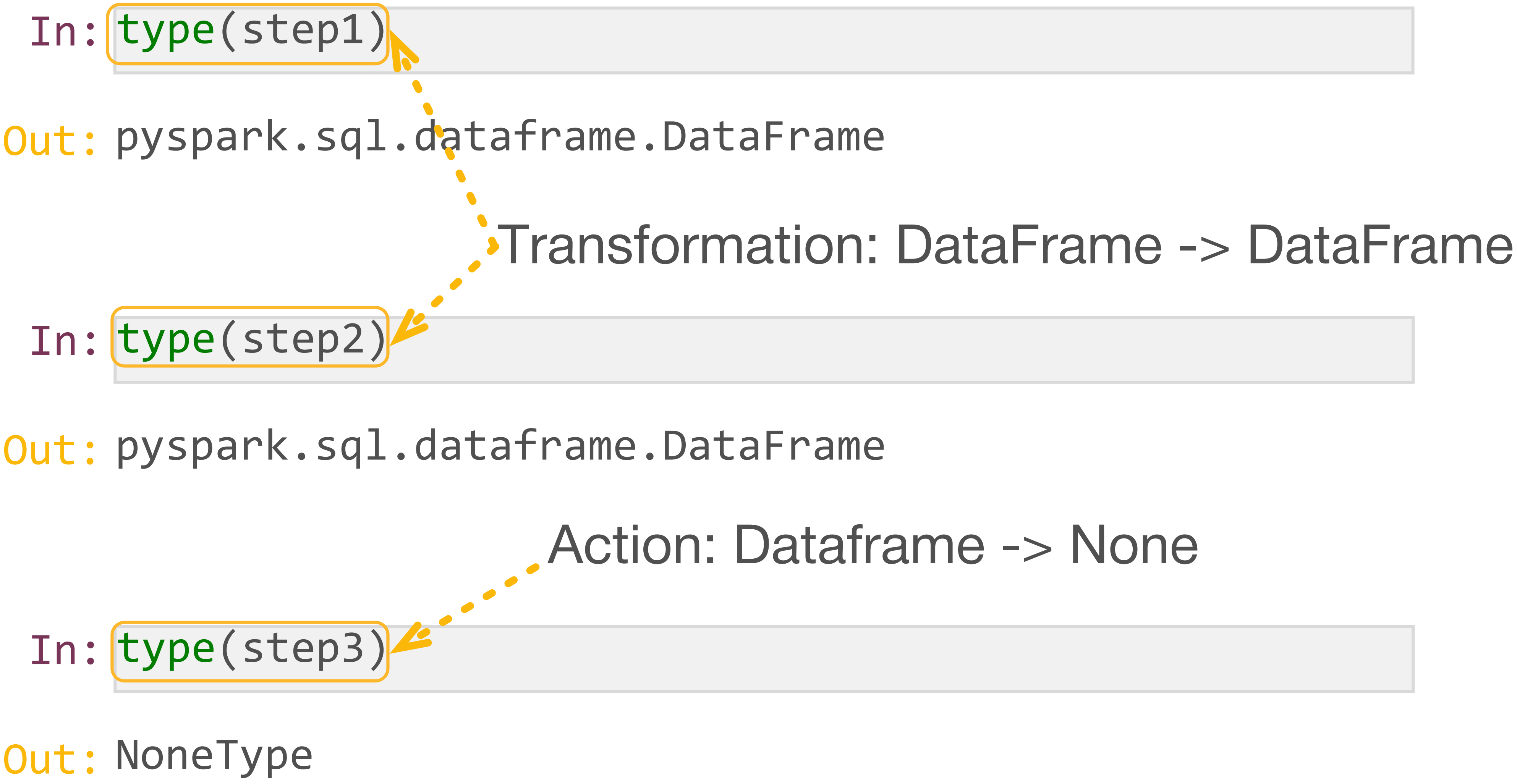
```
In: step1 = geoip_df.select("country","ip")
```

```
In: step2 = step1.where("country = 'Russian Federation'")
```

```
In: step3 = step2.show(3)
```

```
+------------------+--------------+
|           country|            ip|
+------------------+--------------+
|Russian Federation| 193.46.74.166|
|Russian Federation| 46.235.67.202|
|Russian Federation|193.161.193.64|
+------------------+--------------+
only showing top 3 rows
```

Spark execution model: tranformations, actions, lazy evaluation



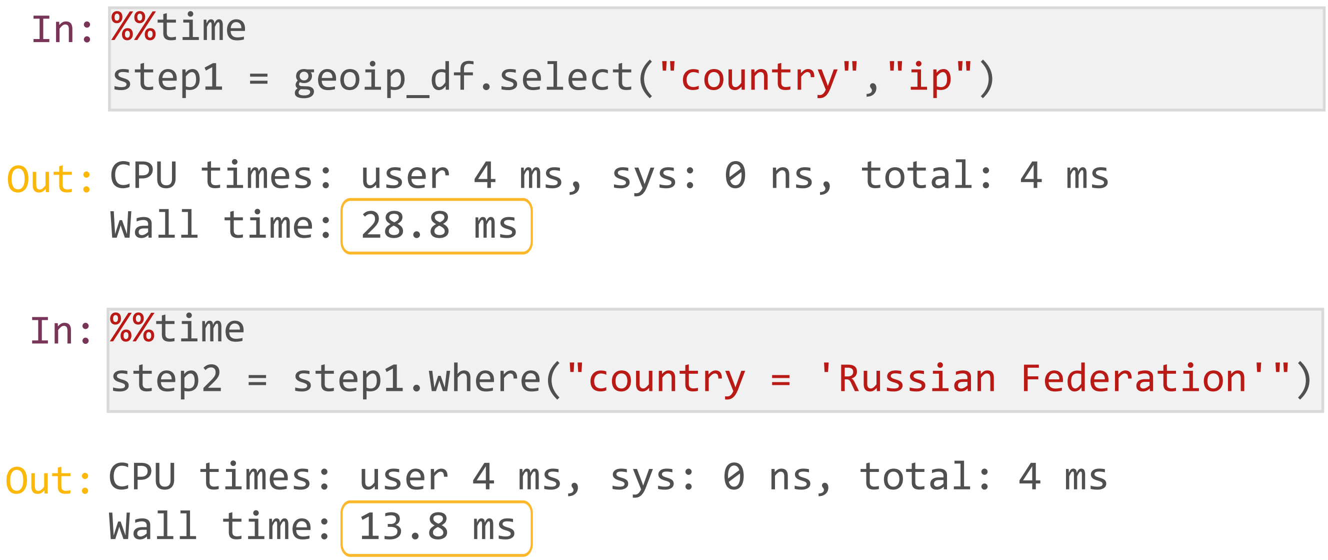
```
In:  type(step1)

Out: pyspark.sql.dataframe.DataFrame
```

Transformation: DataFrame -> DataFrame

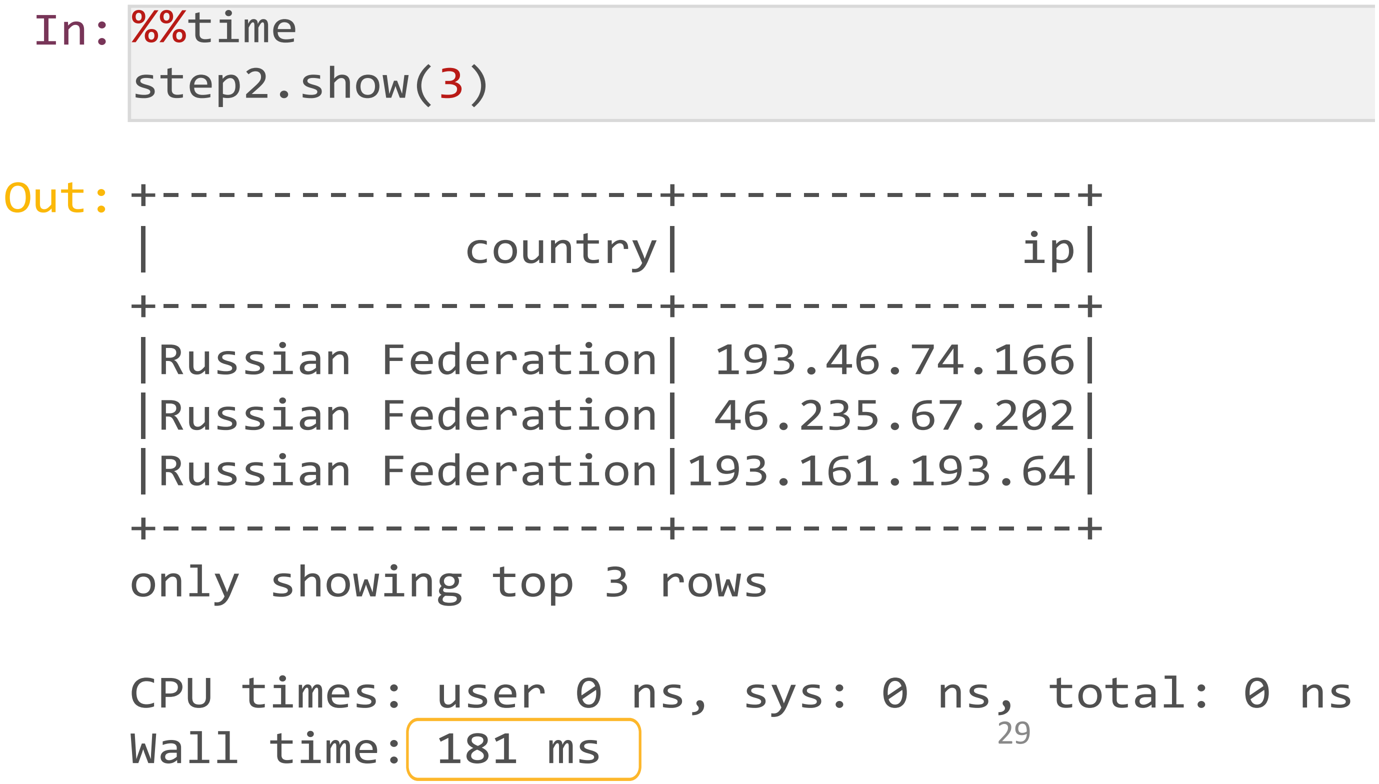
```
In:  type(step2)

Out: pyspark.sql.dataframe.DataFrame
```

Action: Dataframe -> None

```
In:  type(step3)

Out: NoneType
```



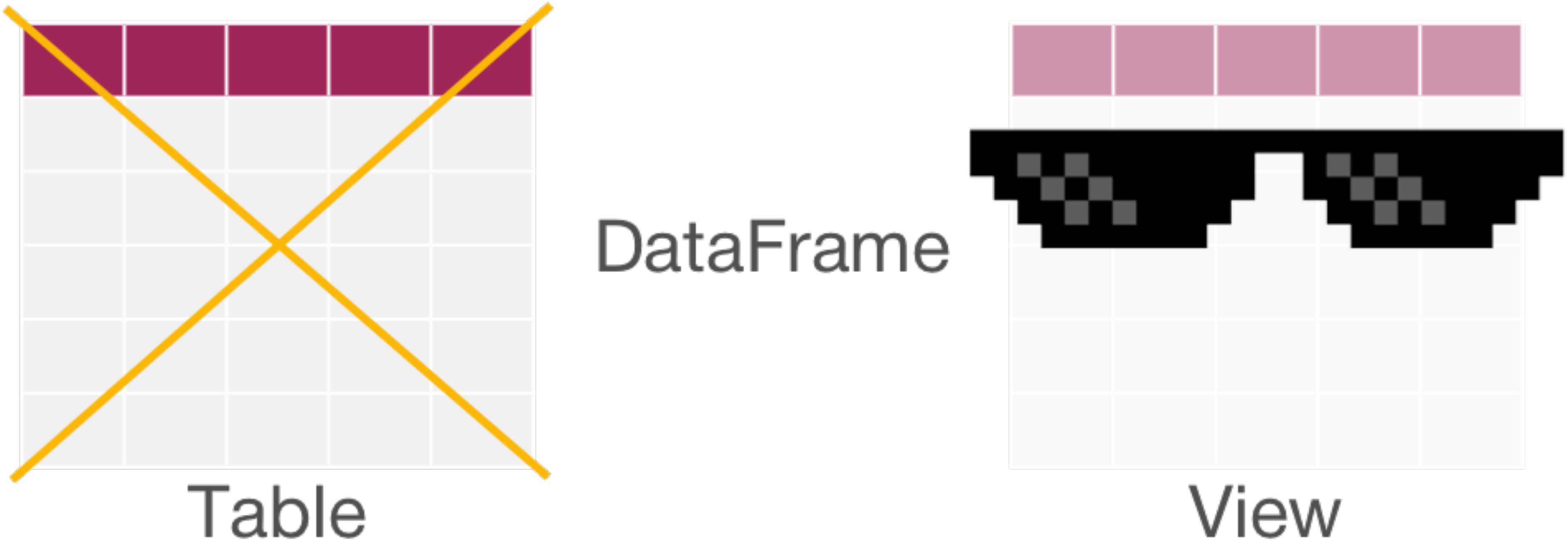
```
In: %%time
    step1 = geoip_df.select("country","ip")

Out: CPU times: user 4 ms, sys: 0 ns, total: 4 ms
     Wall time: 28.8 ms

In: %%time
    step2 = step1.where("country = 'Russian Federation'")

Out: CPU times: user 4 ms, sys: 0 ns, total: 4 ms
     Wall time: 13.8 ms
```

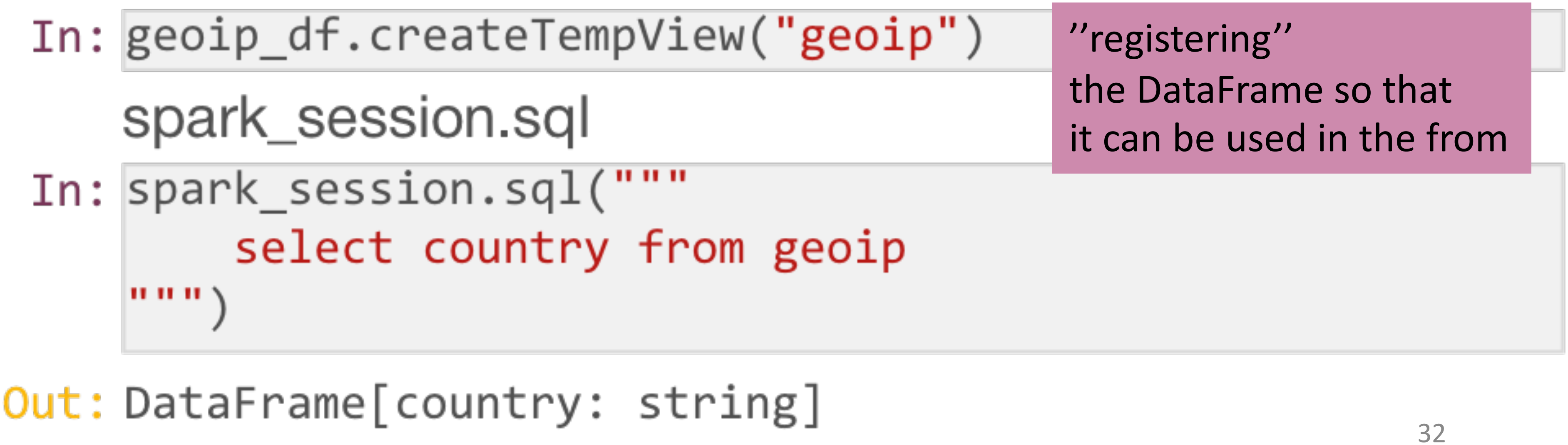


```
In: %%time
    step2.show(3)

Out: +------------------+---------------+
     |           country|             ip|
     +------------------+---------------+
     |Russian Federation| 193.46.74.166|
     |Russian Federation| 46.235.67.202|
     |Russian Federation|193.161.193.64|
     +------------------+---------------+
     only showing top 3 rows

     CPU times: user 0 ns, sys: 0 ns, total: 0 ns
     Wall time: 181 ms
```

DataFrames vs SQL
Which Spark-API to use?

DataFrames and SQL





```
In: geoip_df.createTempView("geoip")
```

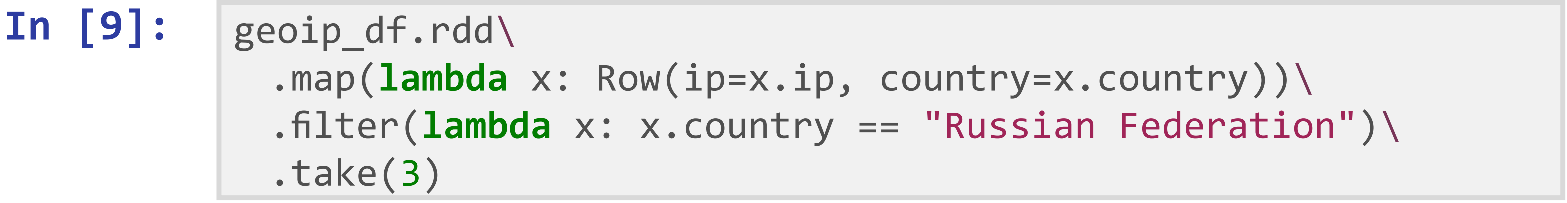
”registering”
the DataFrame so that
it can be used in the from

spark_session.sql

```
In: spark_session.sql("""
        select country from geoip
    """)

Out: DataFrame[country: string]
```

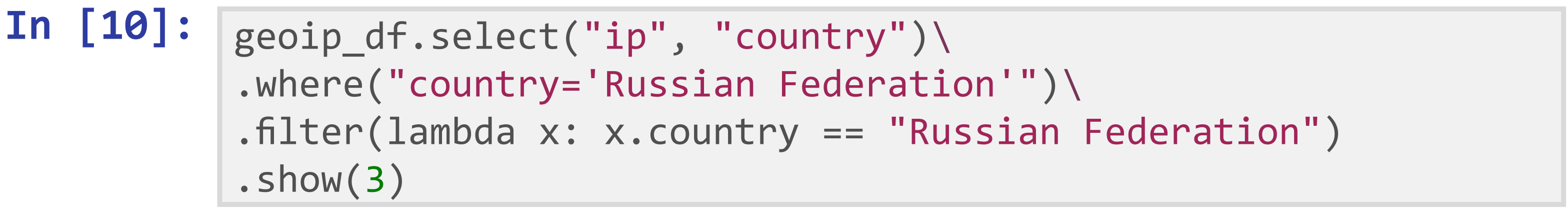
Which Spark API to use?

RDD API



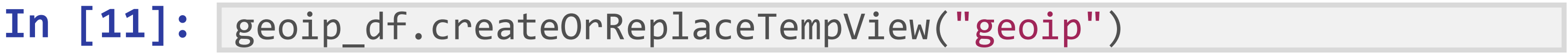
```
In [9]:  geoip_df.rdd\
            .map(lambda x: Row(ip=x.ip, country=x.country))\
            .filter(lambda x: x.country == "Russian Federation")\
            .take(3)
```

DataFrame API

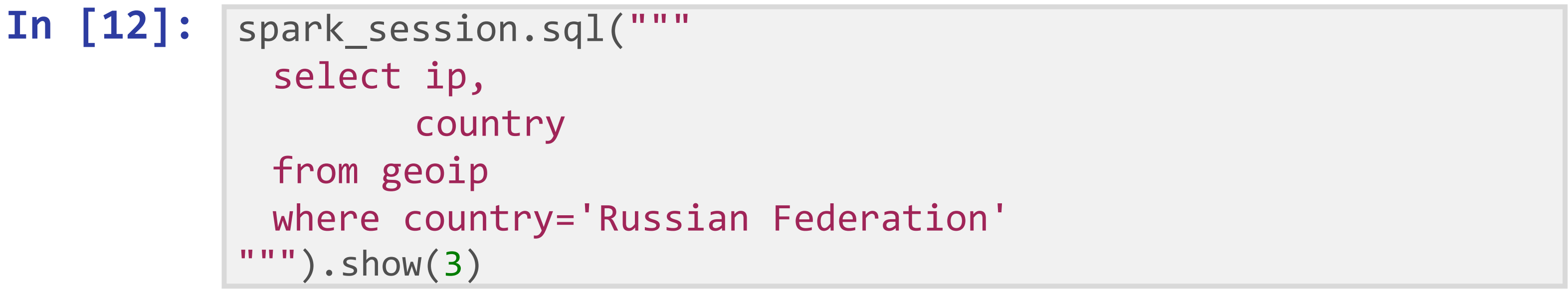


```
In [10]: geoip_df.select("ip", "country")\
         .where("country='Russian Federation'")\
         .filter(lambda x: x.country == "Russian Federation")
         .show(3)
```

SQL

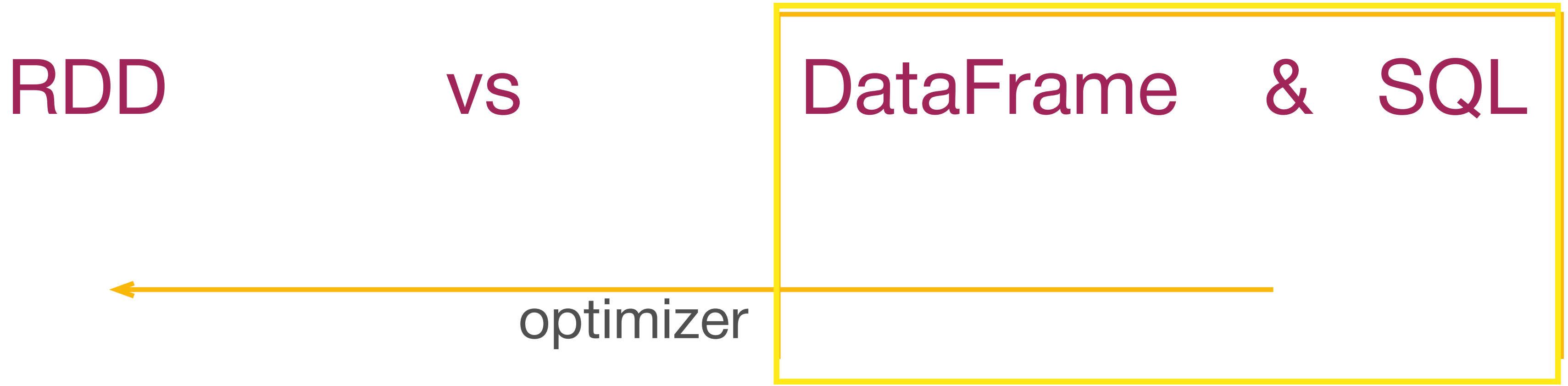


```
In [11]: geoip_df.createOrReplaceTempView("geoip")
```



```
In [12]: spark_session.sql("""
           select ip,
                  country
           from geoip
           where country='Russian Federation'
         """).show(3)
```

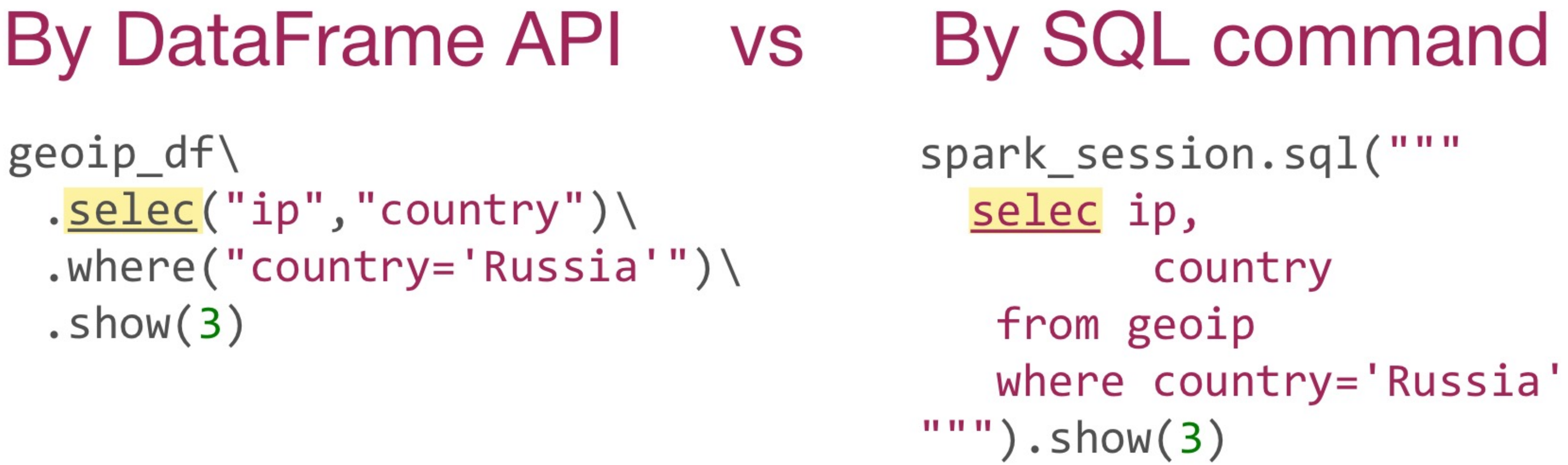
Which Spark API to use?



RDD vs DataFrame & SQL

optimizer

Which Spark API to use?



By DataFrame API vs By SQL command

```
geoip_df\
  .selec("ip","country")\
  .where("country='Russia'")\
  .show(3)
```

```
spark_session.sql("""
  selec ip,
          country
    from geoip
    where country='Russia'
""").show(3)
```

Error will be found at compilation

Error will be found at query call

Creating Data Frames

Creating DataFrames

- DataFrames can be created
  - from CSV (with options, e.g. inferschema, header)
  - from text file (standard «multiline» JSON)
  - from an RDD
    - Programmatically specifying the schema
    - Inferring the schema through reflection

Reading and Writing DataFrames

Built-In | External

{ JSON }
JDBC
Parquet
HIVE
MySQL
PostgreSQL
HDFS
amazon web services S3
H2
AVRO
CSV
dBase
APACHE HBASE
elasticsearch.
cassandra
amazon web services
Amazon Redshift

and more…

Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
```

Write - Formats

```
In: geoip_df.write.save("geoip_csv",
                        format='csv')
```

```
In: spark_session\
        .sparkContext\
        .textFile("geoip_csv")\
        .take(3)
```

```
Out: [u'194.120.126.123,NL,Netherlands',
      u'94.126.119.173,FR,France',
      u'193.46.74.166,RU,Russian Federation']
```

Write - Formats

```
In: geoip_df.write.save("geoip_json",
                        format='json')
```

```
In: spark_session\
        .sparkContext\
        .textFile("geoip_json")\
        .take(3)
```

```
Out: [u'{"ip":"194.120.126.123","code":"NL","country":"Net
     herlands"}',
      u'{"ip":"94.126.119.173","code":"FR","country":"Fran
     ce"}',
      u'{"ip":"193.46.74.166","code":"RU","country":"Russi
     an Federation"}']
```

Read

```
In: geoip_from_table = spark_session\
        .read.table("web.geoip")
```

```
In: geoip_from_table.show(3)
```

```
+---------------+----+------------------+
|             ip|code|           country|
+---------------+----+------------------+
|194.120.126.123|  NL|       Netherlands|
| 94.126.119.173|  FR|            France|
|  193.46.74.166|  RU|Russian Federation|
+---------------+----+------------------+
only showing top 3 rows
```

Read

```
In: geoip_from_json = spark_session\
        .read.json("geoip_json")
```

```
In: geoip_from_json.show(3)
```

```
+----+------------------+---------------+
|code|           country|             ip|
+----+------------------+---------------+
|  NL|       Netherlands|194.120.126.123|
|  FR|            France| 94.126.119.173|
|  RU|Russian Federation|  193.46.74.166|
+----+------------------+---------------+
only showing top 3 rows
```

Read

```
In: geoip_from_csv = spark_session\
        .read.csv("geoip_csv")
```

```
In: geoip_from_csv.show(3)
```

```
+--------------+---+------------------+
|           _c0|_c1|               _c2|
+--------------+---+------------------+
|   217.8.92.38| RU|Russian Federation|
|185.102.10.199| RU|Russian Federation|
|  217.73.57.80| RU|Russian Federation|
+--------------+---+------------------+
only showing top 3 rows
```

Read – Adding column names

```
In: schema = StructType().add("ip",      StringType())\
                         .add("code",    StringType())\
                         .add("country", StringType())
```

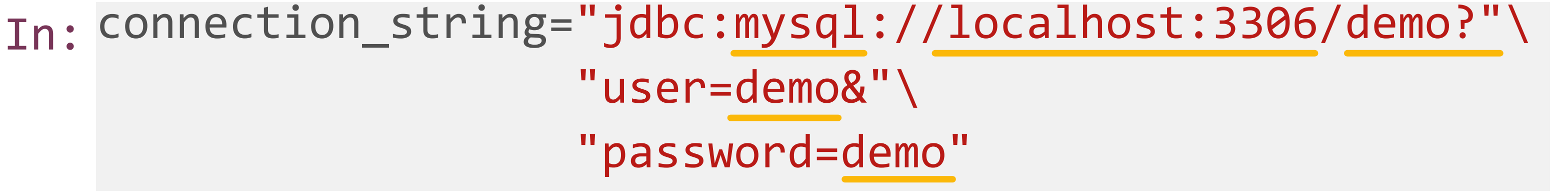
```
In: geoip_from_csv = spark_session\
        .read.csv("geoip_csv")
```

```
In: geoip_from_csv.show(3)
```

```
+--------------+----+------------------+
|            ip|code|           country|
+--------------+----+------------------+
|   217.8.92.38|  RU|Russian Federation|
|185.102.10.199|  RU|Russian Federation|
|  217.73.57.80|  RU|Russian Federation|
+--------------+----+------------------+
only showing top 3 rows
```

JDBC
Import and export data to DBMSs

JDBC - Java DataBase Connectivity



```
In: connection_string="jdbc:mysql://localhost:3306/demo?"\
                      "user=demo&"\
                      "password=demo"
```

```
In: geoip_from_jdbc = spark_session\
        .read.jdbc(connection_string, "geoip")
```

```
In: geoip_from_jdbc.show(3)
```

```
+---------------+----+------------------+
|             ip|code|           country|
+---------------+----+------------------+
|194.120.126.123|  NL|       Netherlands|
| 94.126.119.173|  FR|            France|
|  193.46.74.166|  RU|Russian Federation|
+---------------+----+------------------+
only showing top 3 rows
```

```
In: geoip_df.write.jdbc(connection_string, "geoip")
```

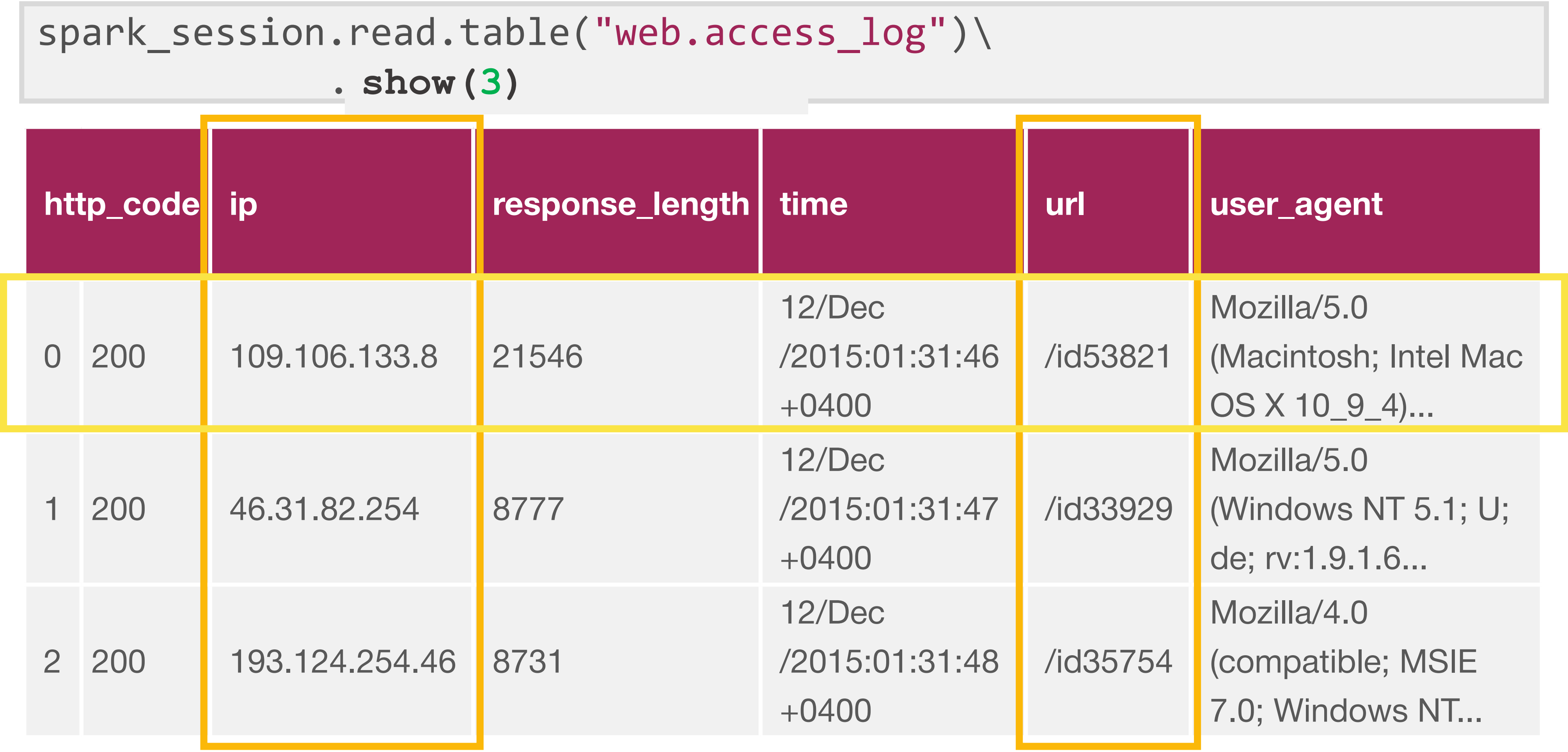
```
In: geoip_from_jdbc = spark_session\
        .read.jdbc(connection_string, "geoip")
```

```
In: geoip_from_jdbc.show(3)
```

```
+---------------+----+------------------+
|             ip|code|           country|
+---------------+----+------------------+
|194.120.126.123|  NL|       Netherlands|
| 94.126.119.173|  FR|            France|
|  193.46.74.166|  RU|Russian Federation|
+---------------+----+------------------+
only showing top 3 rows
```

Data Frames Operations

Projection & Filtering



```
spark_session.read.table("web.access_log")\
                  . show(3)
```

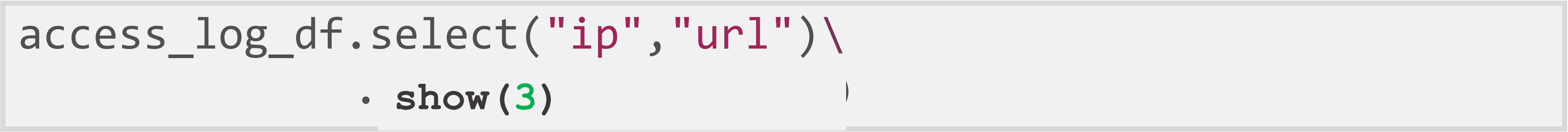
	http_code	ip	response_length	time	url	user_agent
0	200	109.106.133.8	21546	12/Dec /2015:01:31:46 +0400	/id53821	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...
1	200	46.31.82.254	8777	12/Dec /2015:01:31:47 +0400	/id33929	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...
2	200	193.124.254.46	8731	12/Dec /2015:01:31:48 +0400	/id35754	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...

Projection - SQL

```
spark_session.sql("""
     select ip, url
     from web.access_log
""").show(3)
```

	ip	url
0	109.106.133.8	/id53821
1	46.31.82.254	/id33929
2	193.124.254.46	/id35754

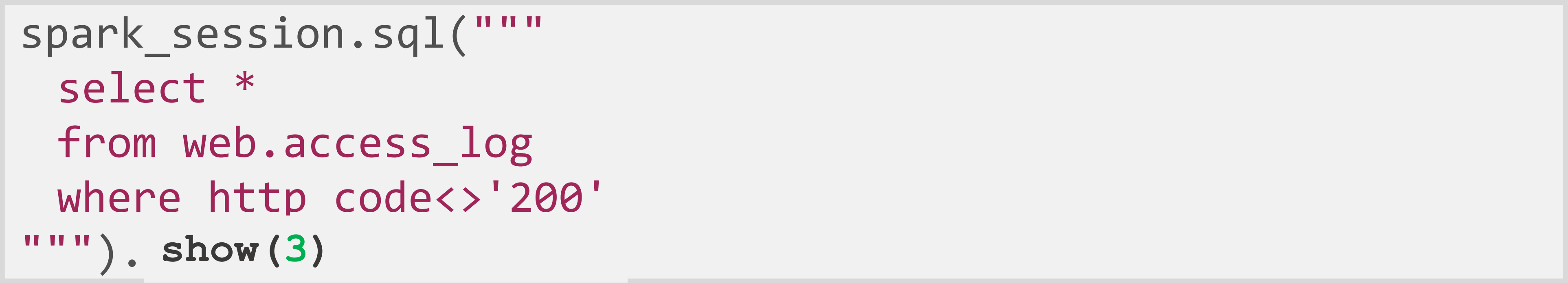
Projection - Dataframes



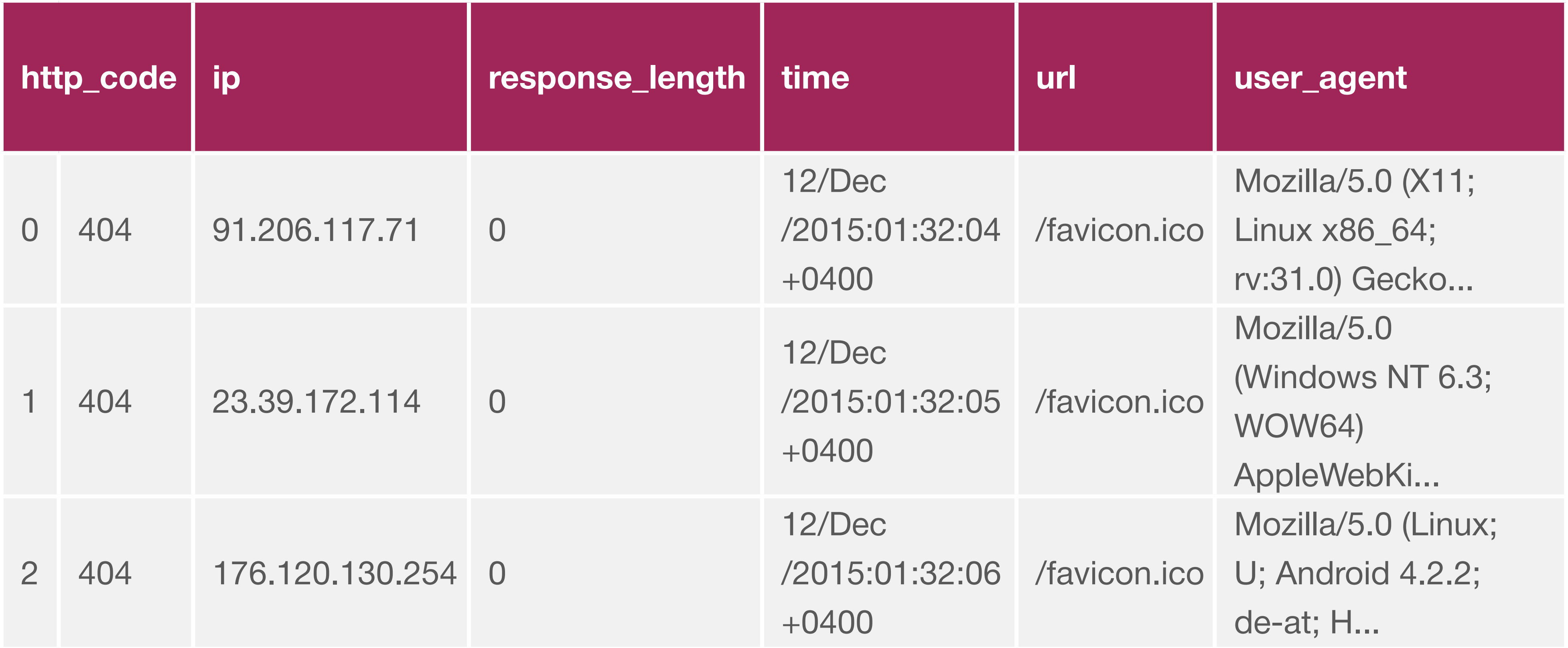
```
access_log_df.select("ip","url")\
                  .show(3)
```

	ip	url
0	109.106.133.8	/id53821
1	46.31.82.254	/id33929
2	193.124.254.46	/id35754

Selection - SQL



```
spark_session.sql("""
  select *
  from web.access_log
  where http code<>'200'
""").show(3)
```

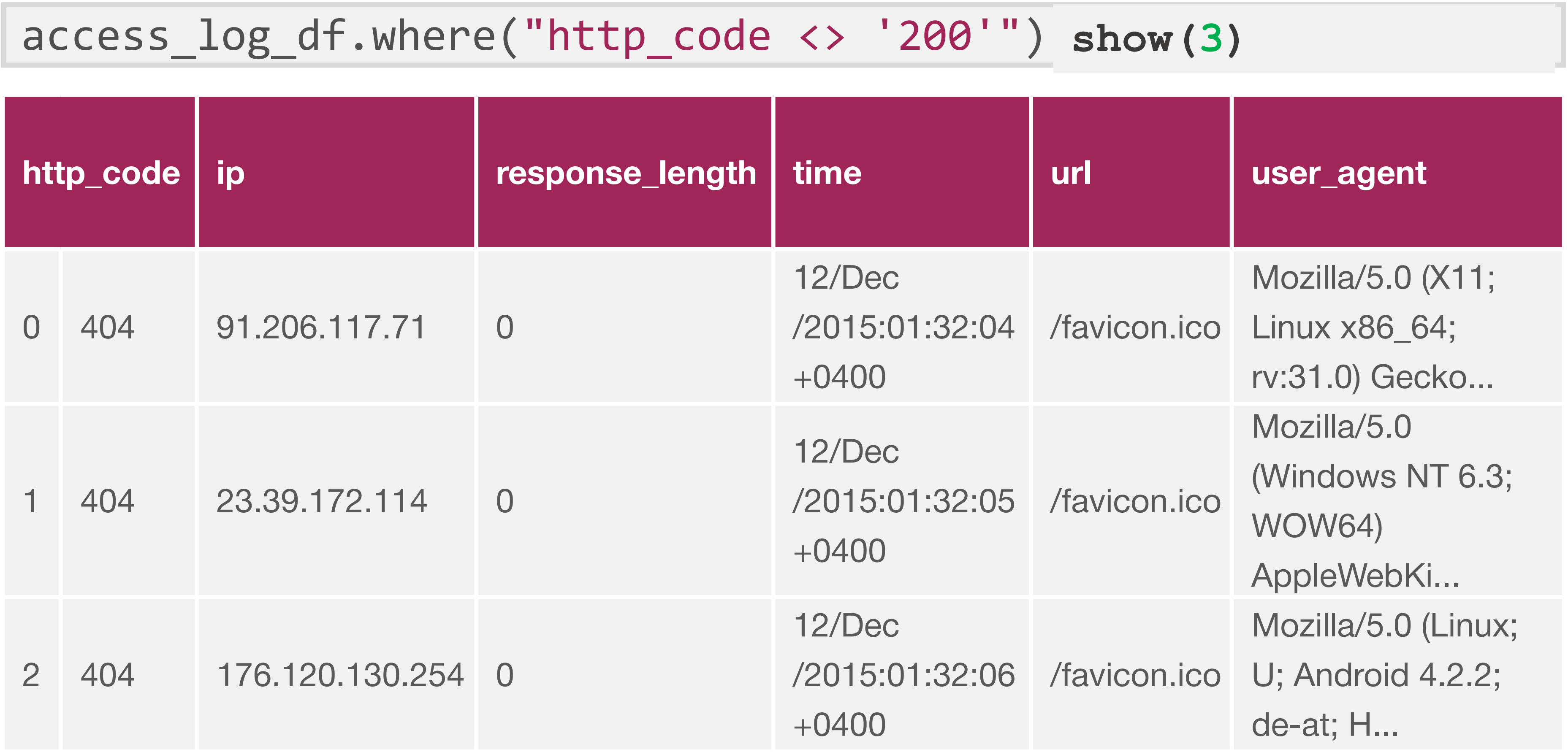


	http_code	ip	response_length	time	url	user_agent
0	404	91.206.117.71	0	12/Dec /2015:01:32:04 +0400	/favicon.ico	Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko...
1	404	23.39.172.114	0	12/Dec /2015:01:32:05 +0400	/favicon.ico	Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKi...
2	404	176.120.130.254	0	12/Dec /2015:01:32:06 +0400	/favicon.ico	Mozilla/5.0 (Linux; U; Android 4.2.2; de-at; H...

Selection - Dataframes

Where

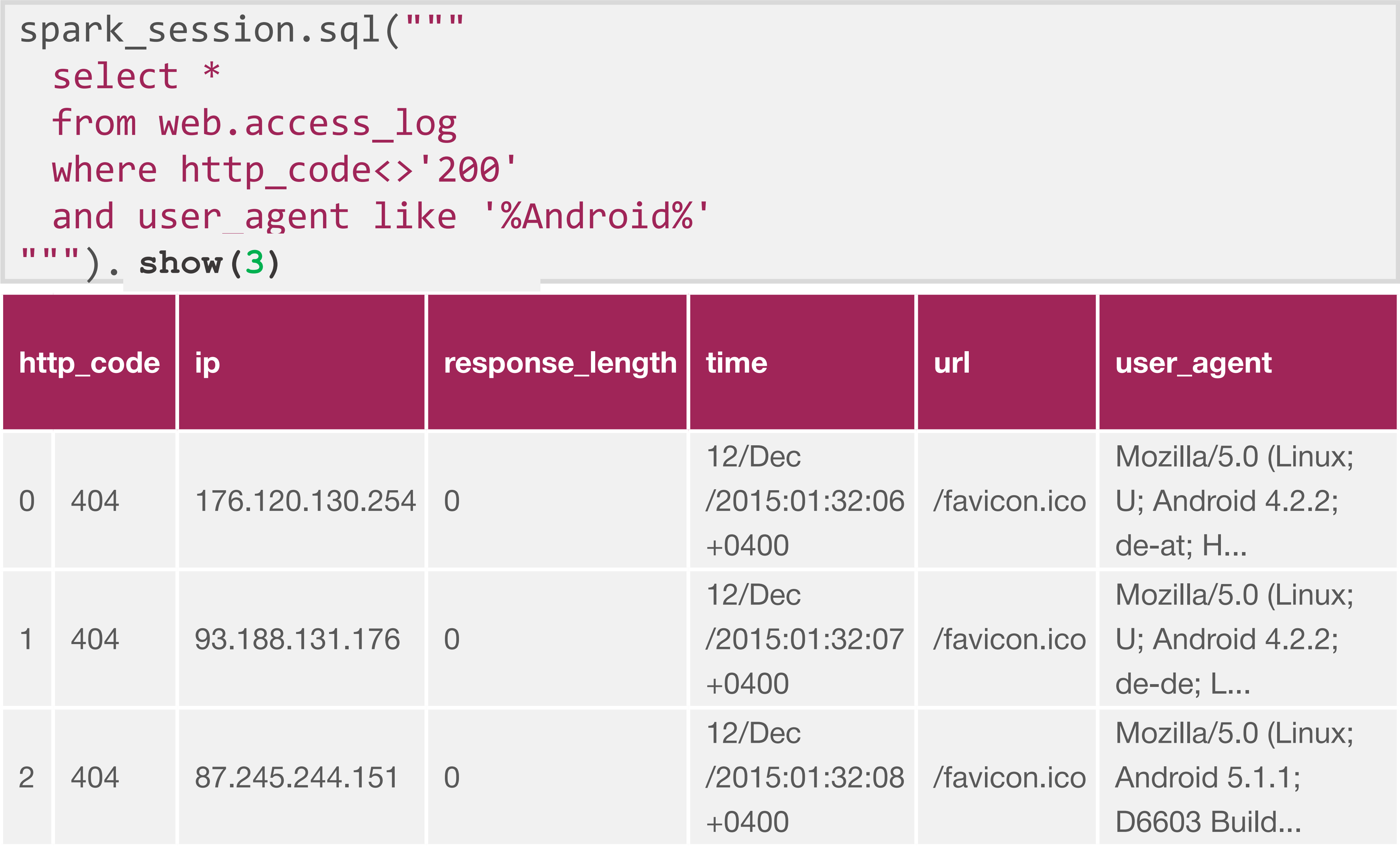
Filter



```
access_log_df.where("http_code <> '200'") show(3)
```

	http_code	ip	response_length	time	url	user_agent
0	404	91.206.117.71	0	12/Dec /2015:01:32:04 +0400	/favicon.ico	Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko...
1	404	23.39.172.114	0	12/Dec /2015:01:32:05 +0400	/favicon.ico	Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKi...
2	404	176.120.130.254	0	12/Dec /2015:01:32:06 +0400	/favicon.ico	Mozilla/5.0 (Linux; U; Android 4.2.2; de-at; H...

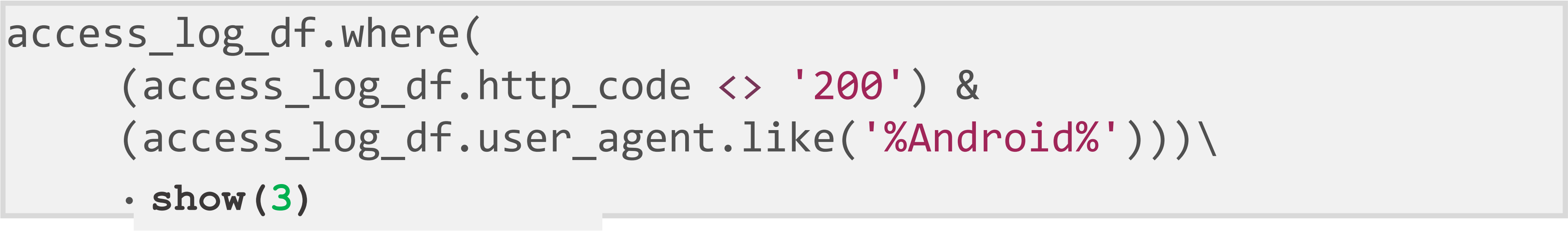
Selection – SQL (Composite Conditions)



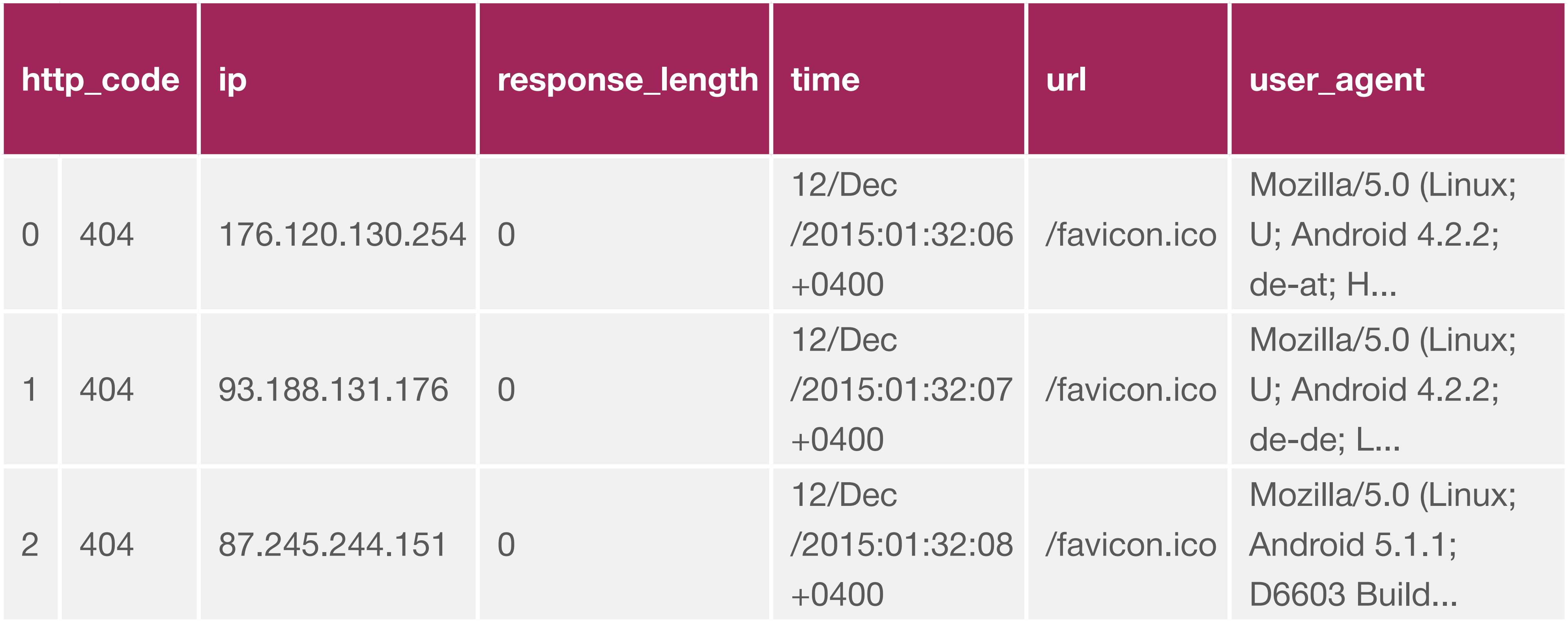
```
spark_session.sql("""
  select *
  from web.access_log
  where http_code<>'200'
  and user_agent like '%Android%'
""").show(3)
```

	http_code	ip	response_length	time	url	user_agent
0	404	176.120.130.254	0	12/Dec /2015:01:32:06 +0400	/favicon.ico	Mozilla/5.0 (Linux; U; Android 4.2.2; de-at; H...
1	404	93.188.131.176	0	12/Dec /2015:01:32:07 +0400	/favicon.ico	Mozilla/5.0 (Linux; U; Android 4.2.2; de-de; L...
2	404	87.245.244.151	0	12/Dec /2015:01:32:08 +0400	/favicon.ico	Mozilla/5.0 (Linux; Android 5.1.1; D6603 Build...

Selection – Dataframes (Composite Conditions)



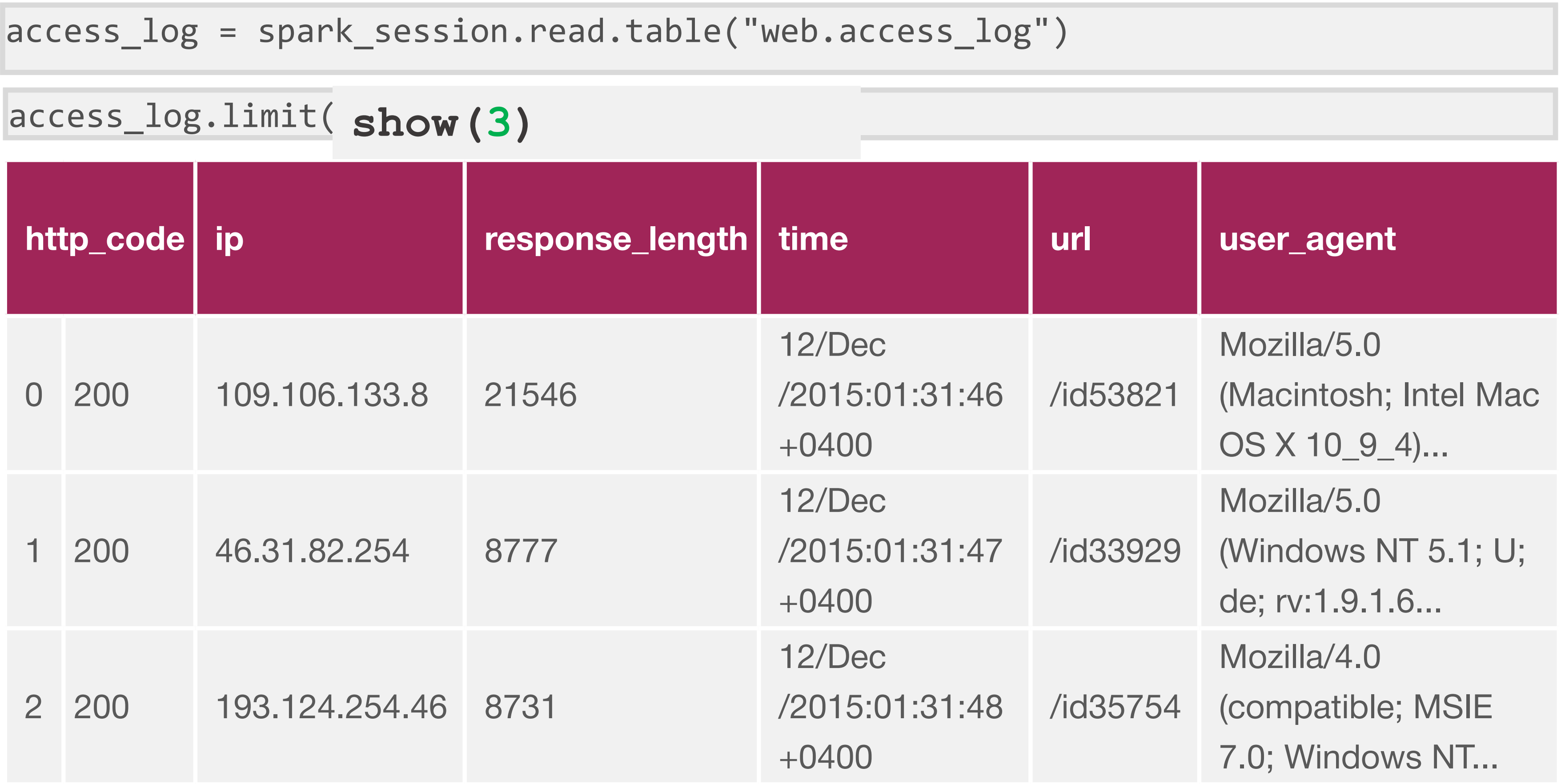
```
access_log_df.where(
     (access_log_df.http_code <> '200') &
     (access_log_df.user_agent.like('%Android%')))\
     .show(3)
```



	http_code	ip	response_length	time	url	user_agent
0	404	176.120.130.254	0	12/Dec /2015:01:32:06 +0400	/favicon.ico	Mozilla/5.0 (Linux; U; Android 4.2.2; de-at; H...
1	404	93.188.131.176	0	12/Dec /2015:01:32:07 +0400	/favicon.ico	Mozilla/5.0 (Linux; U; Android 4.2.2; de-de; L...
2	404	87.245.244.151	0	12/Dec /2015:01:32:08 +0400	/favicon.ico	Mozilla/5.0 (Linux; Android 5.1.1; D6603 Build...

Join

a dataframe
access_log



```
access_log = spark_session.read.table("web.access_log")
```

```
access_log.limit( show(3)
```

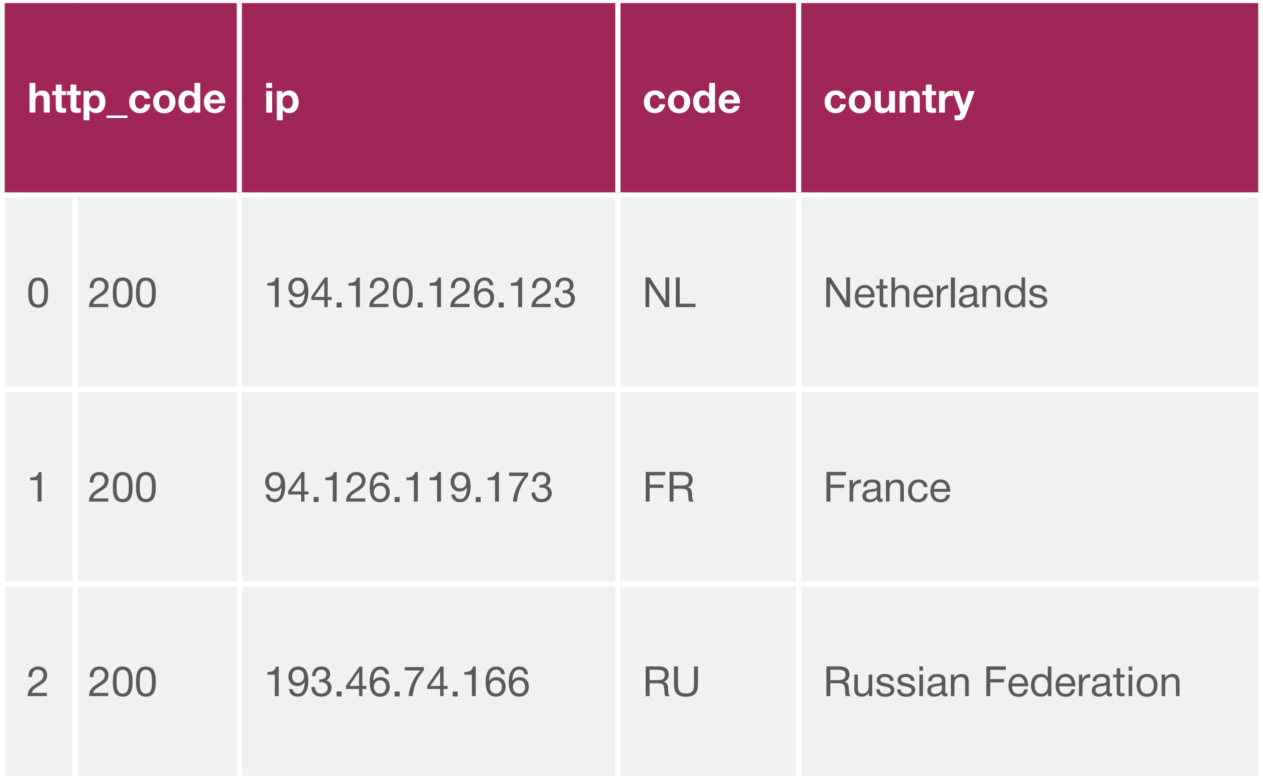
	http_code	ip	response_length	time	url	user_agent
0	200	109.106.133.8	21546	12/Dec /2015:01:31:46 +0400	/id53821	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...
1	200	46.31.82.254	8777	12/Dec /2015:01:31:47 +0400	/id33929	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...
2	200	193.124.254.46	8731	12/Dec /2015:01:31:48 +0400	/id35754	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...

Join

another dataframe geoip

```
geoip = spark_session.read.table("web.geoip")
```

```
geoip. show(3)
```



	http_code	ip	code	country
0	200	194.120.126.123	NL	Netherlands
1	200	94.126.119.173	FR	France
2	200	193.46.74.166	RU	Russian Federation

Join

```
spark_session.sql("""
select *
from web.access_log l
join web.geoip g
on l.ip = g.ip
""").show(3)
```

http_code		ip	response_length	time	url	user_agent	code	country
0	200	109.106.133.8	21546	12/Dec /2015:01:31:46 +0400	/id53821	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	RU	Russian Federation
1	200	46.31.82.254	8777	12/Dec /2015:01:31:47 +0400	/id33929	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...	GB	United Kingdom
2	200	193.124.254.46	8731	12/Dec /2015:01:31:48 +0400	/id35754	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...	RU	Russian Federation

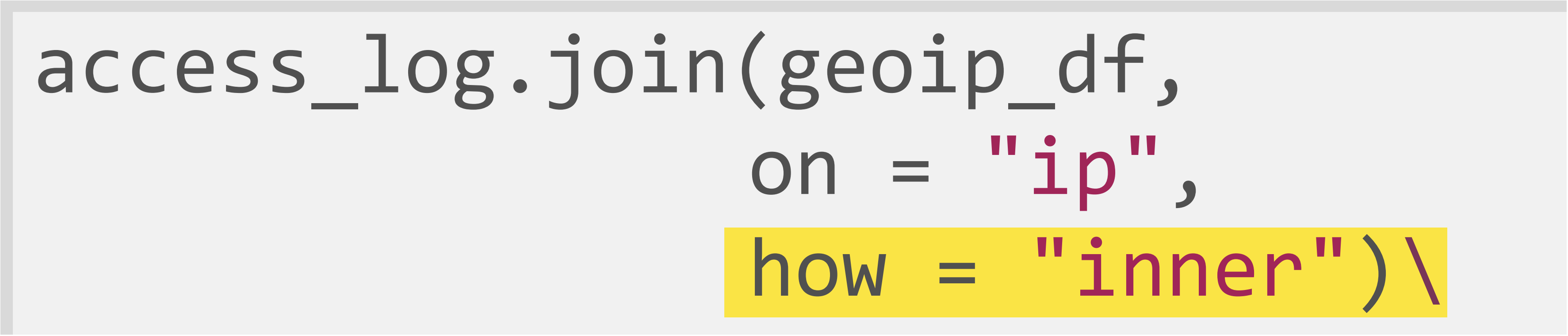
```
access_log.join(geoip_df, on="ip")\
          . show(3)
```

	http_code	ip	response_length	time	url	user_agent	code	country
0	200	109.106.133.8	21546	12/Dec /2015:01:31:46 +0400	/id53821	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	RU	Russian Federation
1	200	46.31.82.254	8777	12/Dec /2015:01:31:47 +0400	/id33929	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...	GB	United Kingdom
2	200	193.124.254.46	8731	12/Dec /2015:01:31:48 +0400	/id35754	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...	RU	Russian Federation

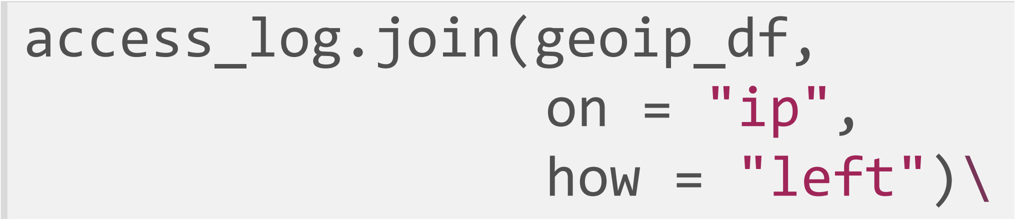
```
access_log.join(geoip_df,
                on = access_log.ip == geoip_df.ip)\
          . show(3)
```

http_code		ip	response_length	time	url	user_agent	code	country
0	200	109.106.133.8	21546	12/Dec /2015:01:31:46 +0400	/id53821	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	RU	Russian Federation
1	200	46.31.82.254	8777	12/Dec /2015:01:31:47 +0400	/id33929	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...	GB	United Kingdom
2	200	193.124.254.46	8731	12/Dec /2015:01:31:48 +0400	/id35754	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...	RU	Russian Federation

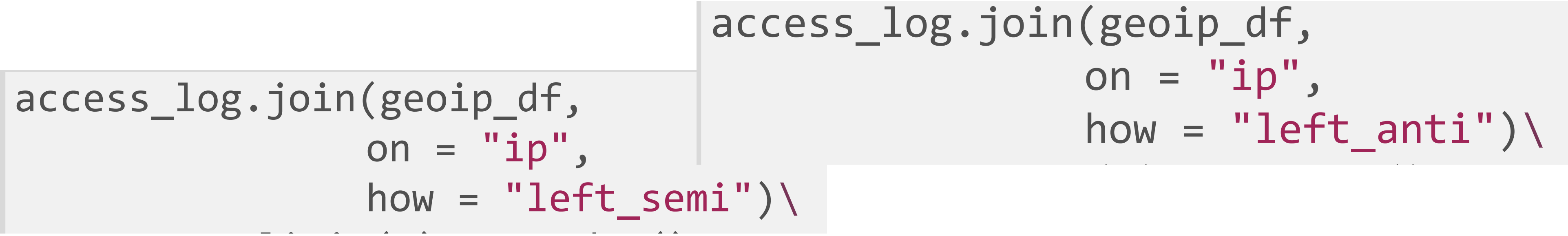
Join Types



```
access_log.join(geoip_df,
                on = "ip",
                how = "inner")\
```

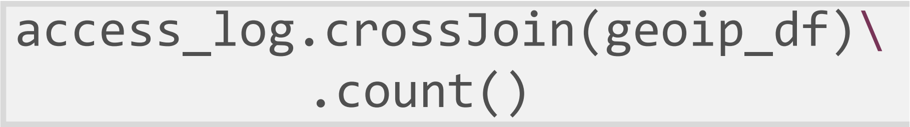


```
access_log.join(geoip_df,
                on = "ip",
                how = "left")\
```



```
access_log.join(geoip_df,
                on = "ip",
                how = "left_semi")\
```

```
access_log.join(geoip_df,
                on = "ip",
                how = "left_anti")\
```



```
access_log.crossJoin(geoip_df)\
          .count()
```

884031460

Join Types – Left join recall

```
SELECT * FROM web.geo_ip LEFT OUTER JOIN
web.access_log ON web.geo_ip.ip = web.accesslog.ip
```

geoip

ip	code	country
130.251.61.19	1	Italy
	null	null

access_log

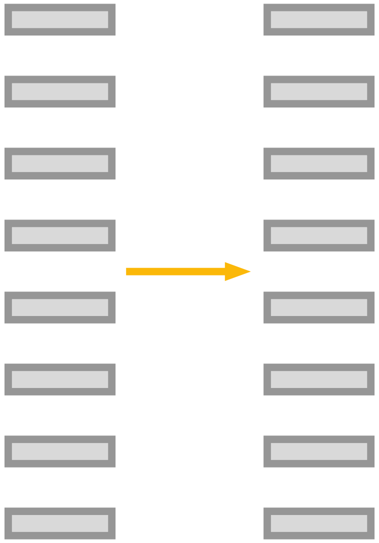
ip

Functions

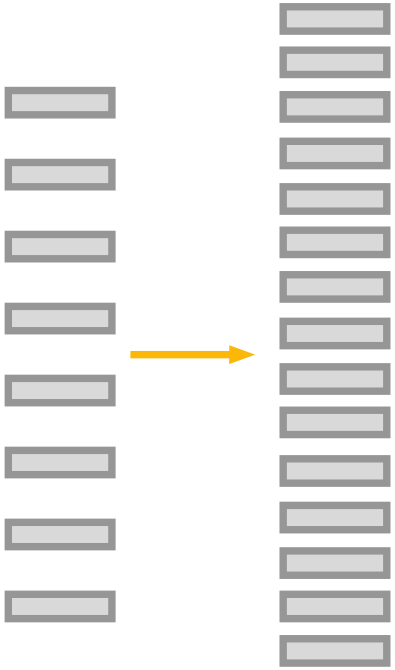
```
spark_session.sql("""
   select user_agent,
         length (user_agent)
   from web.access_log
   limit 3
   """).show()
```

	user_agent	length (user_agent)
0	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	120
1	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...	88
2	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...	153

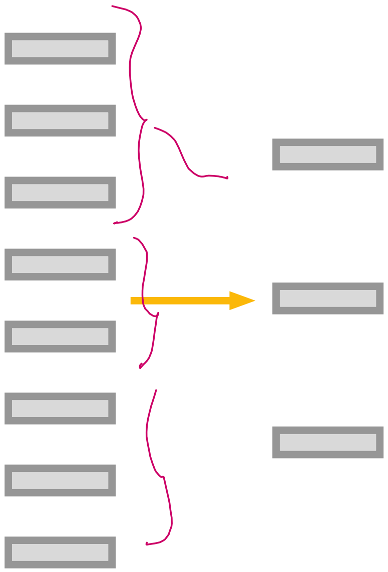
Functions



mapping



generating



aggregating

Mapping (aka scalar) functions

```
access_log.select("url",
                   f.concat(f.lit("http://vk.com"),
                            access_log_df.url)
                   ). show(5)
```

	url	concat (http://vk.com, url)
0	/id53821	http://vk.com/id53821
1	/id33929	http://vk.com/id33929
2	/id35754	http://vk.com/id35754
3	/id78231	http://vk.com/id78231
4	/id39395	http://vk.com/id39395

Aggregation

```
access_log = spark_session.read.table("web.access_log")
```

```
access_log.show(3)
```

	http_code	ip	response_length	time	url	user_agent
0	200	109.106.133.8	21546	12/Dec /2015:01:31:46 +0400	/id53821	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...
1	200	46.31.82.254	8777	12/Dec /2015:01:31:47 +0400	/id33929	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...
2	200	193.124.254.46	8731	12/Dec /2015:01:31:48 +0400	/id35754	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...

Aggregation

```
spark_session.sql("""
     select url,
            count(ip)
     from web.access_log
     group by url
""").show(4)
```

	url	count(ip)
0	/id53821	3
1	/id33929	2
2	/id35754	1
3	/id11744	1

Aggregation

```
import pyspark.sql.functions as f
```

```
access_log.groupBy("url")\
          .agg(f.count("ip"))\
          .show(3)
```

	url	count(DISTINCT ip)
0	/id37020	4
1	/id47695	2
2	/id77559	1

Aggregation

```
access_log.groupBy("url")\
                .agg({"ip":"count"})\
                . show(3)
```

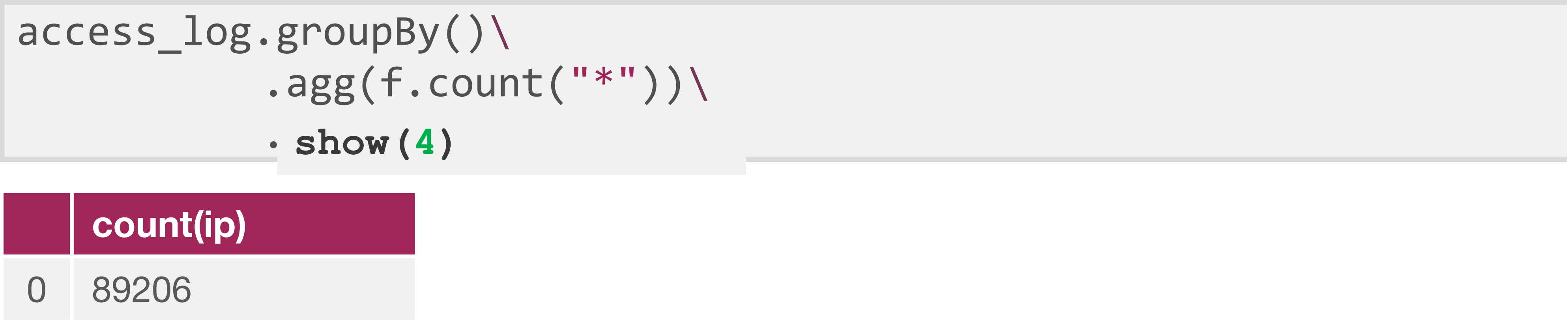
	url	count(DISTINCT ip)
0	/id37020	4
1	/id47695	2
2	/id77559	1

Aggregation

```
access_log.groupBy(f.length("url"))\
          .agg(f.count("*"))\
          .show(4)
```

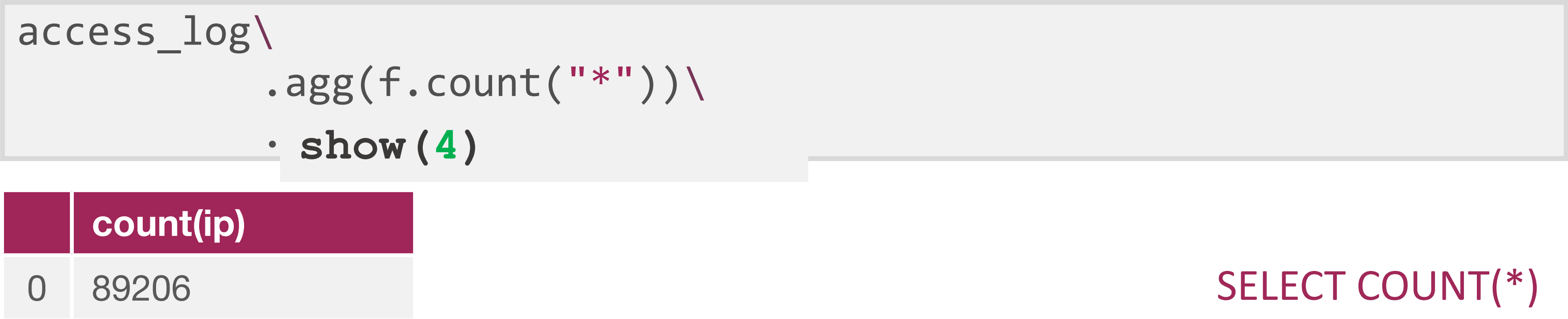
	length(url)	count(1)
0	31	5
1	34	4
2	28	4
3	27	7

Aggregation – a single group



```
access_log.groupBy()\
            .agg(f.count("*"))\
            .show(4)
```

	count(ip)
0	89206



```
access_log\
           .agg(f.count("*"))\
           .show(4)
```

	count(ip)
0	89206

SELECT COUNT(*)
FROM web.access_log

Aggregates

[1, 1, 2, 2, 2, 3, 3, 3, 3]

collect_list = [1, 1, 2, 2, 2, 3, 3, 3, 3]

count = 9

sum = 20

math

min max avg var ...

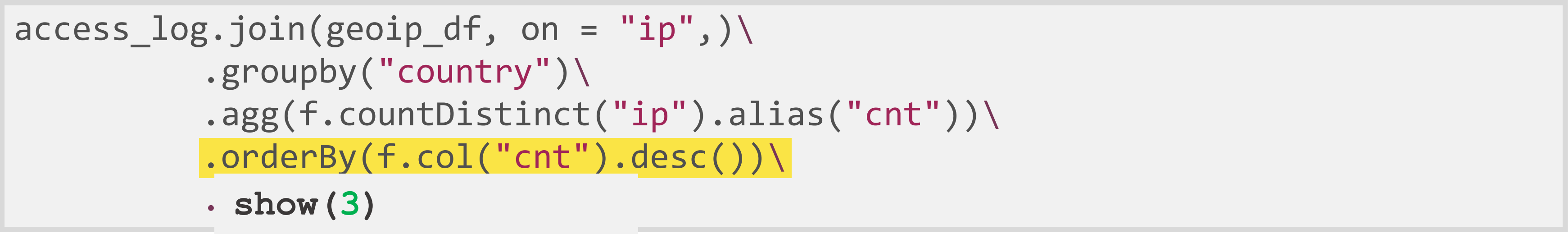
distinct

collect_set = [1, 2, 3]

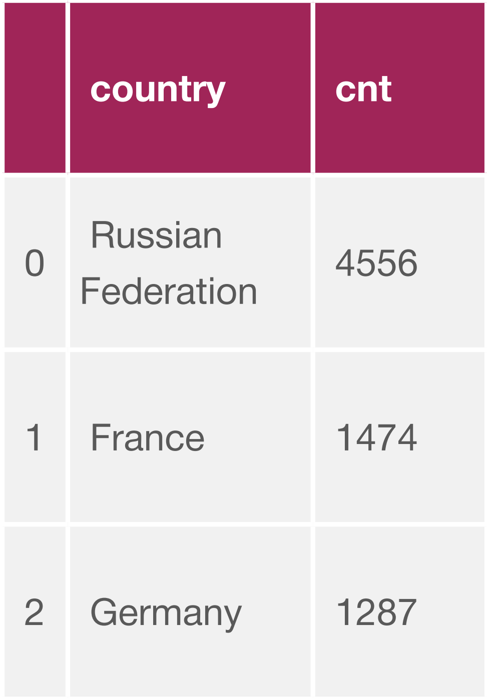
countDistinct = 3

sumDistinct = 6

Combining aggregation with join



```
access_log.join(geoip_df, on = "ip",)\
            .groupby("country")\
            .agg(f.countDistinct("ip").alias("cnt"))\
            .orderBy(f.col("cnt").desc())\
            . show(3)
```



	country	cnt
0	Russian Federation	4556
1	France	1474
2	Germany	1287

Select country, count(distinct ip) as cnt
From web.access_log natural join web.geo_ip
Group by country
Order by cnt desc

Mapping functions

Remark non pure-relational model: list includes a list of values

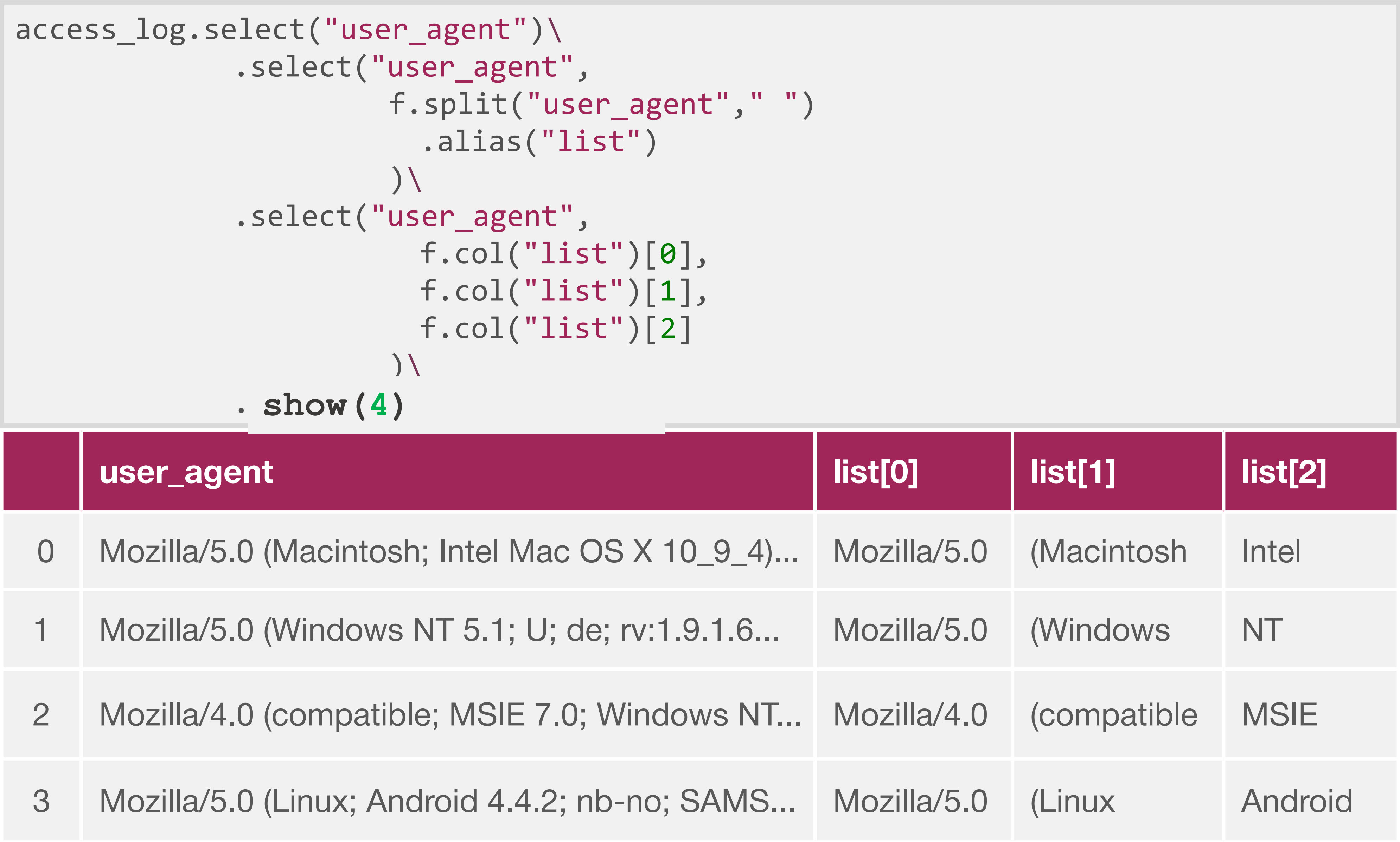
```
access_log.select("user_agent")\
          .select("user_agent",
                  f.split("user_agent"," ")
                    .alias("list")
                  )\
          . show(5)
```

1␣2␣3␣4

[1,2,3,4]

	user_agent	list
0	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	[Mozilla/5.0 (Macintosh; Intel...
1	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...	[Mozilla/5.0 (Windows NT 5.1...
2	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...	[Mozilla/4.0 (compatible; MSIE..
3	Mozilla/5.0 (Linux; Android 4.4.2; nb-no; SAMS...	[Mozilla/5.0 (Linux; Android...
4	Mozilla/5.0 (Linux; Android 4.4.2; nb-no; SAMS...	[Mozilla/5.0 (Linux; Android...

Mapping functions



```
access_log.select("user_agent")\
             .select("user_agent",
                      f.split("user_agent"," ")
                        .alias("list")
                      )\
             .select("user_agent",
                        f.col("list")[0],
                        f.col("list")[1],
                        f.col("list")[2]
                      )\
             . show(4)
```

	user_agent	list[0]	list[1]	list[2]
0	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	Mozilla/5.0	(Macintosh	Intel
1	Mozilla/5.0 (Windows NT 5.1; U; de; rv:1.9.1.6...	Mozilla/5.0	(Windows	NT
2	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT...	Mozilla/4.0	(compatible	MSIE
3	Mozilla/5.0 (Linux; Android 4.4.2; nb-no; SAMS...	Mozilla/5.0	(Linux	Android

Generating functions

```
access_log.select("user_agent")\
             .select("user_agent",
                      f.split("user_agent"," ")
                        .alias("list")
                      )\
             .select("user_agent",
                        f.explode("list"),
                      )\
             . show(4)
```

t 1

t 2

t 3

t 4

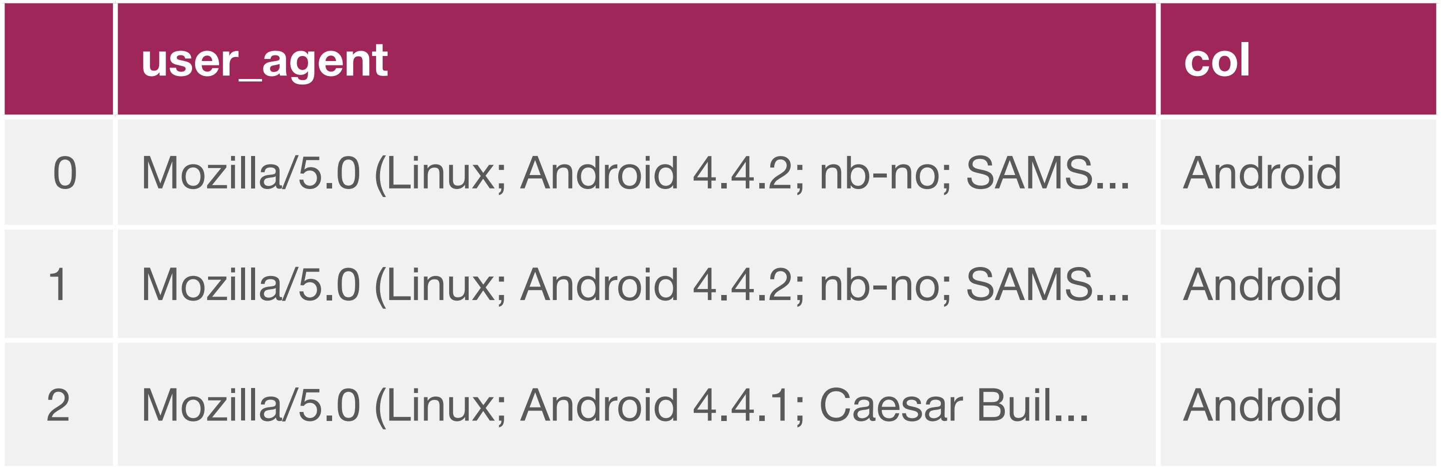
t

[1, 2, 3, 4]

	user_agent	col
0	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	Mozilla/5.0
1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	(Macintosh;
2	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	Intel
3	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	Mac
4	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...	OS

Generating functions

```
access_log.select("user_agent")\
             .select("user_agent",
                      f.split("user_agent"," ")
                        .alias("list")
                      )\
             .select("user_agent",
                        f.explode("list"),
                      )\
             .where(f.col("col") == "Android")\
             .show(4)
```



	user_agent	col
0	Mozilla/5.0 (Linux; Android 4.4.2; nb-no; SAMS...	Android
1	Mozilla/5.0 (Linux; Android 4.4.2; nb-no; SAMS...	Android
2	Mozilla/5.0 (Linux; Android 4.4.1; Caesar Buil...	Android

```
access_log.select(f.split("user_agent", " ").alias("words"))\
         .select(f.explode("words").alias("word"))\
         .groupBy("word")\
         .agg(f.count("*").alias("count"))\
         .orderBy(f.col("count").desc())\
         .show(10)
```

	words	count
0	Mozilla/5.0	75565
1	like	63791
2	Gecko)	58926
3	(KHTML,	58551
4	NT	50439
5	AppleWebKit/537.21	48648
6	Safari/537.36	48648
7	(Windows	37942
8	CLR	32140
9	.NET	31648

more operations …

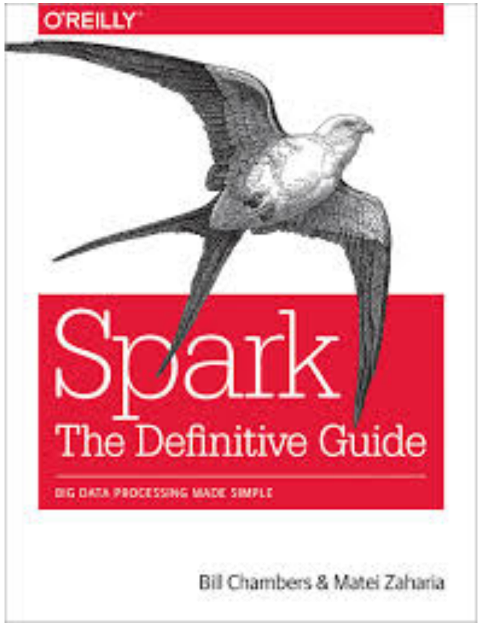
- when()
- Partitions and Windows - OLAP SQL extensions
    - over(), partitionBy()

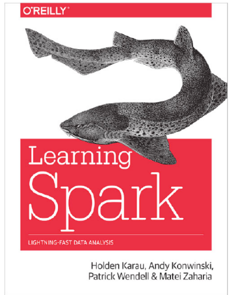
Credits & References

- Yandex “Big Data Analysis” course
- Riccardo Torlone, Università Roma Tre, Roma, slides for the “Big Data” course
- Paolo Garza, Politecnico di Torino, slides for the “ Big data ” course

- Reference:

  Chapter 9 – Spark SQL


O'REILLY
Spark
The Definitive Guide
BIG DATA PROCESSING MADE SIMPLE
Bill Chambers & Matei Zaharia


O'REILLY
Learning
Spark
LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia

https://spark.apache.org/docs/latest/sql-programming-guide.html