Opening CSV Files in Apache Spark - The Spark Data Sources API and Spark-CSV

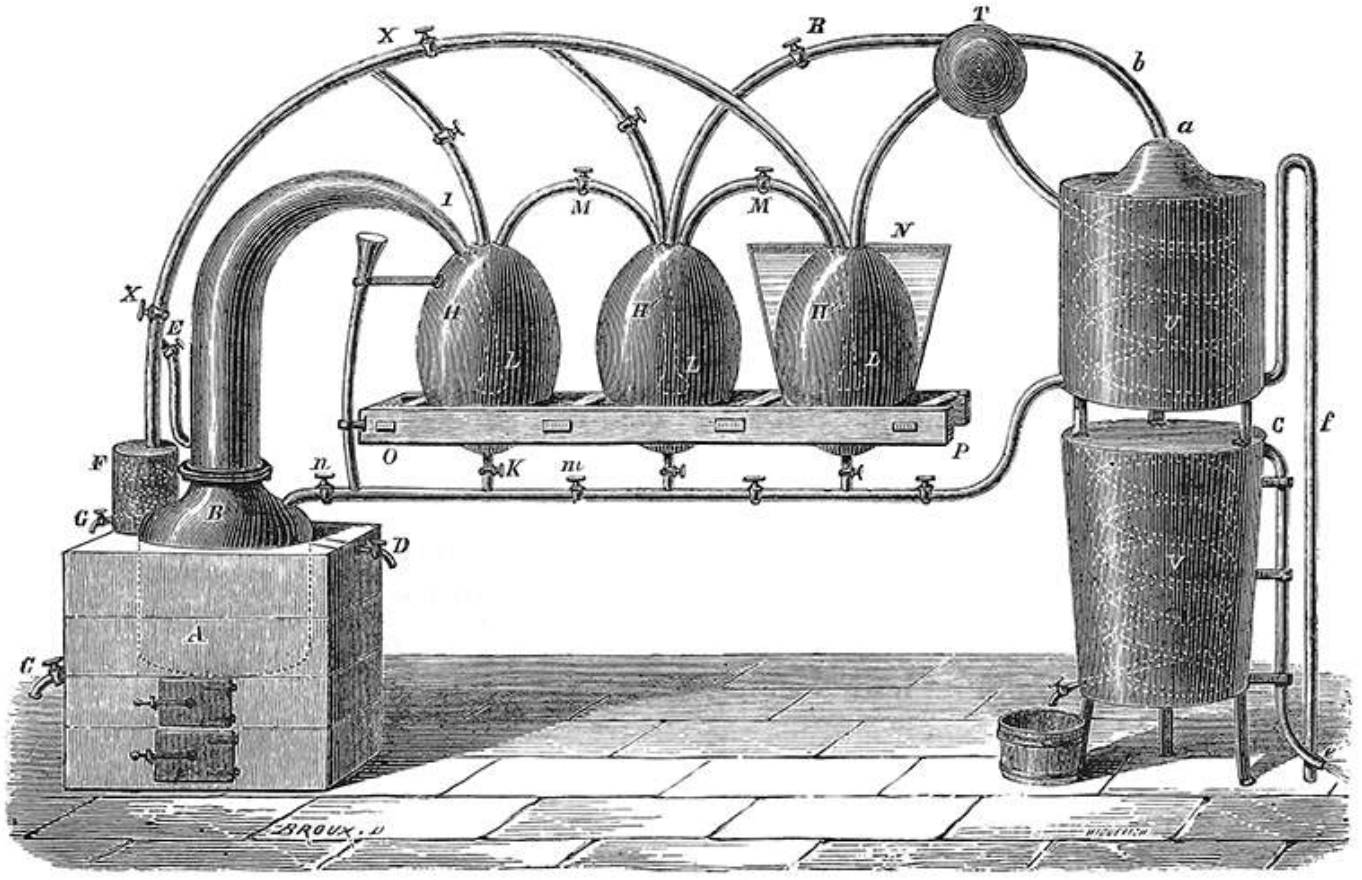

Fig. 245. — Appareil distillatoire d'Édouard Adam, breveté en 1805.

Written by Bill Chambers on Fri, 14 Aug 2015 00:00:00 UTC.

Tweet Share 18 G+1 0

Introduction

CSV files are everywhere and used frequently in the enterprise. Therefore having an efficient way to open and use them is an absolute must for any Apache Spark project or user. Luckily, we all don't have to reinvent the wheel when we want to use csv files because of Spark Packages (http://spark-packages.org/). Now you can leverage the communities ability to create and share their code and even create your own packages. Right now the repository is fairly small but I'm sure in the coming months development is going to speed up. Now fundamentally the problem that we are trying to solve is that we want to load a CSV file with Spark or thought of another way, convert a CSV to an RDD. We can do this manually by reading in the file but the Data

Sources API makes this a bit more repeatable and consistent with our process. Now I'm going to introduce Spark Packages with a simple one that I use often, it's the Databricks (http://spark-packages.org/package/databricks/spark-csv][spark-csv]] package from [[https://databricks.com/).

Requirements

Obviously you're going to need tutorial on Spark DataFrames (/building-apache-spark-on-your-local-machine][Spark setup]] on your computer. If you) That's really all that you're going to need besides a keyboard to specify the inclusion of our csv package.

Including the CSV package

Including packages when you start Spark seems extremely easy, all you do is just pass in some command line arguments when you start your program or shell, from the root directory of Spark, execute the following command. Remember the arguments are a bit different if you're running the python version.

```
./bin/spark-shell --packages com.databricks:spark-csv_2.11:1.0.3
```

Using the CSV Package

And there you have it, once you've started that up you are now including the package in your shell. Keep in mind that you still need to add in any other command line options that you might need to include as well (like number of cores etc.). Now, how does one go about actually using the package? That varies a bit language to language so let's go ahead and explore the details.

Loading a CSV

Loading a CSV file is super straightforward with these packages but there are some pain points. First of all, unlike a nice columnar file format like parquet, we're not going to get back types when we read in a csv - all columns are going to be read in as a string which is a bit of a pain. Let's go ahead and get started to see how it works. The csv package asctually just becomes a new data source under the hood. All we have to do is use it through the SQL context that Spark provides. Because I'm doing this on my local machine I'm not going to choose something too crazy. I'm going to analyze the data from the Data Expo '09 (http://stat-computing.org/dataexpo/2009/the-data.html),

which has a bunch of data on airplane flights. I've downloaded the 2008 data to analyze but in later posts we'll be analyzing all of the data together.

```
val df = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").load("../Downloads/2008.csv")
```

```
df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").lo
ad("../Downloads/2008.csv")
```

Now we have it loaded in as a DataFrame, now of course we can print the schema.

```
df.printSchema()
```

```
df.printSchema()
```

```
root
 |-- Year: string (nullable = true)
 |-- Month: string (nullable = true)
 |-- DayofMonth: string (nullable = true)
 |-- DayOfWeek: string (nullable = true)
 |-- DepTime: string (nullable = true)
 |-- CRSDepTime: string (nullable = true)
 |-- ArrTime: string (nullable = true)
 |-- CRSArrTime: string (nullable = true)
 |-- UniqueCarrier: string (nullable = true)
 |-- FlightNum: string (nullable = true)
 |-- TailNum: string (nullable = true)
 |-- ActualElapsedTime: string (nullable = true)
 |-- CRSElapsedTime: string (nullable = true)
 |-- AirTime: string (nullable = true)
 |-- ArrDelay: string (nullable = true)
 |-- DepDelay: string (nullable = true)
 |-- Origin: string (nullable = true)
 |-- Dest: string (nullable = true)
 |-- Distance: string (nullable = true)
 |-- TaxiIn: string (nullable = true)
 |-- TaxiOut: string (nullable = true)
 |-- Cancelled: string (nullable = true)
 |-- CancellationCode: string (nullable = true)
 |-- Diverted: string (nullable = true)
 |-- CarrierDelay: string (nullable = true)
 |-- WeatherDelay: string (nullable = true)
 |-- NASDelay: string (nullable = true)
 |-- SecurityDelay: string (nullable = true)
 |-- LateAircraftDelay: string (nullable = true)
```

Querying a CSV

Now obviously we're going to struggle computing statistics on this data if they're all strings - things like averages and everything are out of reach. So what we should do is just convert the types. Here's an example of how you can do that at the column level.

```
df.col("Year").cast("int")
```


```
df.col("Year").cast("int")
```


However once we've extracted that column, how do we add it back in? Here's the current best way of setting the type of a column in a DataFrame in Spark. Now this is definitely a work around but it works.

```
val df_1 = df.withColumnRenamed("Year","oldYear")
val df_2 = df_1.withColumn("Year",df_1.col("oldYear").cast("int")).drop("oldYear")
```

```
df_1 = df.withColumnRenamed("Year","oldYear")
df_2 = df_1.withColumn("Year",df_1.col("oldYear").cast("int")).drop("oldYear")
```

Now we've replaced our old column with our new one. This can /definitely/ get messy if you're doing this with a lot of different columns but it works when you need it to. Let's go ahead and play around with some other calculations. To do that, we're going to need to convert a couple more columns. To do this, let's create a simple function to help us out.

```
def convertColumn(df: org.apache.spark.sql.DataFrame, name:String, newType:String)
 = {
  val df_1 = df.withColumnRenamed(name, "swap")
  df_1.withColumn(name, df_1.col("swap").cast(newType)).drop("swap")
}
```

```
def convertColumn(df, name, new_type):
    df_1 = df.withColumnRenamed(name, "swap")
    return df_1.withColumn(name, df_1.col("swap").cast(new_type)).drop("swap")
```

```
val df_3 = convertColumn(df_2, "ArrDelay", "int")
val df_4 = convertColumn(df_2, "DepDelay", "int")
```

```
df_3 = convertColumn(df_2, "ArrDelay", "int")
df_4 = convertColumn(df_2, "DepDelay", "int")
```

There we have it, a simple way of converting columns from a string to an integer (or any other type).

Saving as CSV

Now once we've performed some analysis, we're likely going to want to save that file as a csv. Now this is super easy too because we have available to us, the data sources api. To save the data, all that we have to do is take our DataFrame, select the relevant columns (or just the whole thing) and write it out to a file.

```
df_4.select("Year", "Cancelled").write.format("com.databricks.spark.csv").save("yea
r_cancelled.csv")
```


```
df_4.select("Year", "Cancelled").write.format("com.databricks.spark.csv").save("yea
r_cancelled.csv")
```


It's really that easy. I hope you got a feel for how to read and analyze data using CSVs in Apache Spark.

Questions or comments?

6 Comments Spark Tutorials 1 Login

Recommend 1 Share Sort by Best

Join the discussion…

disqus_XedQ1jFa84 • a year ago

In the Scala code for the function "convertColumn", there's a missing "=" in the assignment of df_1.

It should read:

val df_1 = df.withColumnRenamed(name, "swap")

1 • Reply • Share ›

Karthik Ts • a year ago

I'm not able to partition the data in sub directories when saving in CSV format, although the partitioning info is present as one of the columns, I would expect to save in sub directories named with partitioning column and value, however the it works as expected for other formats like orc and parquet which saves the partitioning info as sub directories. any clue why the csv data is not written in sub directories like orc , parquet etc ?

• Reply • Share ›

Jr • a year ago

Everything *seems* to work except when I save the csv it saves as a folder. Any idea what is going wrong?

• Reply • Share ›

b_a_chambers Mod → Jr • a year ago

Each of the text files in that folder make up the entire csv. This has to do with the number of partitions that your data has.

cat csvFolder/* > new.csv

would give it to you in one file!

• Reply • Share ›

Yugo Gautomo • a year ago

Hi Bill,

I want to ask, if we quit from Spark Shell and running again, is it have to load spark-csv package again?
Because after i quit from the shell, and i want to read csv files to dataframe it doesn't work?

Thanks.

• Reply • Share ›

b_a_chambers Mod → Yugo Gautomo • a year ago

Yes you always have to load the package again because the spark shell doesn't always know what you're going to be using and loading every package every time would be hugely inefficient. You have to specify it at the command line like I did above, or specify it in your build file for scala for inclusion into a jar

• Reply • Share ›

ALSO ON **SPARK TUTORIALS**

Getting Started with Apache Spark DataFrames in Python and Scala

1 comment • 2 years ago•

DurhamDesi — Fantastic article. Nicely done.

Spark MLLib - Predict Store Sales with ML Pipelines

3 comments • a year ago•

Laksh Lumba — Thanks Bill ,I tried your code from the git hub only and created the jar. Now I got the result, It took me around 2 hours to ...

Analyzing Flight Data: A Gentle Introduction to GraphX in Spark

7 comments • 2 years ago•

Avat

Steve — Bill - a really great tutorial - many thanks! Being new to scala and spark, I was trying to follow this in Spark 2.0.2 and hit ...

Reading and Writing S3 Data with Apache Spark

2 comments • 2 years ago•

Avat

Suriya — Spark always create a temporary file. if all your jobs got completed properly, it will be deleted automatically

Subscribe Add Disqus to your site Add Disqus Add Privacy

Follow us For More Apache Spark Tutorials

Via RSS (/feed.xml)

Via Feedly (http://cloud.feedly.com/#subscription%2Ffeed%2Fhttp%3A%2F%2Fwww.sparktutorials.net%2Ffeed.xml)

Via Email:

Email Address *

First Name

Subscribe

~1 email/month.
unsubscribe any time.

Apache®, Apache Spark, Spark® (http://spark.apache.org/), and the spark logo are either registered trademarks or trademarks of the Apache Software Foundation (http://www.apache.org/) in the United States and/or other countries. The Apache Software Foundation as no affiliation with and does not endorse or review the materials provided at this website.