Module 4: Big Data Analytics in Apache Spark

Big Data Analytics with Spark

- Spark Dataframes to work with tabular data
- Data cleaning, summary, statistics
- Spark Dataframes with SQL and Hive

Open PySpark

```
PYSPARK_DRIVER_PYTHON=ipython pyspark
```

Introduction to Spark Dataframes

Types of RDD: text

from local filesystem:

```
text_RDD =
sc.textFile("file:///home/cloudera/testfile1")
```

```
text_RDD.collect()
Out[]: [u'A long time ago in a galaxy far far away']
```

Types of RDD: key-value pairs

```
def split_words(line):
    return line.split()

def create_pair(word):
    return (word, 1)

pairs_RDD=text_RDD.flatMap(split_words
).map(create_pair)
```

```
pairs_RDD.collect()
Out[]: [(u'A', 1),
 (u'long', 1),
 (u'time', 1),
 (u'ago', 1),
 (u'in', 1),
 (u'a', 1),
 (u'galaxy', 1),
 (u'far', 1),
 (u'far', 1),
 (u'away', 1)]
```

Tabular dataset

Most real-world datasets have
records (rows)

each with
multiple values (columns)

Tweets

user	text	datetime	favorites	retweets
andreazonca	"spark is cool"	"2015-10-1 9:04"	5	3

Reviews

business	text	datetime	starts	user
Pan Bon	"great pizza!"	"2015-10-1 9:04"	5	andreazonca

Logs

http_code	ip	datetime	user_agent
200	127.0.0.1	"2015-10-1 9:04"	Firefox

Tabular datasets

```
students = sc.parallelize([
[100, "Alice", 8.5, "Computer Science"],
[101, "Bob", 7.1, "Engineering"],
[102, "Carl", 6.2, "Engineering"]
])
```

Mean of a column

```
def extract_grade(row):
    return row[2]

students.map(extract_grade).mean()
Out[]: 17.26666
```

Group by column

```
def extract_degree_grade(row):
    return (row[3], row[2])

degree_grade_RDD =
students.map(extract_degree_grade)
degree_grade_RDD.collect()
```

Group by column

Intermediate RDD:

```
degree_grade_RDD.collect()
Out[]:
[('Computer Science', 8.5),
 ('Engineering', 7.0999999999999996),
 ('Engineering', 6.2000000000000002)]
```

Group by column

Reduce by key to get the final result:

```
degree_grade_RDD.reduceByKey(max).collect()
```

```
Out[]:
[('Engineering', 7.0999999999999996),
 ('Computer Science', 8.5)]
```

Introducing Spark Dataframes

User friendly interface

Under-the-hood optimization for table-like datasets

```
students_df = sqlCtx.createDataFrame(students,
    ["id", "name", "grade", "degree"])

students_df.printSchema()
root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- grade: double (nullable = true)
 |-- degree: string (nullable = true)
```

sqlCtx.createDataFrame?

```
Create a DataFrame from an RDD of tuple/list, list or pandas.DataFrame.

`schema` could be :class:`StructType` or a list of column names.

When `schema` is a list of column names, the type of each column
will be inferred from `rdd`.

When `schema` is None, it will try to infer the column name and type
from `rdd`, which should be an RDD of :class:`Row`, or namedtuple,
or dict.

If referring needed, `samplingRatio` is used to determined how many
rows will be used to do referring. The first row will be used if
`samplingRatio` is None.

:param data: an RDD of Row/tuple/list/dict, list, or pandas.DataFrame
:param schema: a StructType or list of names of columns
:param samplingRatio: the sample ratio of rows used for inferring
:return: a DataFrame

>>> l = [('Alice', 1)]
>>> sqlCtx.createDataFrame(l).collect()
[Row(_1=u'Alice', _2=1)]
>>> sqlCtx.createDataFrame(l, ['name', 'age']).collect()
[Row(name=u'Alice', age=1)]
```

Mean of a column

```
students_df.agg({"grade": "mean"}).collect()

Out[]: [Row(AVG(grade#30)=7.266666666666666)]
```

Find all available operations:

[redacted]

Group by column

```
students_df.groupBy("degree").max("grade").collect()
Out[]:
[Row(degree=u'Computer Science',
MAX(grade#30)=8.5),
 Row(degree=u'Engineering',
MAX(grade#30)=7.09999999999)]
```

Pretty print with show

```
students_df.groupBy("degree").max("grade").show()
```

```
degree            MAX(grade#30)

Computer Science  8.5

Engineering       7.1
```

Final remarks on Dataframes

- special kind of RDD
- transformations/actions/DAG work the same way
- automatic optimization to Java bytecode
- Python as fast as Scala/Java

Create Spark Dataframes

Specify a Schema

In the last video:

```
students_df = sqlCtx.createDataFrame(students,
    ["id", "name", "grade", "degree"])
```

```
from pyspark.sql.types import *

schema = StructType([

StructField("id", LongType(), True),

StructField("name", StringType(), True),

StructField("grade", DoubleType(), True),

StructField("degree", StringType(), True) ])


students_df = sqlCtx.createDataFrame(students, schema)
```

```
students_df.printSchema()
```

```
root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- grade: double (nullable = true)
 |-- degree: string (nullable = true)
```

Load a JSON file

```
students_json = [
'{"id":100, "name":"Alice", "grade":8.5,
"degree":"Computer Science"}',
'{"id":101, "name":"Bob", "grade":7.1,
"degree":"Engineering"}']
with open("students.json", "w") as f:
    f.write("\n".join(students_json))
```

Dump JSON file conent

```
!cat students.json

{"id":100, "name":"Alice", "grade":8.5, "degree":"Computer
Science"}
{"id":101, "name":"Bob", "grade":7.1,
"degree":"Engineering"}
```

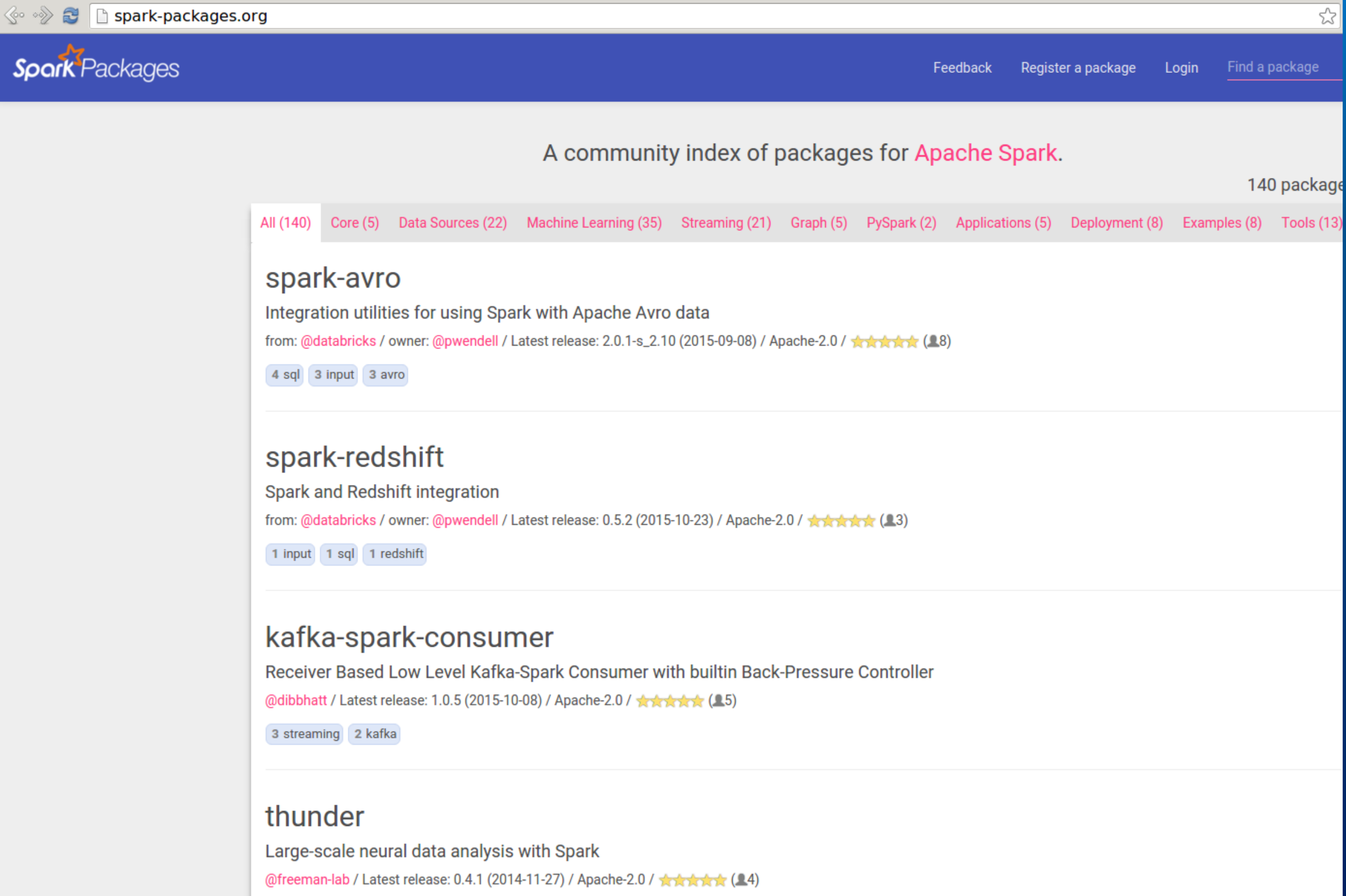
Create Dataframe with jsonFile

```
sqlCtx.jsonFile("file:///home/cloudera/students.json").show()

degree            grade id    name
Computer Science  8.5    100  Alice
Engineering       7.1    101  Bob
```

Load Dataframe from CSV

- Not included in Spark
- Load from spark-packages.org



A community index of packages for Apache Spark.

140 package

All (140) Core (5) Data Sources (22) Machine Learning (35) Streaming (21) Graph (5) PySpark (2) Applications (5) Deployment (8) Examples (8) Tools (13)

spark-avro

Integration utilities for using Spark with Apache Avro data

from: @databricks / owner: @pwendell / Latest release: 2.0.1-s_2.10 (2015-09-08) / Apache-2.0 / ★★★★★ (👤8)

4 sql 3 input 3 avro

spark-redshift

Spark and Redshift integration

from: @databricks / owner: @pwendell / Latest release: 0.5.2 (2015-10-23) / Apache-2.0 / ★★★★★ (👤3)

1 input 1 sql 1 redshift

kafka-spark-consumer

Receiver Based Low Level Kafka-Spark Consumer with builtin Back-Pressure Controller

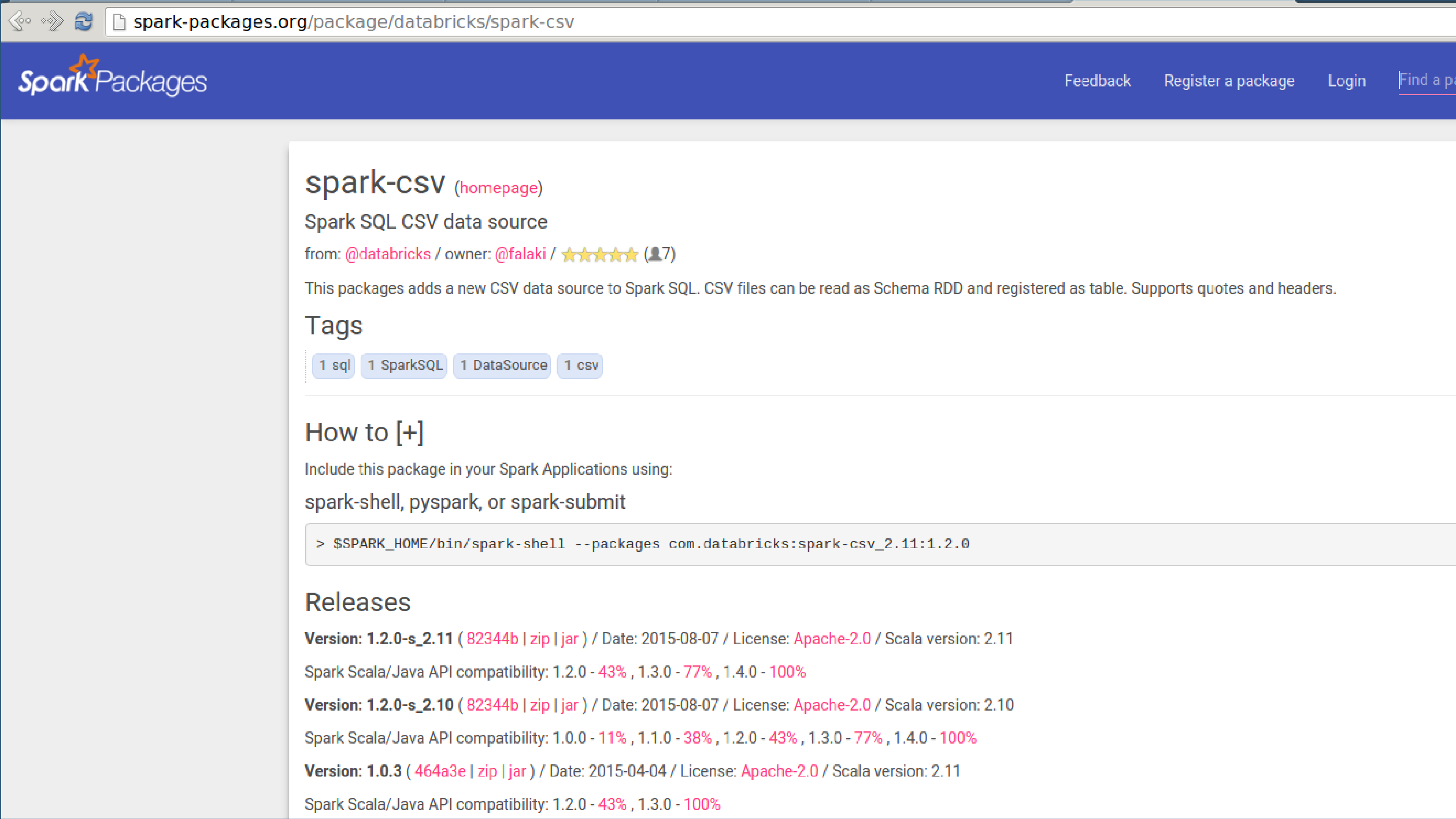
@dibbhatt / Latest release: 1.0.5 (2015-10-08) / Apache-2.0 / ★★★★★ (👤5)

3 streaming 2 kafka

thunder

Large-scale neural data analysis with Spark

@freeman-lab / Latest release: 0.4.1 (2014-11-27) / Apache-2.0 / ★★★★★ (👤4)



spark-csv (homepage)

Spark SQL CSV data source

from: @databricks / owner: @falaki / ★★★★★ (7)

This packages adds a new CSV data source to Spark SQL. CSV files can be read as Schema RDD and registered as table. Supports quotes and headers.

Tags

1 sql 1 SparkSQL 1 DataSource 1 csv

How to [+]

Include this package in your Spark Applications using:

spark-shell, pyspark, or spark-submit

```
> $SPARK_HOME/bin/spark-shell --packages com.databricks:spark-csv_2.11:1.2.0
```

Releases

Version: 1.2.0-s_2.11 (82344b | zip | jar) / Date: 2015-08-07 / License: Apache-2.0 / Scala version: 2.11

Spark Scala/Java API compatibility: 1.2.0 - 43% , 1.3.0 - 77% , 1.4.0 - 100%

Version: 1.2.0-s_2.10 (82344b | zip | jar) / Date: 2015-08-07 / License: Apache-2.0 / Scala version: 2.10

Spark Scala/Java API compatibility: 1.0.0 - 11% , 1.1.0 - 38% , 1.2.0 - 43% , 1.3.0 - 77% , 1.4.0 - 100%

Version: 1.0.3 (464a3e | zip | jar) / Date: 2015-04-04 / License: Apache-2.0 / Scala version: 2.11

Spark Scala/Java API compatibility: 1.2.0 - 43% , 1.3.0 - 100%

Restart PySpark

```
PYSPARK_DRIVER_PYTHON=ipython pyspark --packages com.databricks:spark-csv_2.10:1.2.0
```

Automatically download and include new packages and dependencies

Load sample yelp csv

```
yelp_df = sqlCtx.load(
source="com.databricks.spark.csv",
header = 'true',
inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_co
nfigs_yelp_demo/index_data.csv')
```

```
yelp_df.printSchema()
root
 |-- business_id: string (nullable = true)
 |-- cool: integer (nullable = true)
 |-- date: string (nullable = true)
 |-- funny: integer (nullable = true)
 |-- id: string (nullable = true)
 |-- stars: integer (nullable = true)
 |-- text: string (nullable = true)
 |-- type: string (nullable = true)
 |-- useful: integer (nullable = true)
 |-- user_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- full_address: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- neighborhoods: string (nullable = true)
 |-- open: string (nullable = true)
 |-- review_count: integer (nullable = true)
 |-- state: string (nullable = true)
```

```
yelp_df.count()
Out[]: 1000L
```

Analytics with Dataframes on a Yelp reviews dataset

Explore the Yelp dataset

```
yelp_df = sqlCtx.load(
source='com.databricks.spark.csv',
header = 'true',
inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_configs_yelp_demo/index_data.csv')
```

Reference a column

As attribute:

```
yelp_df.useful
      Out[]: Column<useful>
```

As key:

```
yelp_df["useful"]
      Out[]: Column<useful>
```

Filtering

```
yelp_df.filter(yelp_df.useful >= 1).count()

yelp_df.filter(yelp_df["useful"] >= 1).count()

yelp_df.filter("useful >= 1").count()

Out[]: 601L
```

select

```
yelp_df["useful"].agg({"useful":"max"}).collect()
Out[]: AttributeError: 'Column' object has no attribute 'agg'

yelp_df.select("useful")
Out[]: DataFrame[useful: int]

yelp_df.select("useful").agg({"useful":"max"}).collect()
Out[]: [Row(MAX(useful#267)=28)]
```

Create a modified DataFrame

Rescale the useful column from 0-28 to 0-100.

Create a 2 columns DataFrame

```
yelp_df.select("id", "useful").take(5)

[Row(id=u'fWKvX83p0-ka4JS3dc6E5A', useful=5),
 Row(id=u'IjZ33sJrzXqU-0X6U8NwyA', useful=0),
 Row(id=u'IESLBzqUCLdSzSqm0eCSxQ', useful=1),
 Row(id=u'G-WvGaISbqqaMHlNnByodA', useful=2),
 Row(id=u'1uJFq2r5QfJG_6ExMRCaGw', useful=0)]
```

Modify column

```
yelp_df.select("id", yelp_df.useful/28*100).show(5)

id                        ((useful / 28) * 100)
fWKvX83p0-ka4JS3d...   17.857142857142858
ljZ33sJrzXqU-0X6U...    0.0
IESLBzqUCLdSzSqm0... 3.571428571428571
G-WvGaISbqqaMHlNn... 7.142857142857142
1uJFq2r5QfJG_6ExM...  0.0

```

Cast (truncate) to integer

```
yelp_df.select("id",
(yelp_df.useful/28*100).cast("int")).show(5)

id                           CAST(((useful / 28) * 100)), IntegerType)
fWKvX83p0-ka4JS3d... 17
IjZ33sJrzXqU-0X6U...     0
IESLBzqUCLdSzSqm0... 3
G-WvGaISbqqaMHINn... 7
1uJFq2r5QfJG_6ExM... 0
```

Save as new dataframe

```
useful_perc_data = yelp_df.select(
    "id",
    (yelp_df.useful/28*100).cast("int")
)
```

```
useful_perc_data.columns
```

```
Out[]: [u'id', u'CAST(((useful / 28) * 100), IntegerType)']
```

alias - rename a column

```
useful_perc_data = yelp_df.select(
    "id",
    (yelp_df.useful/28*100).cast("int").alias("useful_perc")
)
```

```
useful_perc_data.columns
```

```
Out[]: [u'id', u'useful_perc']
```

alias - rename a column

```
useful_perc_data = yelp_df.select(
    "id",
    (yelp_df.useful/28*100).cast("int").alias("useful_perc")
)
```

```
useful_perc_data.columns
```

```
Out[]: [u'id', u'useful_perc']
```

alias - rename also id

```
useful_perc_data = yelp_df.select(
    yelp_df["id"].alias("uid"),
    (yelp_df.useful/28*100).cast("int").alias("useful_perc")
)
```

```
useful_perc_data.columns
```

```
Out[]: [u'uid', u'useful_perc']
```

Ordering by column

Import functions for ascending/descending order:

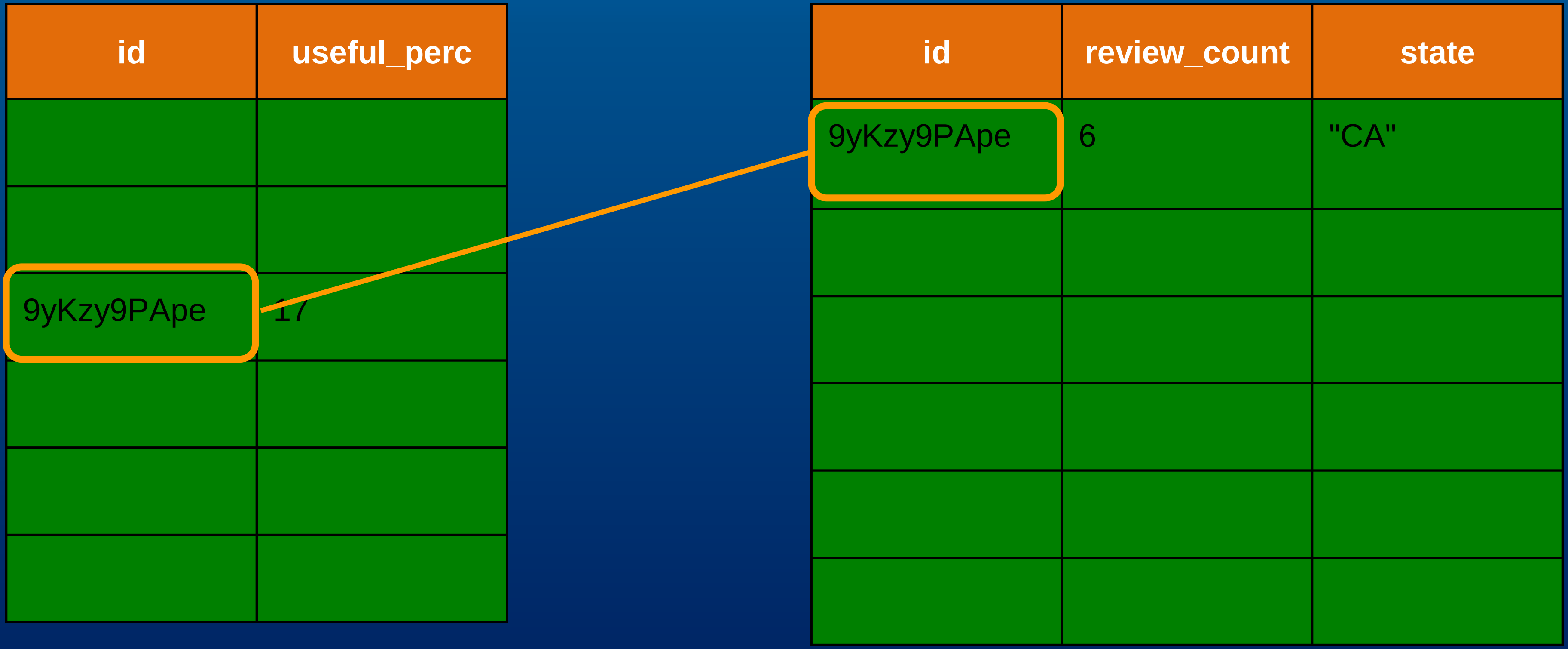
```
from pyspark.sql.functions import asc, desc
```

order by usefulness

```
useful_perc_data = yelp_df.select(
    yelp_df["id"].alias("uid"),
    (yelp_df.useful/28*100).cast("int").alias("useful_perc")
).orderBy(desc("useful_perc"))
```

```
useful_perc_data.show(2)
uid                        useful_perc
RqwFPp_qPu-1h87pG... 100
YAXPKM-Hck6-mjF74... 82
```

Join inputs



id	useful_perc
9yKzy9PApe	17

id	review_count	state
9yKzy9PApe	6	"CA"

Join results

id	useful_perc	review_count
9yKzy9PApe	17	6

Join

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
)
```

Join - select

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
).select(useful_perc_data.uid, "useful_perc", "review_count")
```

Join - select - show

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
).select(useful_perc_data.uid, "useful_perc",
"review_count").show(5)
```

Output dataset

```
uid                      useful_perc   review_count
WRBYytJAaJI1BTQG5...    71              362
GXj4PNAi095-q9ynP...      3               76
1sn0-eY_d1Dhr6Q2u...     0                9
MtFe-FuiOmo0vlo16...     0                7
EMYmuTlyeNBy5QB9P...   7               19
```

Cache in memory

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
).cache().select(useful_perc_data.uid, "useful_perc",
"review_count").show(5)
```

Run it again!

Analytics with Dataframes on HTTP server logs

Log analytics

Available in the Cloudera VM at:

```
/usr/lib/hue/apps/search/examples/collections/solr_configs_log_analytics_demo/index_data.csv
```

Log analytics

Check file contents on the terminal:

```
head /usr/lib/hue/apps/search/examples/collections/solr_configs_log_analytics_demo/index_data.csv
```

Columns

code,protocol,request,app,user_agent_major,region_code,country_code,id,city,subapp,latitude,method,client_ip,user_agent_family,bytes,referer,country_name,extension,url,os_major,longitude,device_family,record,user_agent,time,os_family,country_code3

Start PySpark

Need to load spark-csv for CSV support:

```
PYSPARK_DRIVER_PYTHON=ipython pyspark --packages com.databricks:spark-csv_2.10:1.X.X
```

(Try to) read logs CSV

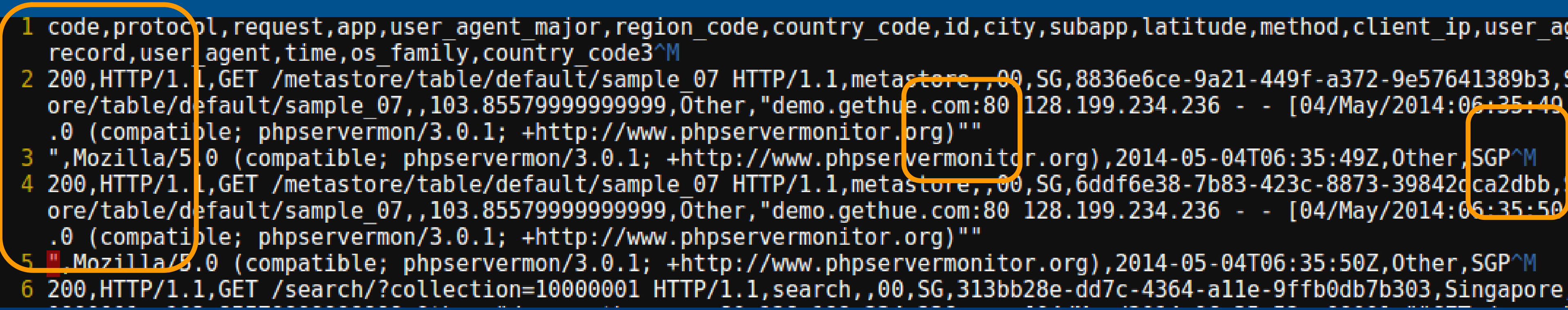
```
logs_df = sqlCtx.load(
source="com.databricks.spark.csv",
header = 'true',
inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_configs_log_analytics_demo/index_data.csv')

logs_df.count()
```

Parsing error

```
ERROR csv.CsvRelation$: Exception while parsing
line: ",Mozilla/4.0 (compatible; MSIE 7.0;
Windows NT 5.1; Trident/4.0; ....
```

Inspect the file with VIM



```
code,protocol,request,app,user_agent_major,region_code,country_code,id,city,subapp,latitude,method,client_ip,user_ag
record,user_agent,time,os_family,country_code3^M
200,HTTP/1.1,GET /metastore/table/default/sample_07 HTTP/1.1,metastore,,00,SG,8836e6ce-9a21-449f-a372-9e57641389b3,S
ore/table/default/sample_07,,103.855799999999,Other,"demo.gethue.com:80 128.199.234.236 - - [04/May/2014:06:35:49
.0 (compatible; phpservermon/3.0.1; +http://www.phpservermonitor.org)""
",Mozilla/5.0 (compatible; phpservermon/3.0.1; +http://www.phpservermonitor.org),2014-05-04T06:35:49Z,Other,SGP^M
200,HTTP/1.1,GET /metastore/table/default/sample_07 HTTP/1.1,metastore,,00,SG,6ddf6e38-7b83-423c-8873-39842dca2dbb,S
ore/table/default/sample_07,,103.855799999999,Other,"demo.gethue.com:80 128.199.234.236 - - [04/May/2014:06:35:50
.0 (compatible; phpservermon/3.0.1; +http://www.phpservermonitor.org)""
",Mozilla/5.0 (compatible; phpservermon/3.0.1; +http://www.phpservermonitor.org),2014-05-04T06:35:50Z,Other,SGP^M
200,HTTP/1.1,GET /search/?collection=10000001 HTTP/1.1,search,,00,SG,313bb28e-dd7c-4364-a11e-9ffb0db7b303,Singapore,
```

Access Hadoop configuration

Spark relies on Hadoop functionality for reading data.

```
sc._jsc.hadoopConfiguration()
```

Set input file delimiter

Spark relies on Hadoop functionality for reading data.

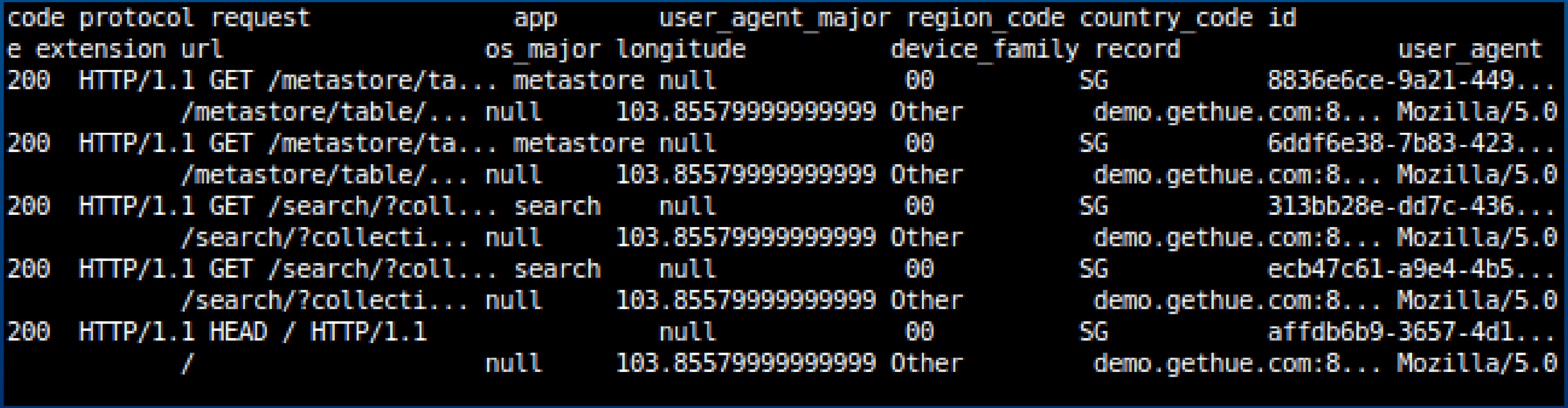
```
sc._jsc.hadoopConfiguration().set('textinputformat.record.delimiter', '\r\n')
```

Read logs CSV

```
logs_df = sqlCtx.load(
source="com.databricks.spark.csv",
header = 'true', inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_co
nfigs_log_analytics_demo/index_data.csv')

logs_df.count()
Out[]: 9410L
```

Display of logs DataFrame



```
code protocol request              app       user_agent_major region_code country_code id
e extension url                  os_major longitude          device_family record               user_agent
200  HTTP/1.1 GET /metastore/ta... metastore null             00          SG           8836e6ce-9a21-449...
            /metastore/table/... null     103.85579999999999 Other         demo.gethue.com:8... Mozilla/5.0
200  HTTP/1.1 GET /metastore/ta... metastore null             00          SG           6ddf6e38-7b83-423...
            /metastore/table/... null     103.85579999999999 Other         demo.gethue.com:8... Mozilla/5.0
200  HTTP/1.1 GET /search/?coll... search    null             00          SG           313bb28e-dd7c-436...
            /search/?collecti... null     103.85579999999999 Other         demo.gethue.com:8... Mozilla/5.0
200  HTTP/1.1 GET /search/?coll... search    null             00          SG           ecb47c61-a9e4-4b5...
            /search/?collecti... null     103.85579999999999 Other         demo.gethue.com:8... Mozilla/5.0
200  HTTP/1.1 HEAD / HTTP/1.1                null             00          SG           affdb6b9-3657-4d1...
            /                    null     103.85579999999999 Other         demo.gethue.com:8... Mozilla/5.0
```

```
root
 |-- code: integer (nullable = true)
 |-- protocol: string (nullable = true)
 |-- request: string (nullable = true)
 |-- app: string (nullable = true)
 |-- user_agent_major: integer (nullable = true)
 |-- region_code: string (nullable = true)
 |-- country_code: string (nullable = true)
 |-- id: string (nullable = true)
 |-- city: string (nullable = true)
 |-- subapp: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- method: string (nullable = true)
 |-- client_ip: string (nullable = true)
 |-- user_agent_family: string (nullable = true)
 |-- bytes: integer (nullable = true)
 |-- referer: string (nullable = true)
 |-- country_name: string (nullable = true)
 |-- extension: string (nullable = true)
```

Count by HTTP code

Count the log events by HTTP code (i.e. how many 200 OK, 404 Not found...)

```
logs_df.groupBy("code").count().show()
```

```
code count
500  2
301  71
302  1943
502  6
304  117
400  1
200  7235
401  10
404  11
```

```
from pyspark.sql.functions import asc, desc
logs_df.groupBy("code").count().orderBy(desc("count")).show()
```

```
code count
200  7235
302  1943
304  117
301  71
408  14
404  11
```

Compute average

```
logs_df.groupBy("code").avg("bytes").show()

code AVG(bytes#47)
500  4684.5
301  424.61971830985914
302  415.6510550694802
502  581.0
304  185.26495726495727
400  0.0
```

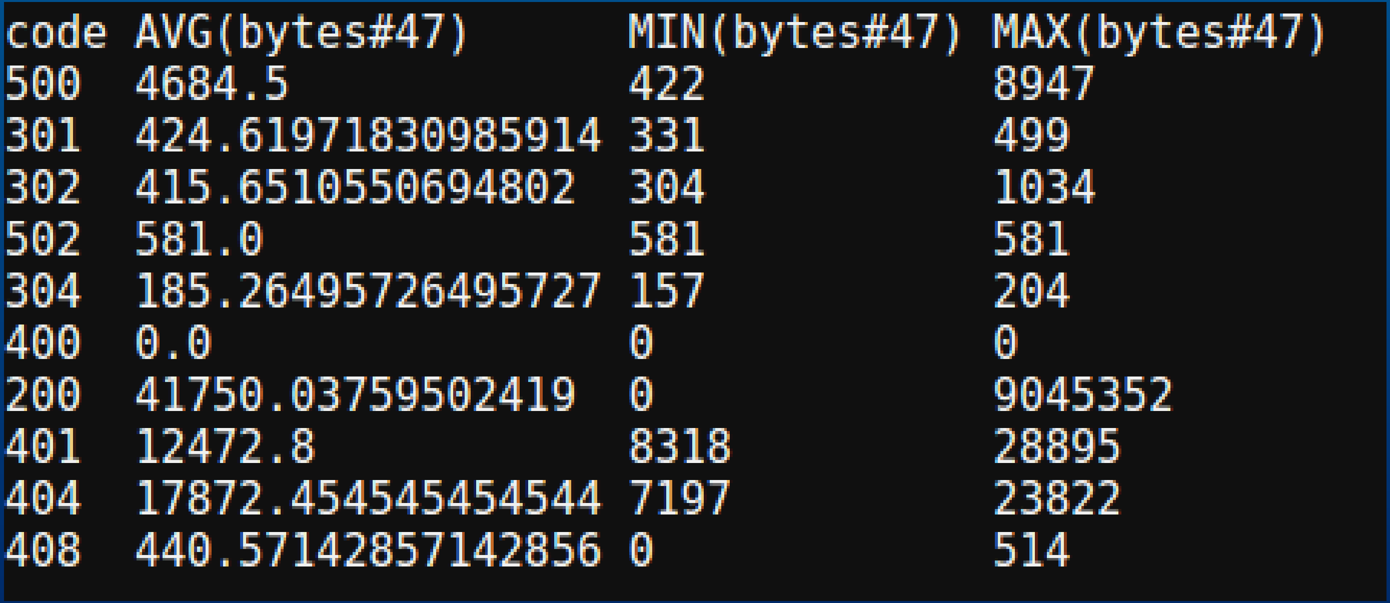
Mean, Min, Max by code

Compute in a single operation Mean, Min and Max by HTTP code

```
import pyspark.sql.functions as F

logs_df.groupBy("code").agg(
                    logs_df.code,
                    F.avg(logs_df.bytes),
                    F.min(logs_df.bytes),
                    F.max(logs_df.bytes)
).show()
```

Mean, Min, Max by code



```
code AVG(bytes#47)      MIN(bytes#47) MAX(bytes#47)
500  4684.5             422           8947
301  424.61971830985914 331           499
302  415.6510550694802  304           1034
502  581.0              581           581
304  185.26495726495727 157           204
400  0.0                0             0
200  41750.03759502419  0             9045352
401  12472.8            8318          28895
404  17872.454545454544 7197          23822
408  440.57142857142856 0             514
```

Completed DataFrames

- Completed analytics with DataFrames
- Next we'll focus on interoperability with SQL query language and Hive