SparkIntro

Jay Urbain

Apache Spark is an open-source, distributed processing system commonly used for big data workloads. Apache Spark utilizes in-memory caching and optimized execution for fast performance, and it supports general batch processing, streaming analytics, machine learning, graph databases, and ad hoc queries.

This document provides and introduction to Spark, and a hands on pySpark programming tutorial through a series of Jupyter Notebooks executed on the Databricks hosted computing environment. Covers Spark core, Spark SQL, Spark Streaming, and Spark MLib.

References:
Documentation
http://spark.apache.org/docs/2.1.0/api/python/index.html

Spark Programming Guide
https://spark.apache.org/docs/latest/rdd-programming-guide.html

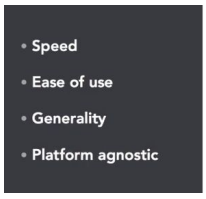
Short guide to useful pySpark dataframe commands:
https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/

Learning Spark, Matei Zaharia, Patrick Wendell, Andy Konwinski, Holden Karau. O'Reilly Media, Inc., February 2015. Note: A little dated.

Advanced Analytics with Spark: Patterns for Learning from Data at Scale, Sandy Ryza, Uri Laserson, Josh Wills, Sean Owen. O'Reilly Media, April 2015. Note: RDD focus, a little dated.

Introduction

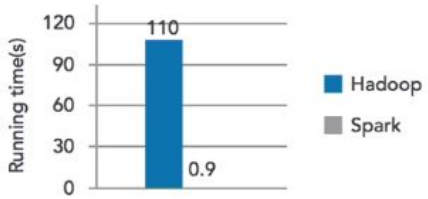
- Spark - fast and general engine for large scale data processing
- Designed for data science, supports data processing and streaming.
- In-memory distributed processing



Can run spark on Hadoop and other platforms.

Spark vs MapReduce on logistic regression

- **Speed**
- Ease of use
- Generality
- Platform agnostic



Spark supports Python, Scala, Java, R, etc. Can run from shell or notebook.

Example: PySpark wordcount with RDD (right):

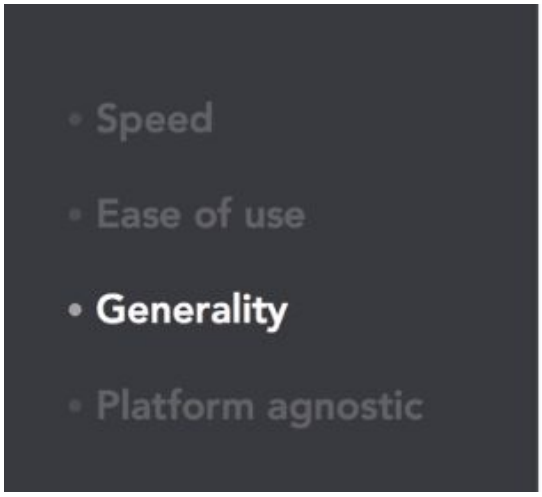
- Speed
- **Ease of use**
- Generality
- Platform agnostic

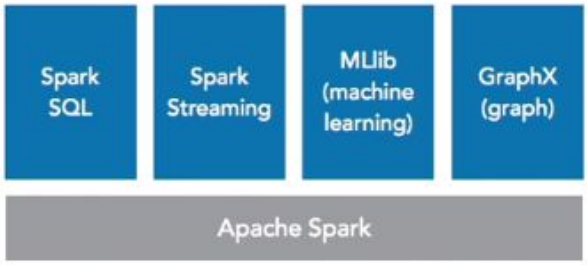
```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API

Many open source libraries are built into the platform to optimize computation over Spark's distributed processing system.

Main Spark components (GraphX not covered in this tutorial).





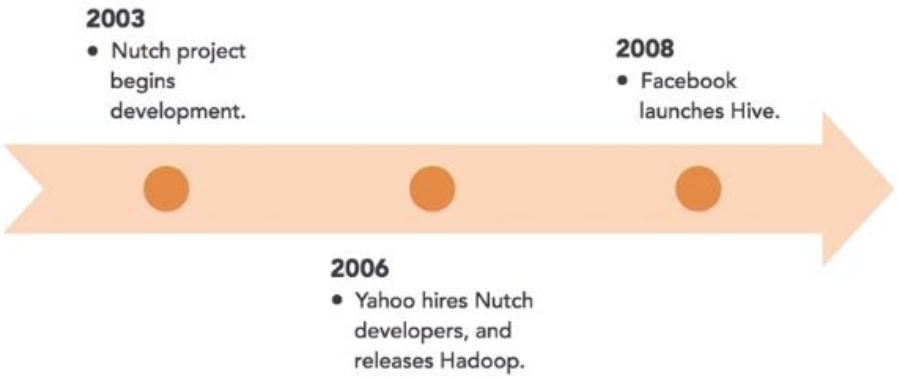
Spark is platform agnostic and can read data on any server in the Hadoop distributed file system (HDFS): Cassandra, HBASE, and noSQL databases such as Hive or other Hadoop data source, and SQL databases.

Spark has been designed to work well with other tools.

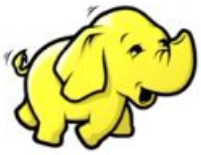
Origins of Spark

Hadoop lacks good tools for data science. This was the inspiration for Spark.

Timeline

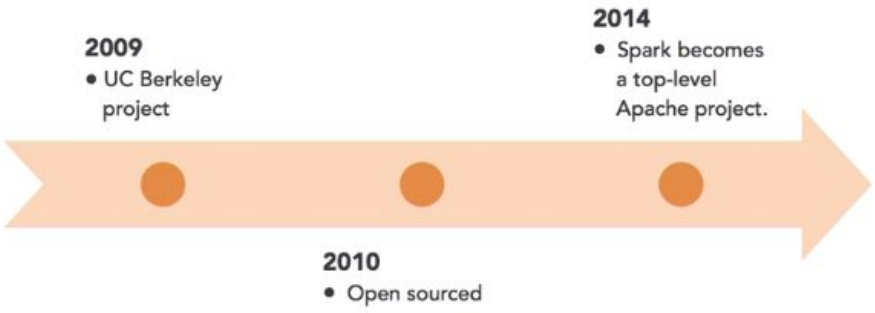


At this point there are two ways of interacting with data on Hadoop data: MapReduce (Java), and Hive. Hive is basically an abstraction of SQL that runs on top of MapReduce.



MapReduce
Java-based, batch-oriented, and slow

Hive
SQL abstraction for MapReduce



Spark Abstraction

Every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster.

The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.

RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), and transforming it.

Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.

A second abstraction in Spark is *shared variables* that can be used in parallel operations. By default, when Spark runs a function in parallel as a set of tasks on different nodes, it ships a copy of each variable used in the function to each task. Sometimes, a variable needs to be shared across tasks, or between tasks and the driver program.

Spark supports two types of shared variables: *broadcast variables*, which can be used to cache a value in memory on all nodes, and *accumulators*, which are variables that are only "added" to, such as counters and sums.

Blog explaining RDDs, Dataframes, and Datasets..
https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html

Spark Core

All spark components are dependent on Spark Core

- Foundational component
- Task distribution
- Scheduling
- Input/output

Spark SQL

- Use data frames similar to Pandas.
- Close to standard SQL

Spark Streaming

- Streaming analytics
- Micro batches
- Lambda architecture - incremental update of metrics, does not re-query underlying data

MLib

- Machine Learning
- 9x faster than Apache Mahout
- Includes common functions

GraphX

- Graph processing - relationships between entities
- In-memory version

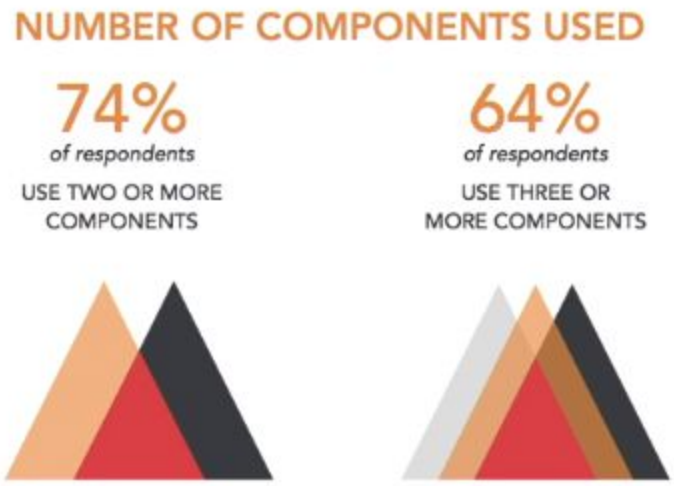
SparkR

- R package for spark
- Distributed DataFrames
- R Studio integration

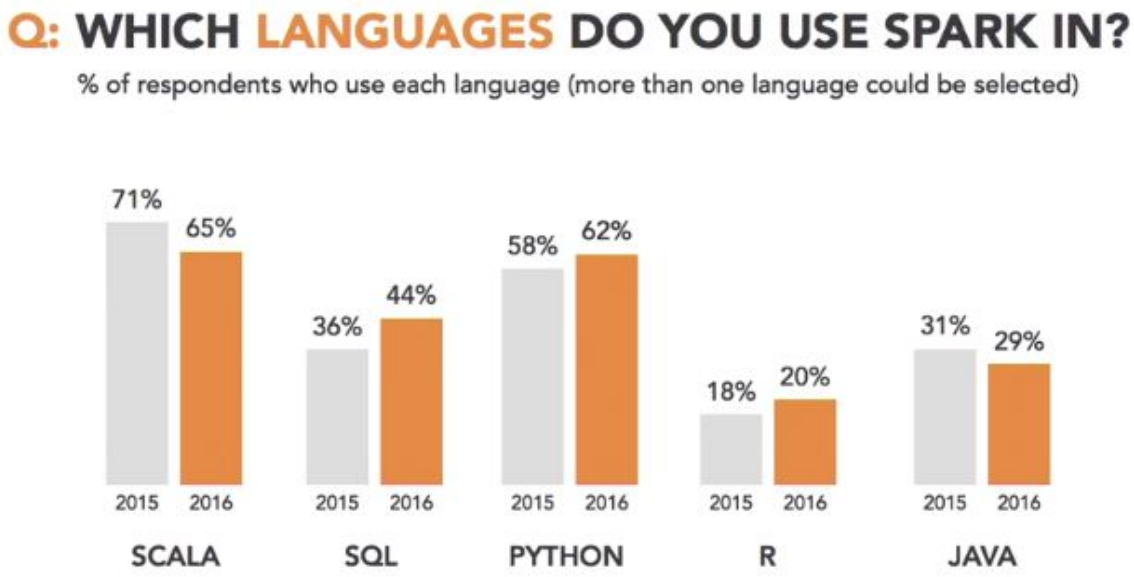
Where Spark shines

- Data integration/ETL
- Machine Learning
- BI/Analytics
- Real-time Processing
- Recommendation Engines

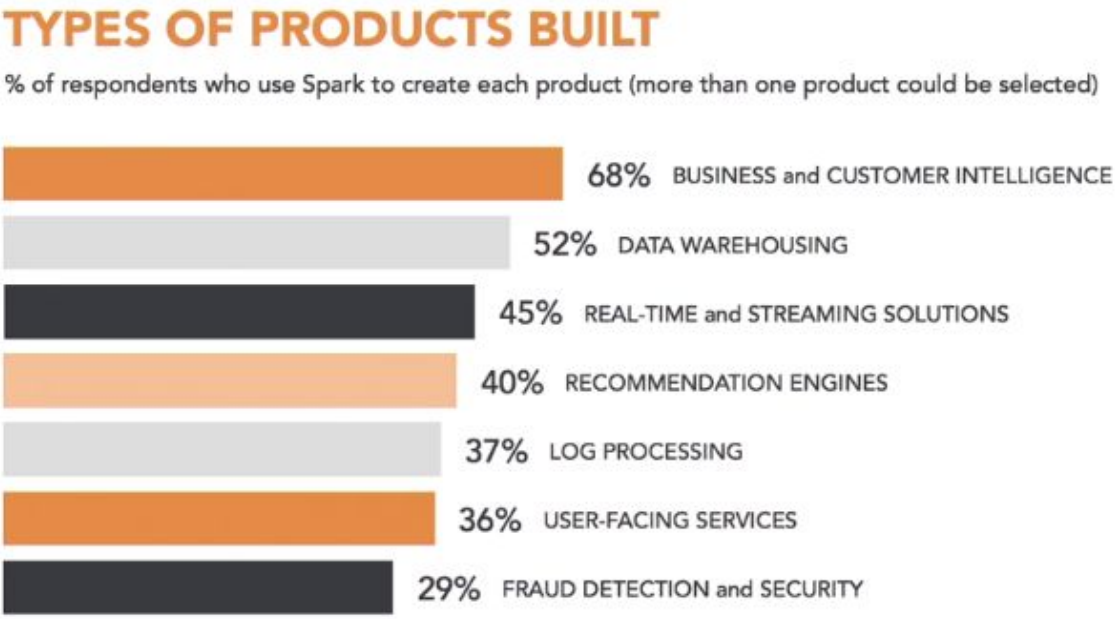
Spark is typically used as a collection of tools



What languages are being used?

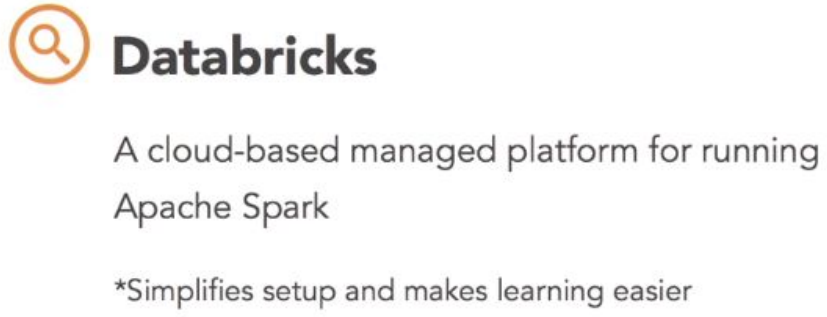


Types of applications



Overview of Databricks

- Created by people who invented Spark
- Many features for developing and deploying Spark applications
- Reduce complexity of setup and admin



Can also run on Cloudera, Hortonworks, AWS, Azure, etc.

Use community edition.

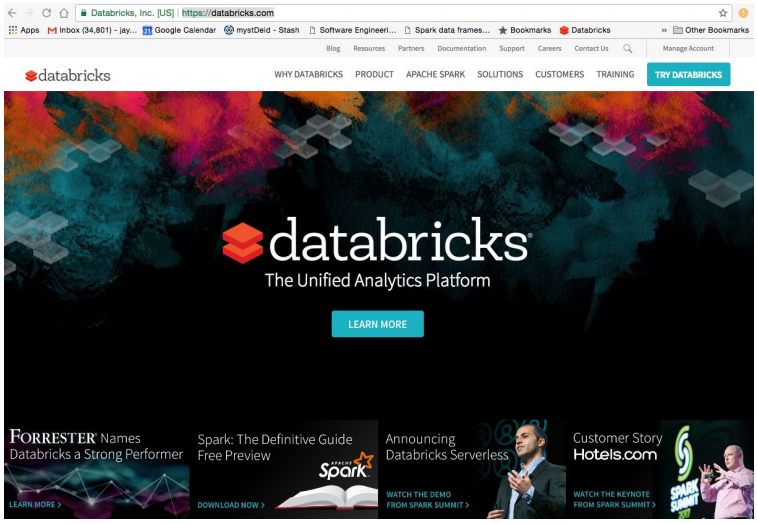
Full Platform	Community Edition
Unlimited clusters	Mini 6GB cluster
Notebooks, dashboards, production jobs, RESTful APIs	Interactive notebooks and dashboards
Interactive guide to Spark and Databricks	Public environment to share your work
Deployed to your AWS VPC	
BI tools integration	
14-day free trial (excludes AWS charges)	

Spark Terminology

- Workspaces - collection of your files
- Notebooks - environment running Python, Scala, etc.
- Libraries - extend Spark functionality, e.g., Scikit-learn
- Tables - SQL tables
- Clusters - collection of processing nodes
- Jobs - running projects

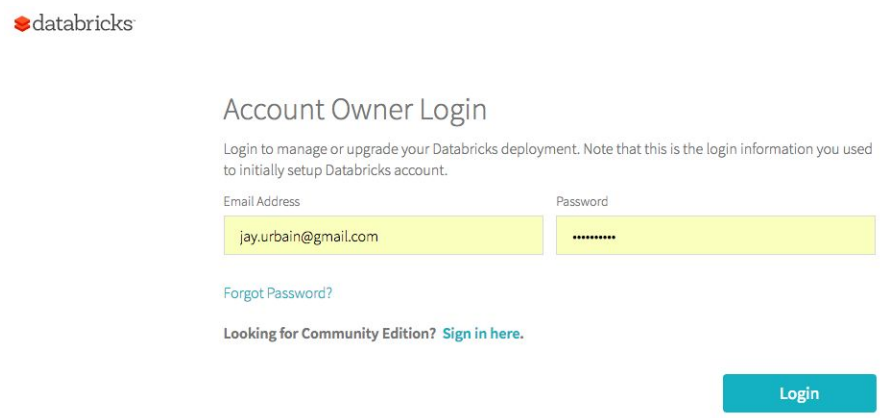
Create account on Databricks

https://databricks.com/

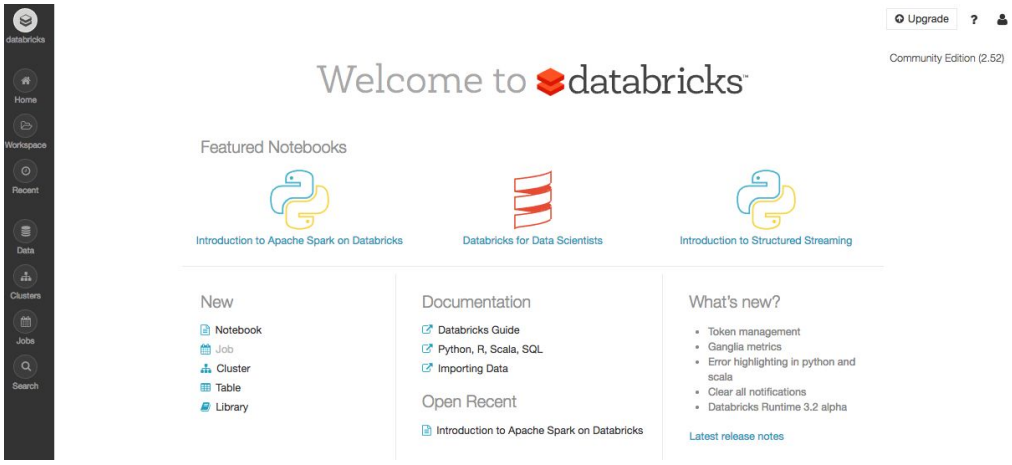


Community edition:

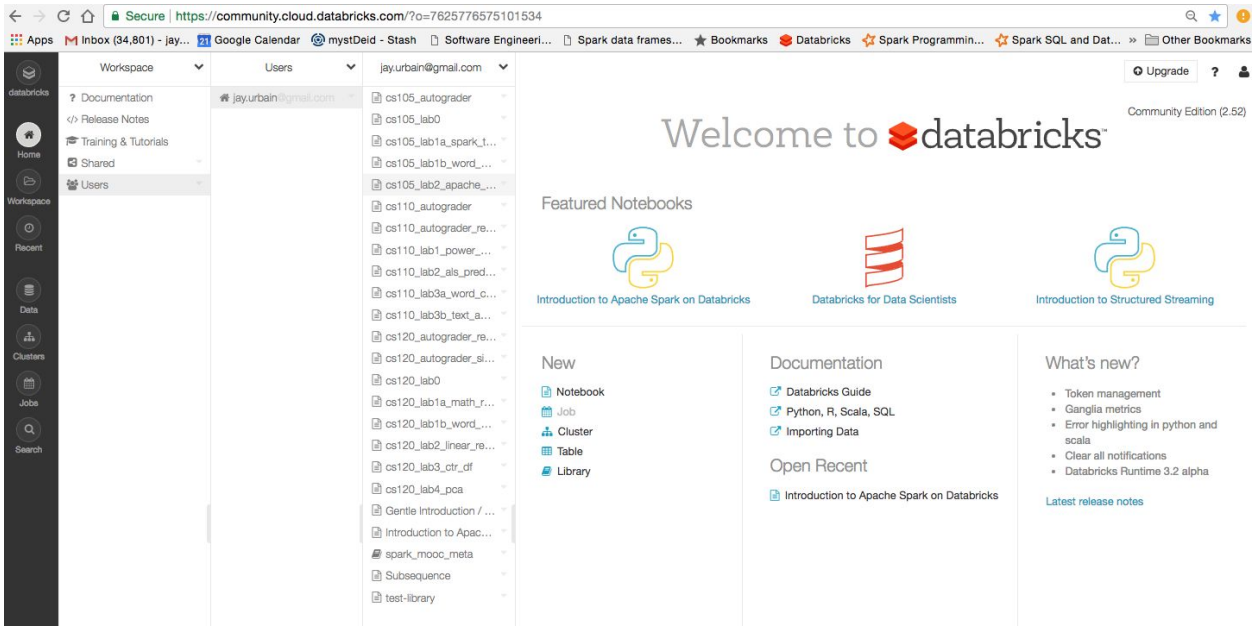
https://community.cloud.databricks.com



You can review their featured tutorials later.



Select Home (left vertical navigation bar)

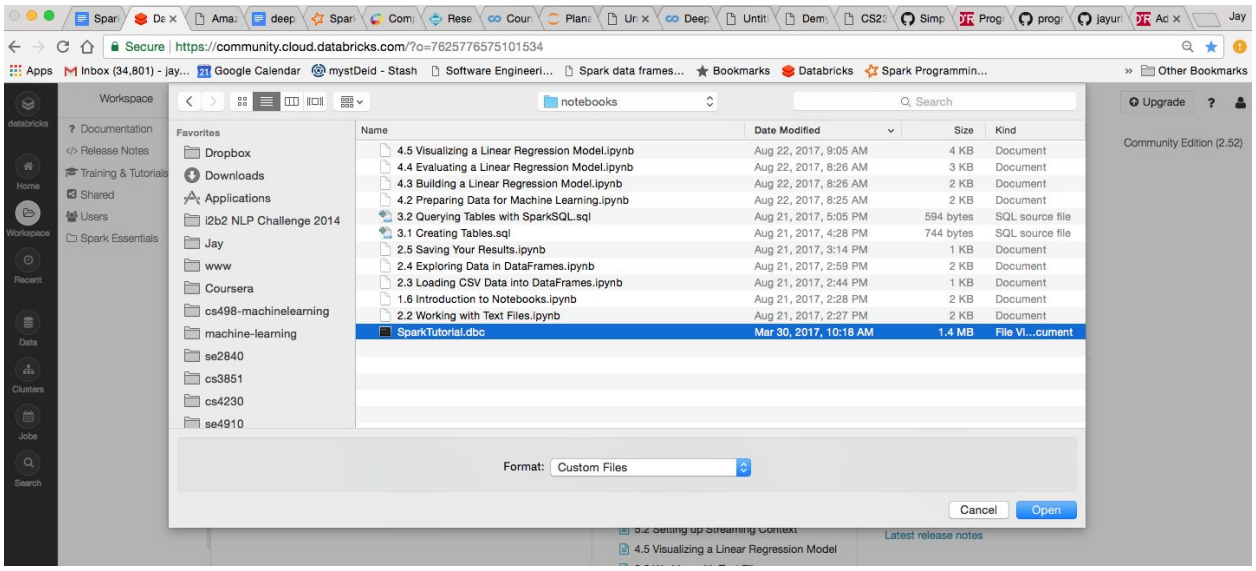


Notebooks and PySpark

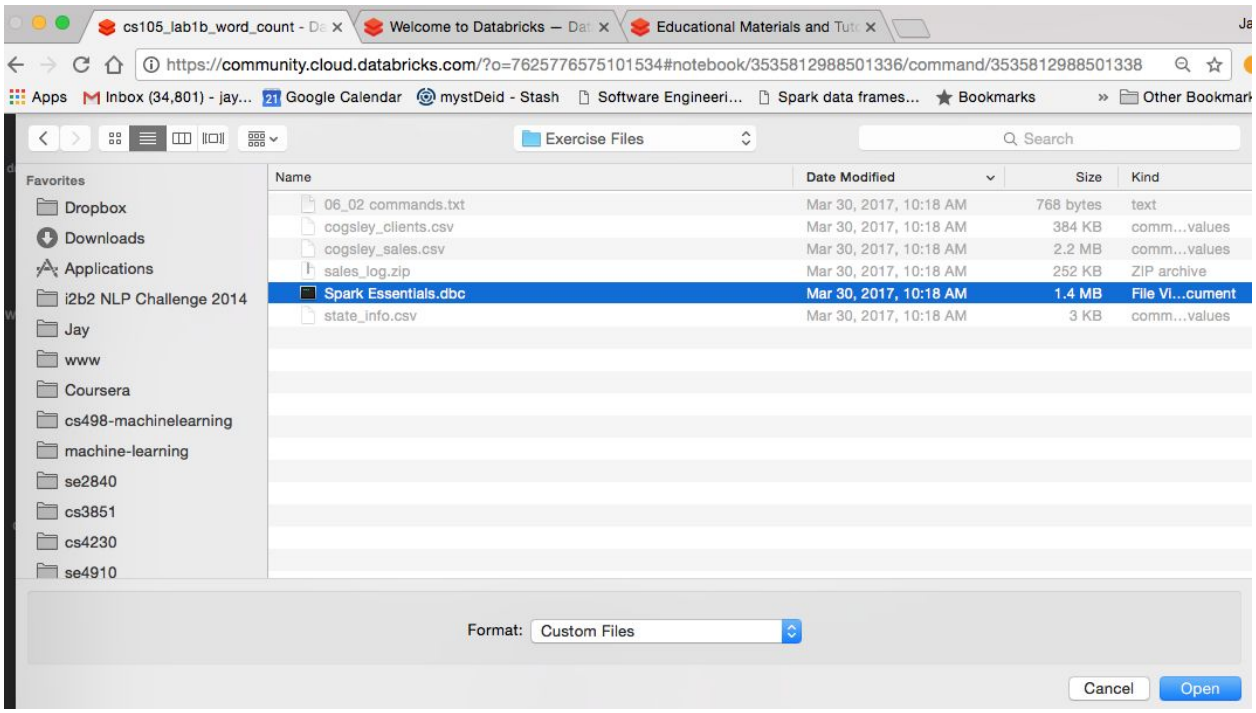
- Collection of code and markup
- Able to run Python, Scala, SQL, etc.

Select Import files

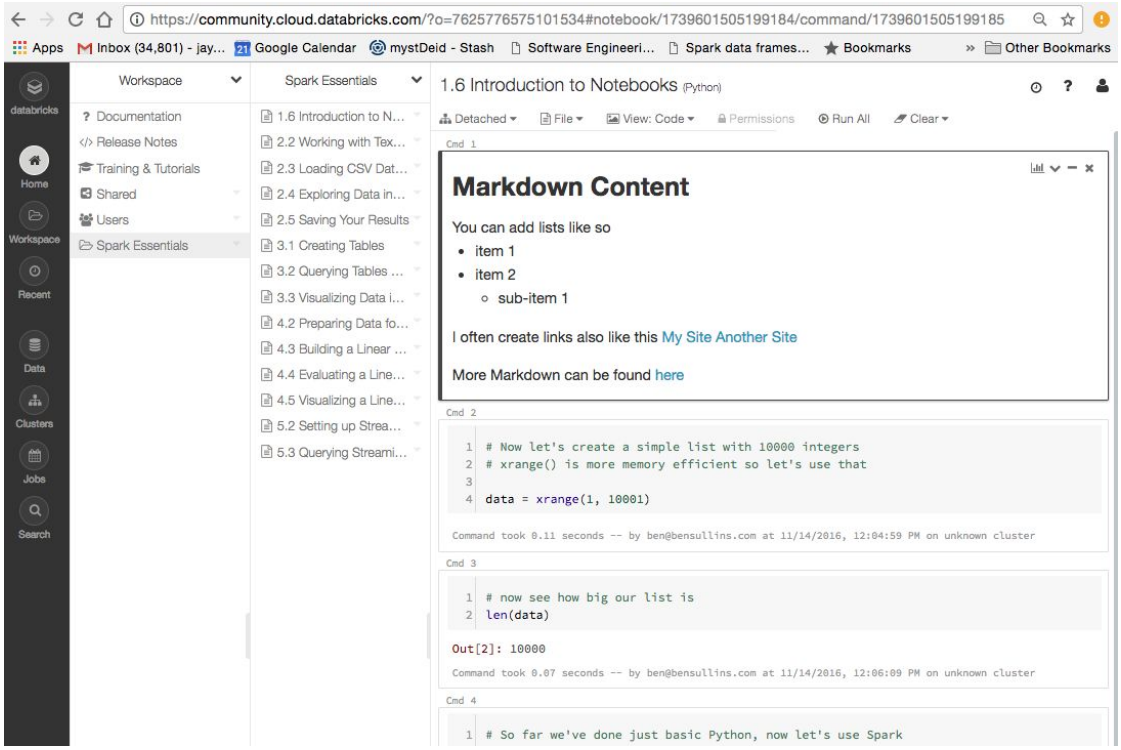
- Right mouse click on Workspace



Select “SparkTutorial.dbc” from notebooks sub-directory. Dbc is a databricks archive and will load all notebooks. The individual notebooks are also provided.



Select Home

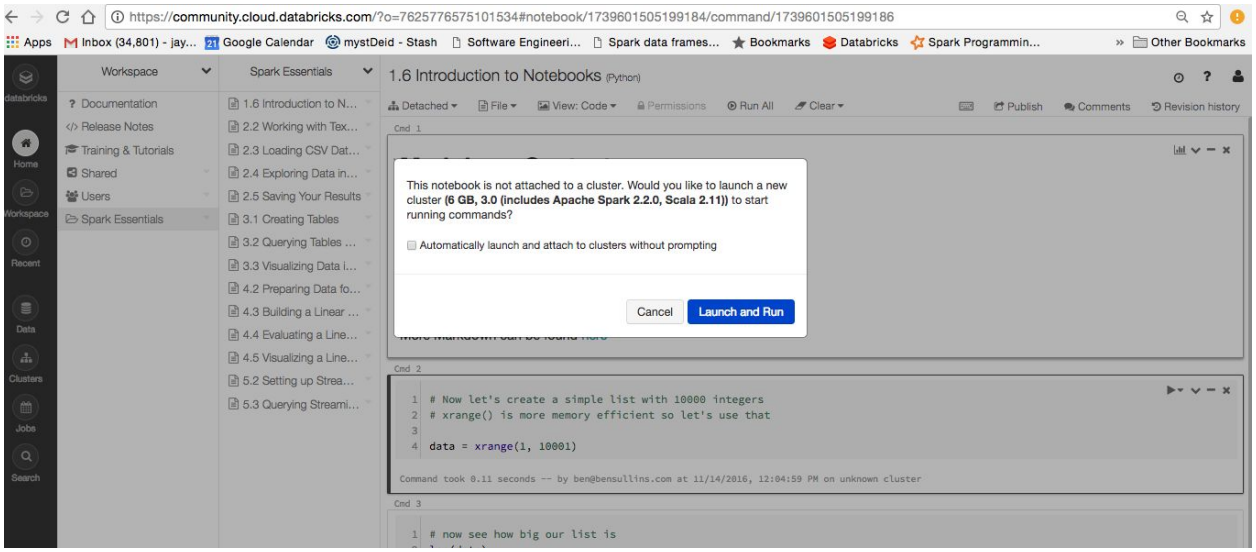


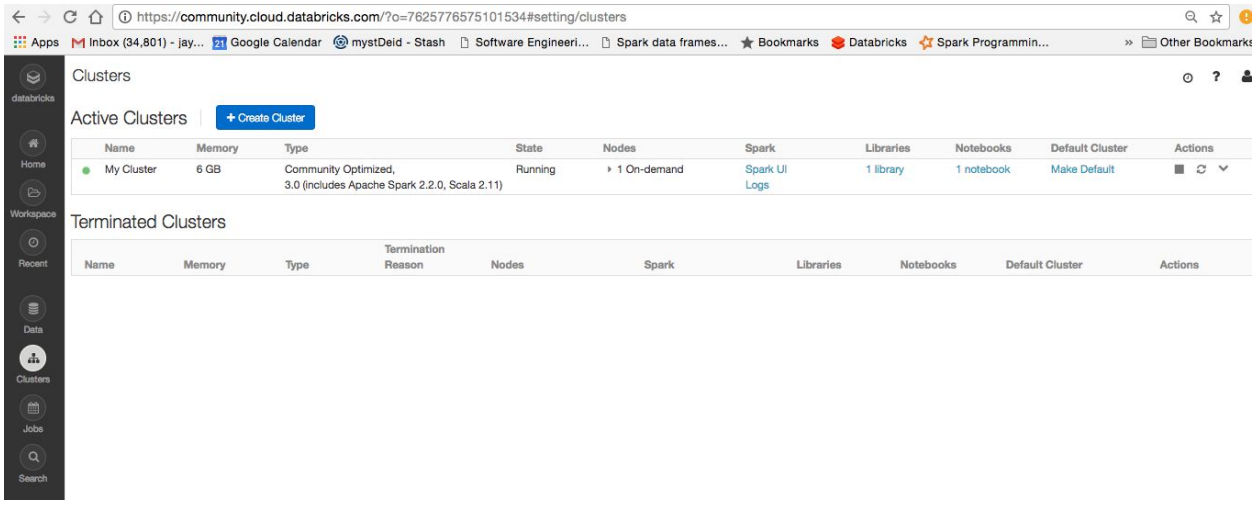
Open the introduction notebook

Execute the first programming cell. Select the cell, press shift-enter.

It will require you to start a cluster. You can stop and restart the cluster. It will shut down automatically after a period of time.

A default cluster will be provided, or you can select one. Use the default for now.





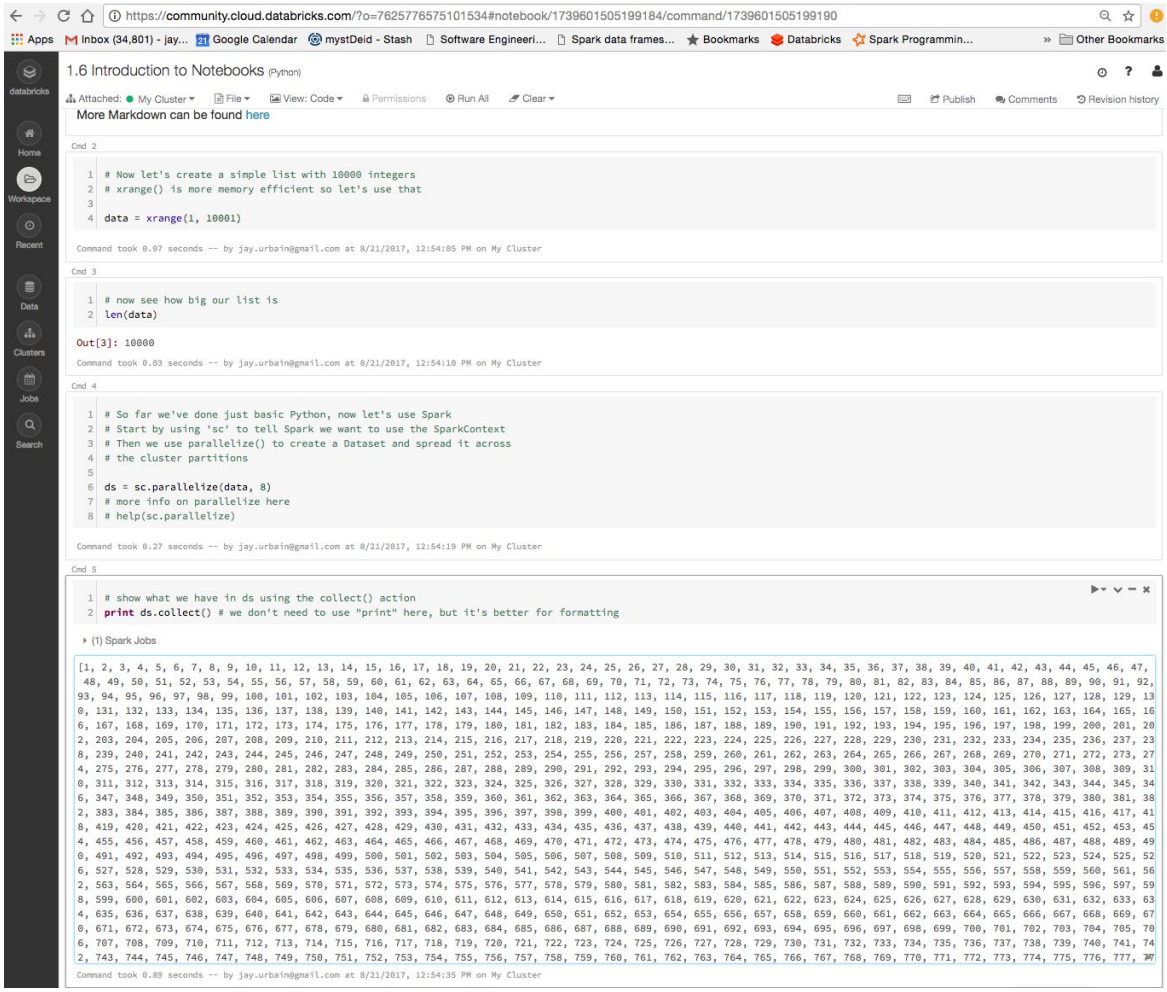
Return to workspace and the Introduction to Notebooks tutorial

- Execute each cell and read the comments
- sc is the Spark Context necessary for interacting with the Spark Framework (more later)
- ds = sc.parallelize(data, 8) creates an RDD with 8 partitions
- ds.collect() collects the entire contents of the RDD
- ds.take(10) retrieves the top ten rows in the RDD

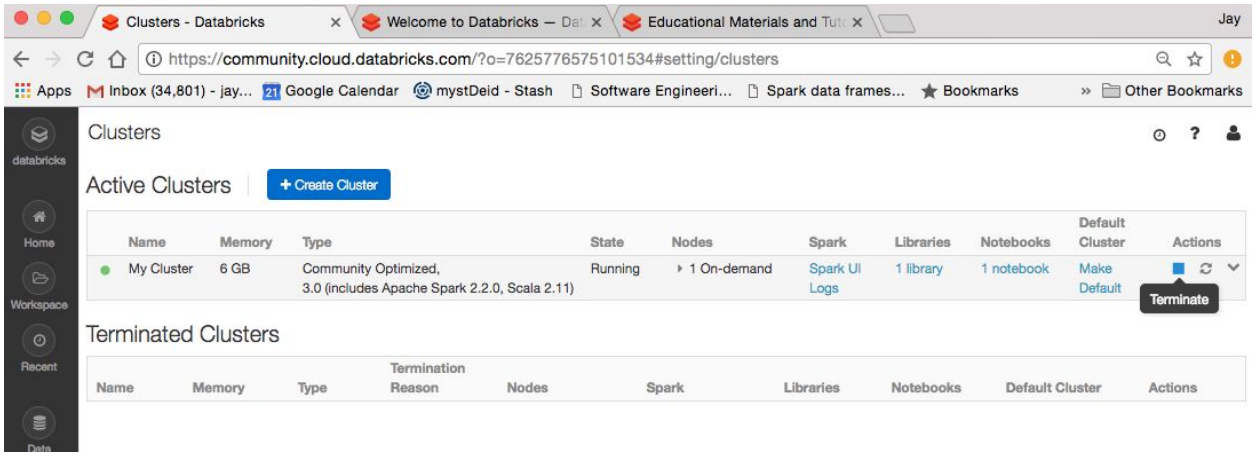
Note: the Databricks environment creates the Spark Context for you automatically. If you are not running in a Databricks environment, you will need to do the following:

```
from pyspark import SparkContext

sc = SparkContext()
```

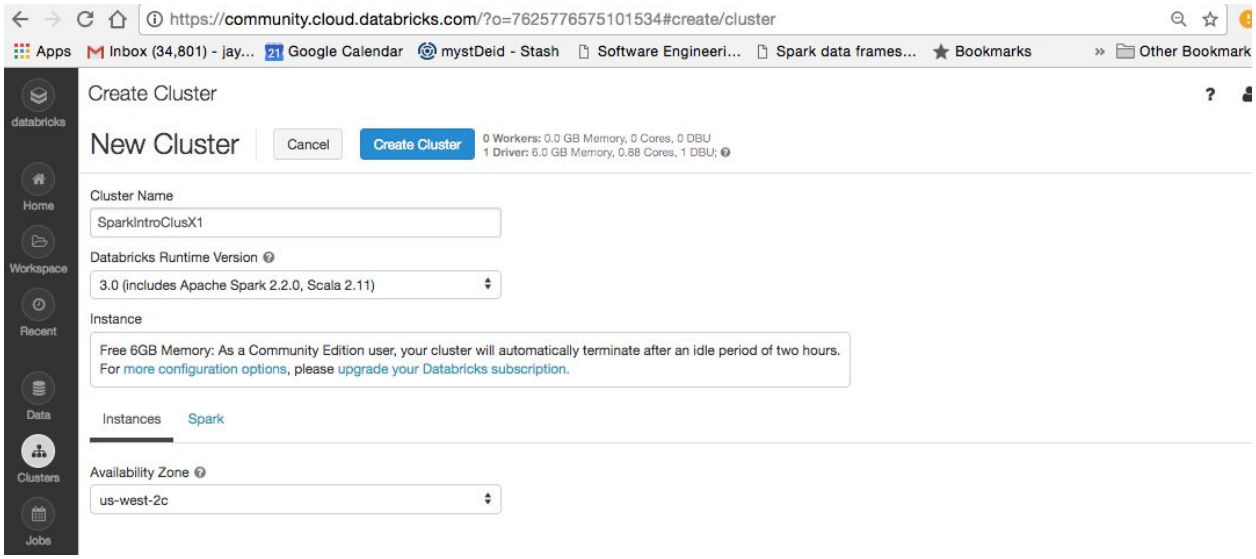
Complete the Introduction to Notebooks.



Terminate your cluster:



Create the following cluster configuration (Python 3 Spark 2.2):



Note: APIs and functionality can vary significantly between different major Spark releases.

Understanding Data in Spark

Resilient Distribute Dataset (RDD)

- Lowest level API for working with data in Spark
- Makes Spark fast with resilient distributed in-memory data caches
- Container for working with objects
- These objects can be of varying types and spread across many machines in the cluster
- In general, you will want to work with DataFrames.

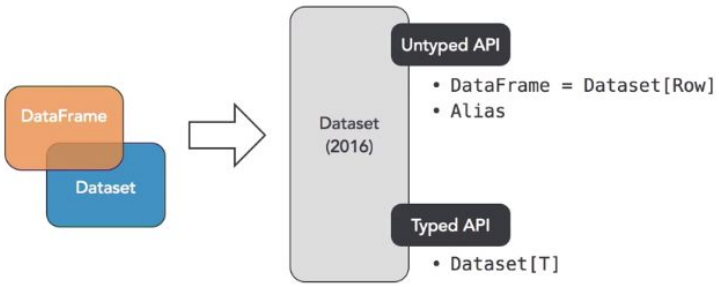
DataFrame

- Like Python Pandas dataframe or R dataframe
- Tables of data that allow you to query it
- Can use with SQL table, i.e., write SQL like queries against one or more dataframes.

Dataset

- Spark's latest way of working with data
- Combines elements of RDD's and DataFrames
- Like a strongly typed dataframe
- Dataframe and Dataset APIs were merged in Spark 2.0
- My head is exploding

Unified Apache Spark 2.0 API



Operations

- Actions- execute an operation on a computational graph
- Transformations - build operations in computational graph, oonly executed when an action is run

Actions	Transformation (L)
show	select
count	distinct
collect	groupBy
save	sum
	orderBy
	filter
	limit

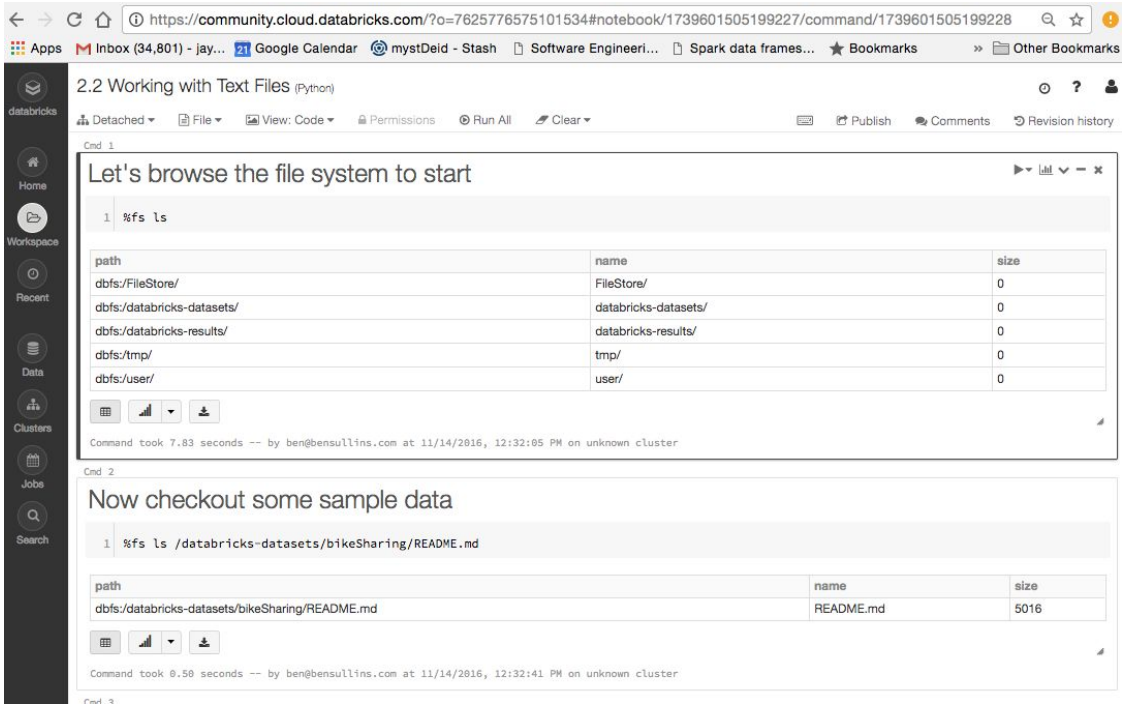
Working with Text Files

Steps:

- Find data
- Read in text file
- Read in directory
- Create dataframe

Load the “Working with Text” notebook.

Attach your notebook (Spark 2.2) to a cluster and execute the first couple of cells



%fs - is a notebook “magic command” that provides access to the file system.

Load data into an RDD

logData = sc.textFile(logFile).cache()

- Read text files into an RDD, cache keeps the data in memory
- logData is an RDD

sc.textFile(path)

- creates an RDD from the text file

numBikes = logData.filter(lambda s: 'bike' in s.lower()).count()

- converts each line to lower case
- Counts the number of lines containing ‘bike’
- Lambda is an anonymous function: for each line convert to lowercase, if ‘bike’ in line, return the line.

filenames = files.toDF(['name', 'data'])

- Converts RDD to dataframe

Test snippet:

```
rd = sc.parallelize(["bike to town", "bike to bike", "dog ate food", "dog ate bike"],5)
rd.collect()
rd.filter(lambda s: 'bike' in s.lower()).count()
```

Note:

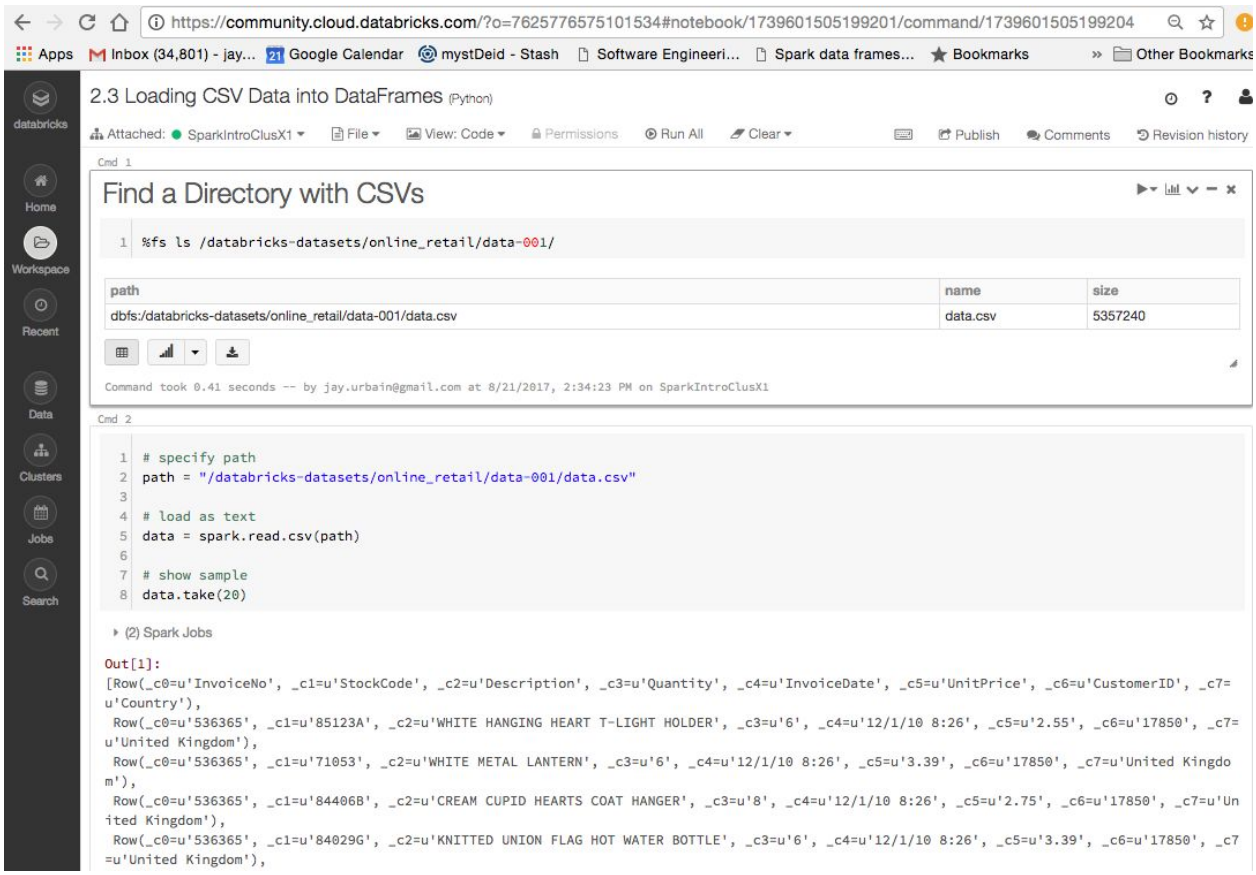
- dbfs is for DataBricks file system. For hadoop you would preface file with hdfs
- display() used in these notebooks is a special Databricks function. You can use standard print() or matplotlib plotting functions in other environments.

Loading CSV data into DataFrames

Open the Loading CSV Data into DataFrames notebook

Attach a cluster

Execute the first couple of cells



data = spark.read.csv(path)

- Reads a comma separated file directly into a *Dataframe*

df = spark.read.load(path,
format='com.databricks.spark.csv',

```
        header='true',
        inferSchema='true')
```

- Reads in csv file, uses existing header, infers schema

Use df.printSchema() to review the dataframe schema.

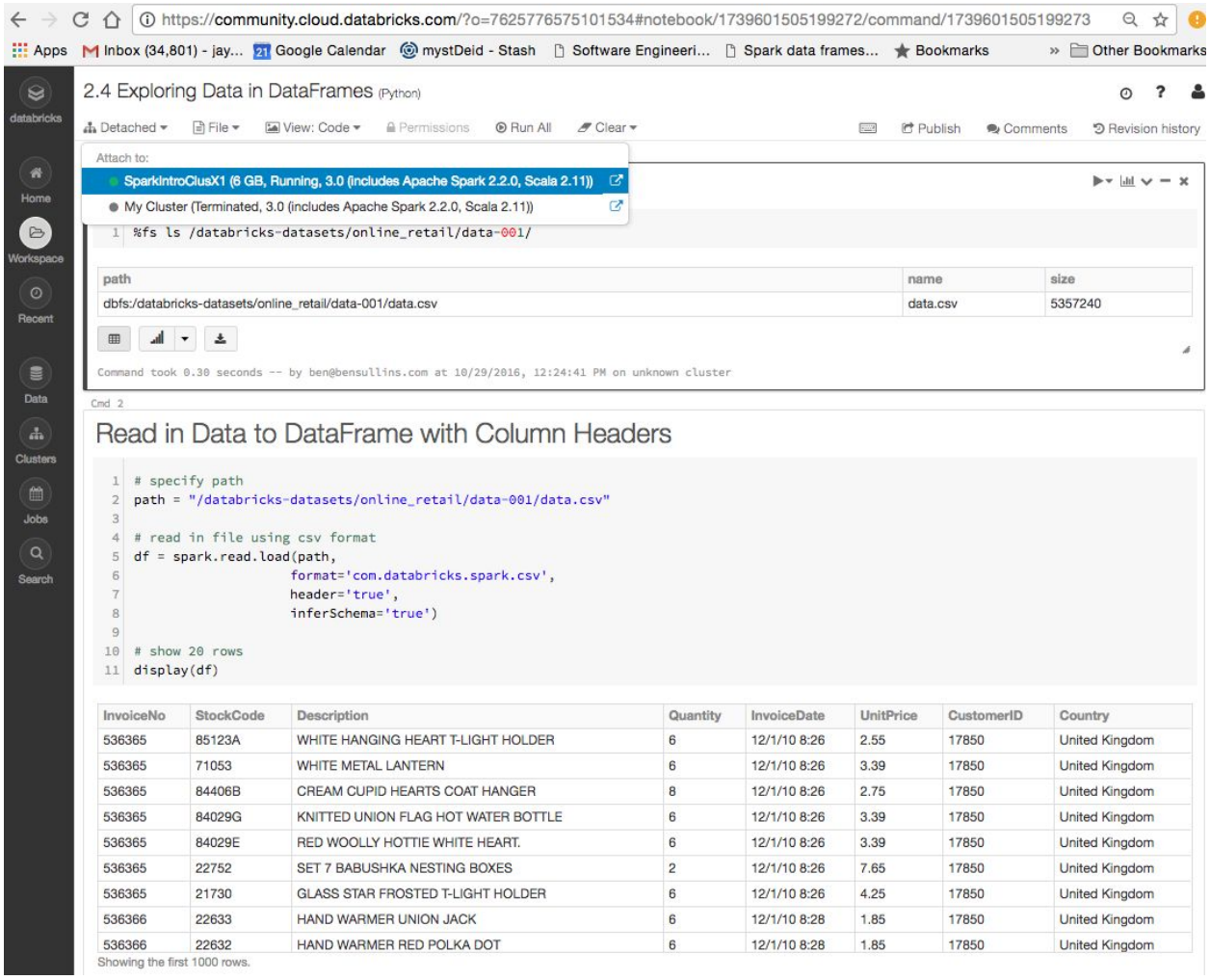
```
root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: integer (nullable = true)
 |-- Country: string (nullable = true)
```

```
df
 .select("Country") # chooses just the 1 column
 .distinct() # removes duplicates
 .orderBy("Country") # sorts results in ascending
```

- Equivalent to: select distinct Country from df order by Country

Exploring data in DataFrames

Load “Exploring Data in DataFrames” notebook




```
df = spark.read.load(path,
                     format='com.databricks.spark.csv',
                     header='true',
                     inferSchema='true')
```

- Load csv into Dataframe

```
df.printSchema()
```

- Displays inferred schema

```
df.select("Country").show()
```

- SQL: select Country from df;

```
df .select(df["InvoiceNo"],df["UnitPrice"]*df["Quantity"])
   .groupBy("InvoiceNo")
   .sum()
```

- select InvoiceNo, sum(UnitPrice*Quantity) from df group by InvoiceNo

```
df.filter(df["InvoiceNo"]==536596).show()
```

- select * from df where InvoiceNo=536596

```
df .select(df["Country"], df["Description"],(df["UnitPrice"]*df["Quantity"]).alias("Total"))
   .groupBy("Country", "Description")
   .sum()
   .filter(df["Country"]=="United Kingdom")
   .sort("sum(Total)", ascending=False)
   .limit(10)
```

- select Country, Description, UnitPrice*Quantity as Total
  from df
  where Country = “United Kingdom”
  order by sum(Total) desc
  limit 10;

Saving your results

DataFrameWriter

- Save df as table in Spark.

Load “Saving Your Results” notebook

```
df = spark.read.load(path,
                     format='com.databricks.spark.csv',
                     header='true',
                     inferSchema='true')
```

- read csv

```
df
   .select(df["Country"], df["Description"],(df["UnitPrice"]*df["Quantity"]).alias("Total"))
   .groupBy("Country", "Description")
   .sum()
   .filter(df["Country"]=="United Kingdom")
   .sort("sum(Total)", ascending=False)
```

- select Country, Description, UnitPrice*Quantity as Total
  from df
  where Country = “United Kingdom”
  order by sum(Total) desc
  limit 10;

```
r1 = df.select(df["Country"], df["Description"],(df["UnitPrice"]*df["Quantity"]).alias("Total"))
```

- select Country, Description, UnitPrice*Quantity as Total
  from df

```
r1.write.saveAsTable("product_sales_by_country")
```

- Save as table in datastore Hive

Save as csv (2.0 Spark):

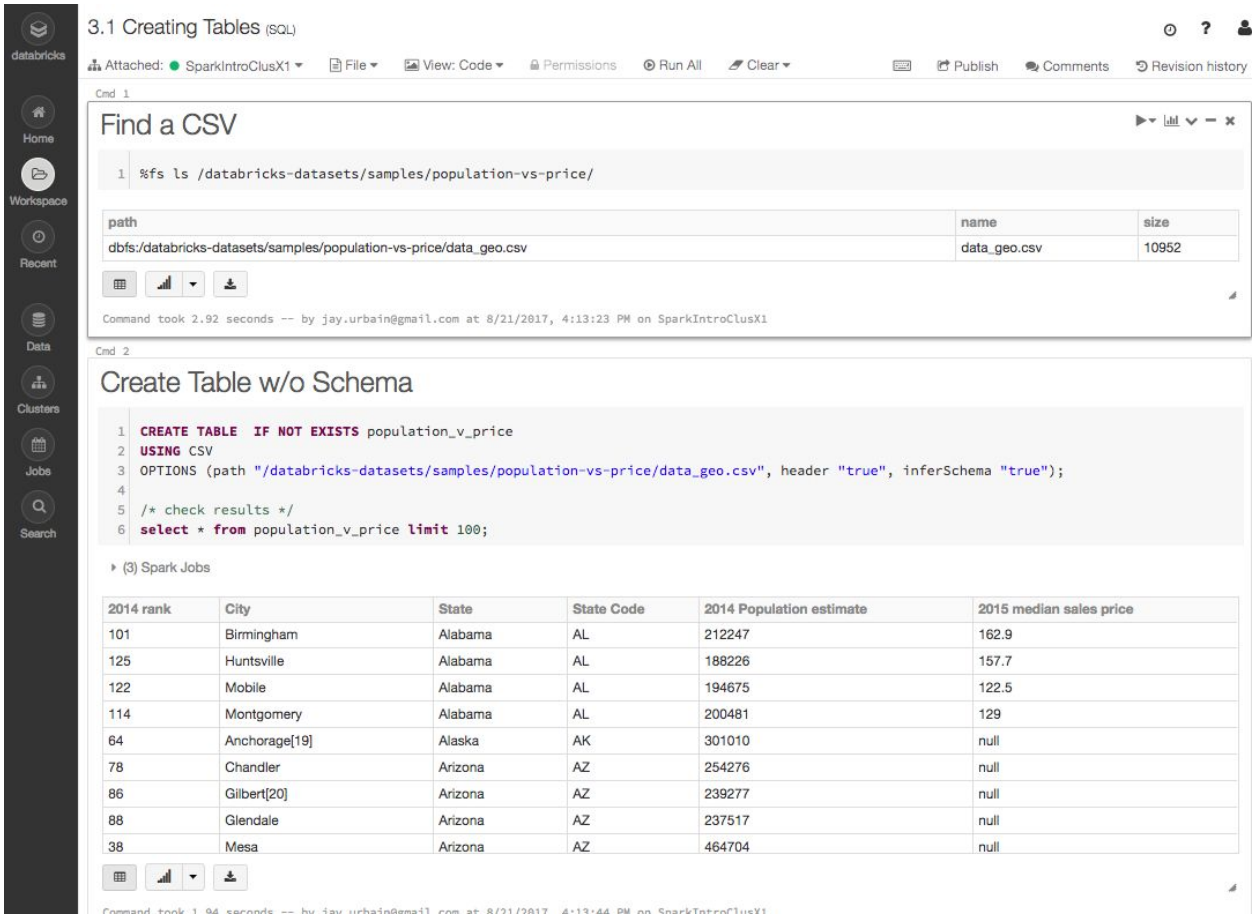
```
r1.write.csv(“my.csv”)
```

Using Spark SQL to Analyze Data

Create table syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] [db_name.]table_name
[(col_name1[:] col_type1 [COMMENT col_comment1], ...)]
USING datasource [OPTIONS (key1=val1, key2=val2, ...)]
[PARTITIONED BY (col_name1, col_name2, ...)]
[CLUSTERED BY (col_name3, col_name4, ...) INTO num_buckets BUCKETS]
[AS select_statement]
```

Load the “Creating Tables” SQL notebook:



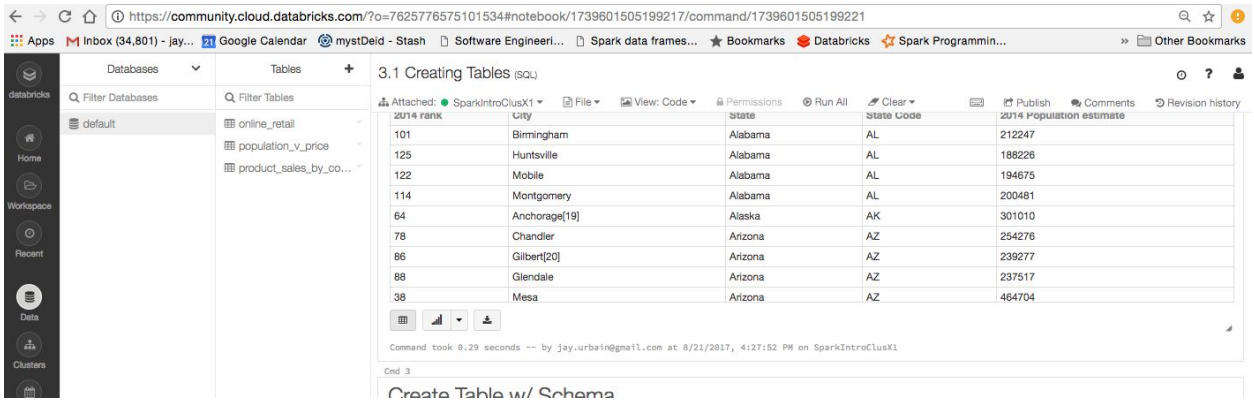
Create table by inferring schema:

```
CREATE TABLE  IF NOT EXISTS population_v_price
USING CSV
OPTIONS (path "/databricks-datasets/samples/population-vs-price/data_geo.csv", header
"true", inferSchema "true");

/* check results */
select * from population_v_price limit 100;
```

Create table with schema:

```
CREATE TABLE IF NOT EXISTS online_retail(
InvoiceNo string,
StockCode string,
Description string,
Quantity int,
InvoiceDate string,
UnitPrice double,
CustomerID int,
Country string)
USING CSV
OPTIONS (path "/databricks-datasets/online_retail/data-001/data.csv", header "true");

/* check results */
select * from online_retail limit 100;
```

Verify table exists in workspace:



Example of defining your own schema (note in notebook)

Read headerless tab-delimited text file into RDD and defining your own schema

```
# Import SQLContext and data types
from pyspark.sql import SQLContext, HiveContext
from pyspark.sql.types import *
from datetime import datetime
import time
from pyspark.sql.functions import *

# sc is an existing SparkContext.
sqlContext = HiveContext(sc)

# Load a text file and convert each line to a tuple
lines = sc.textFile("hdfs:///crdw//patient_mother_06072017.tsv")
parts = lines.map(lambda l: l.split("\t"))
patient_mother = parts.map(lambda p: (p[0], p[1], int(p[2]), p[3], p[4], p[5], p[6], p[7], p[8].strip() )
)

fields = [StructField("PAT_ID", StringType(), True),
          StructField("PAT_ID_ENC", StringType(), True),
          StructField("DATE_OFF", IntegerType(), True),
          StructField("PATIENT_NUM", StringType(), True),
          StructField("PAT_ENC_CSN_ID", StringType(), True),
          StructField("ENCOUNTER_NUM", StringType(), True),
          StructField("PAT_FIRST_NAME", StringType(), True),
          StructField("PAT_LAST_NAME", StringType(), True),
          StructField("PAT_MIDDLE_NAME", StringType(), True)]
schema = StructType(fields)

# Apply the schema to the RDD.
schema_patient_mother = sqlContext.createDataFrame(patient_mother, schema)
schema_patient_mother.columns
schema_patient_mother.printSchema()

# Register the DataFrame as a table.

sqlContext.sql("DROP TABLE IF EXISTS schema_patient_mother")
sqlContext.registerDataFrameAsTable(schema_patient_mother, "schema_patient_mother")

sqlContext.sql("SELECT * FROM schema_patient_mother limit 100").show(5)
```

Note: How to Convert dataframe to table using in Python configured (not SQL) notebook:

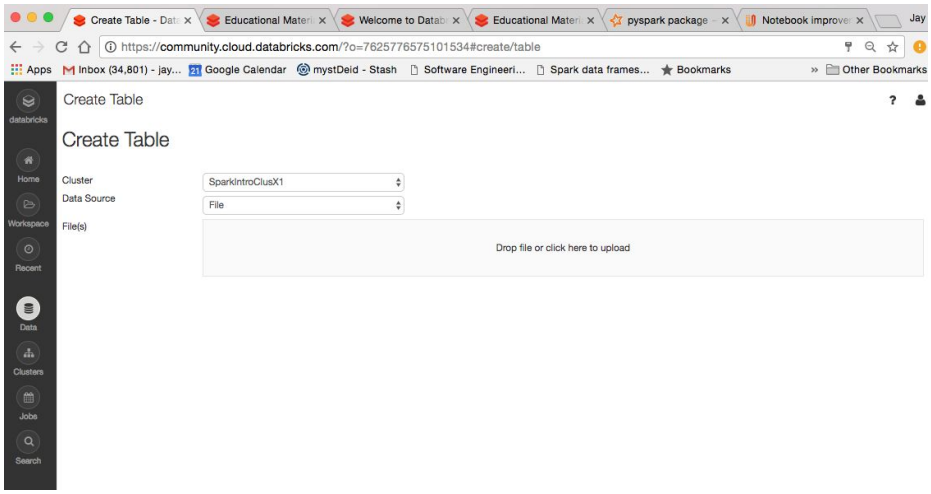
Register the DataFrame as a table.

sc.sql("DROP TABLE IF EXISTS schema_patient_mother")
sc.registerDataFrameAsTable(schema_patient_mother, "schema_patient_mother")

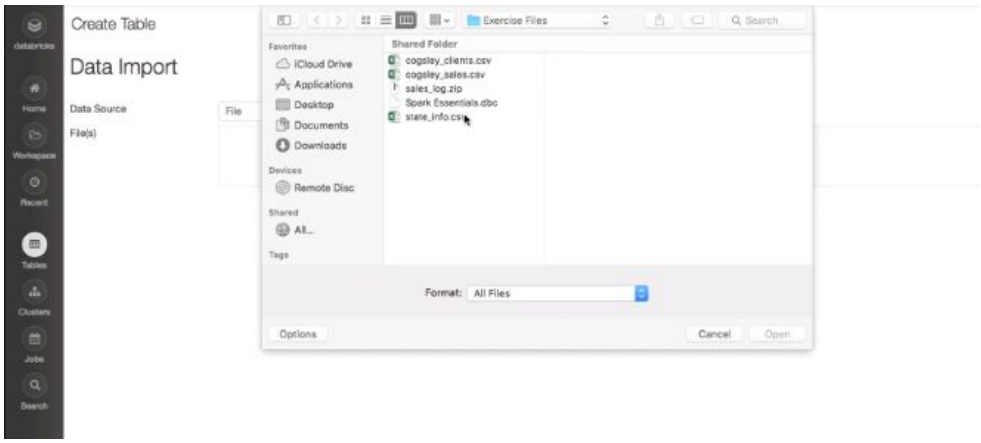
sc.sql("SELECT * FROM schema_patient_mother limit 100").show(5)

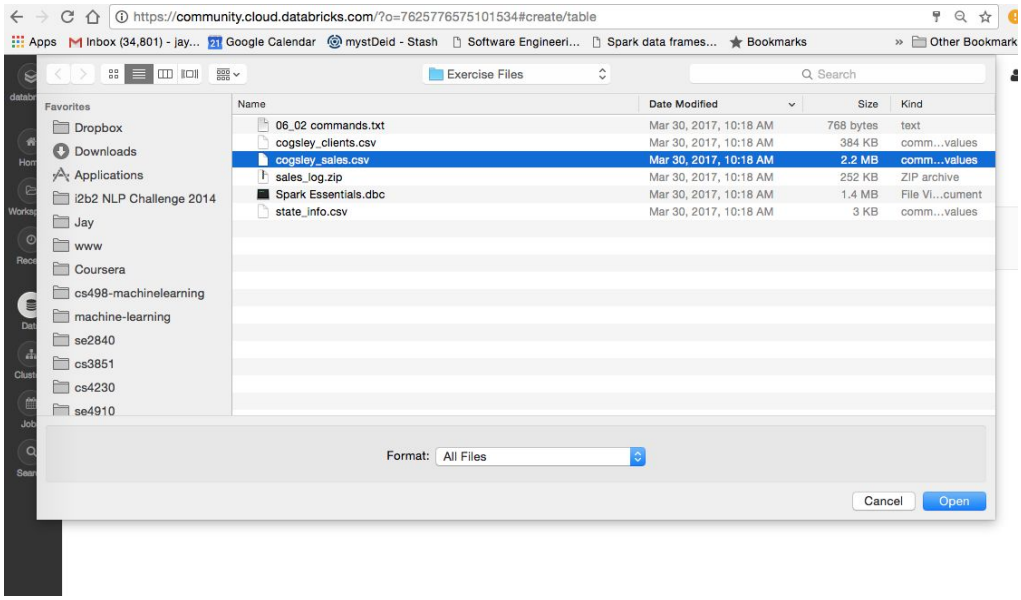
Manually creating a table from a csv file:

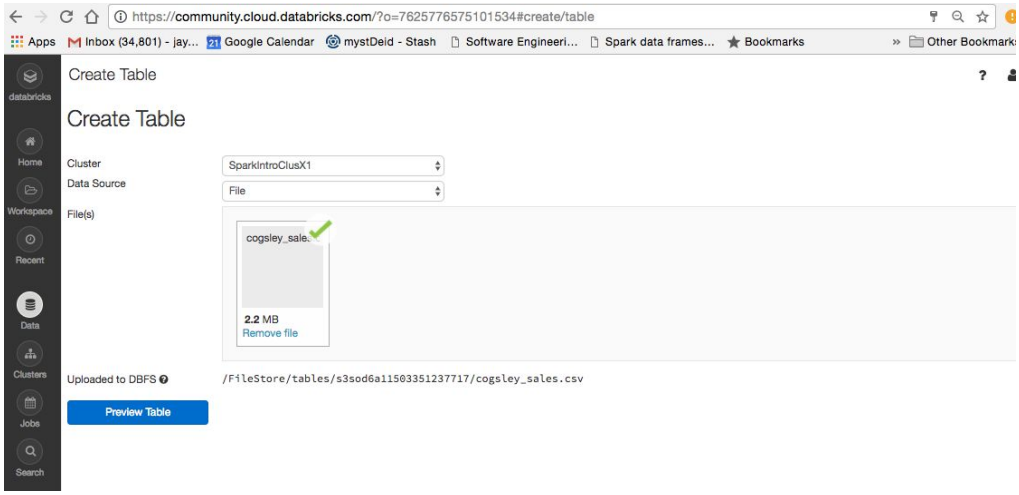
Select Data -> Tables -> '+'



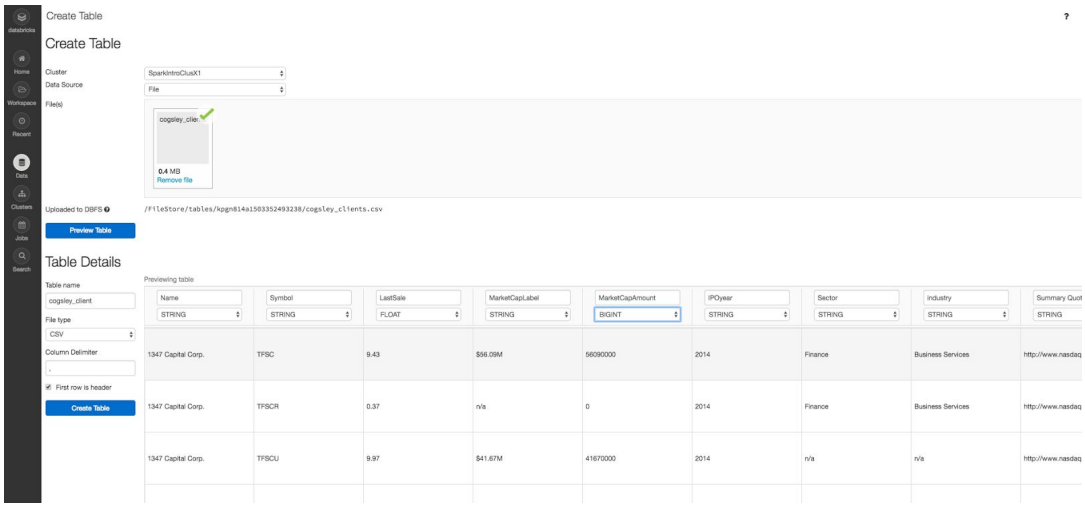
Note: data is in tutorial data sub-directory.



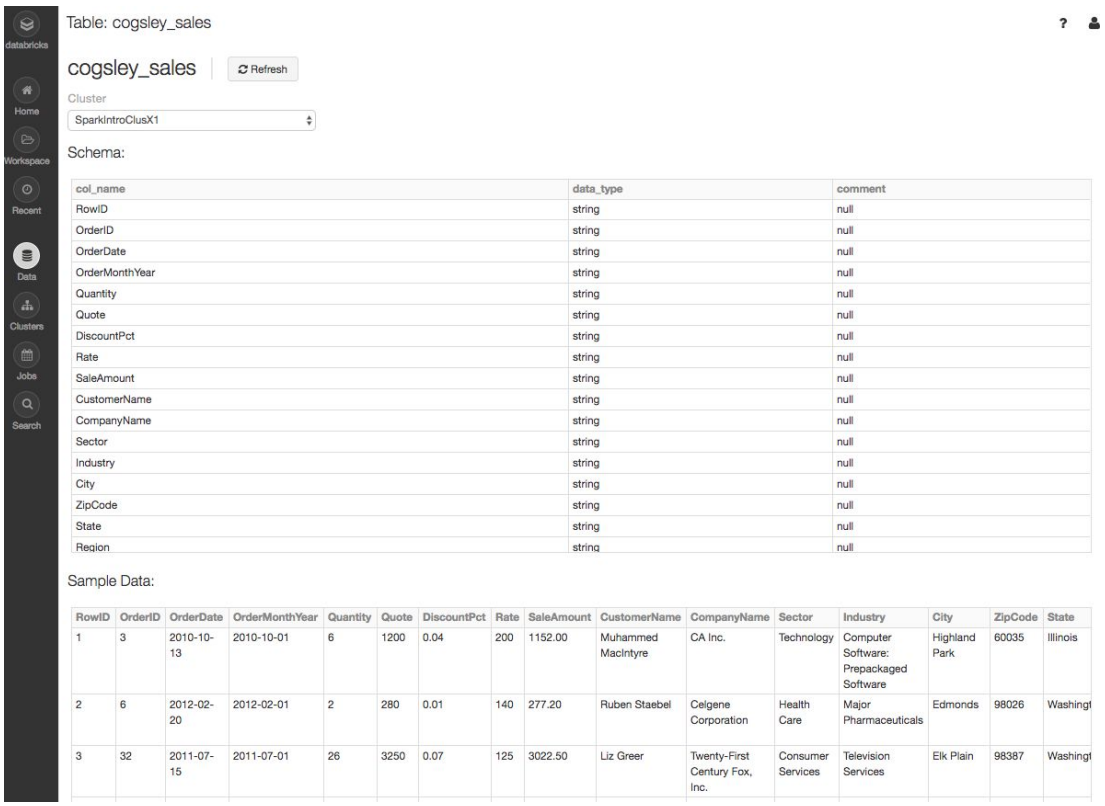

https://community.cloud.databricks.com/?o=7625776575101534#create/table
Apps
Inbox (34,801) - jay...
Google Calendar
mystDeid - Stash
Software Engineeri...
Spark data frames...
Bookmarks
Other Bookmark
Exercise Files
Search
Favorites
Dropbox
Downloads
Applications
i2b2 NLP Challenge 2014
Jay
www
Coursera
cs498-machinelearning
machine-learning
se2840
cs3851
cs4230
se4910
Name
Date Modified
Size
Kind
06_02 commands.txt
Mar 30, 2017, 10:18 AM
768 bytes
text
cogsley_clients.csv
Mar 30, 2017, 10:18 AM
384 KB
comm...values
cogsley_sales.csv
Mar 30, 2017, 10:18 AM
2.2 MB
comm...values
sales_log.zip
Mar 30, 2017, 10:18 AM
252 KB
ZIP archive
Spark Essentials.dbc
Mar 30, 2017, 10:18 AM
1.4 MB
File Vi...cument
state_info.csv
Mar 30, 2017, 10:18 AM
3 KB
comm...values
Format:
All Files
Cancel
Open


https://community.cloud.databricks.com/?o=7625776575101534#create/table
Apps
Inbox (34,801) - jay...
Google Calendar
mystDeid - Stash
Software Engineeri...
Spark data frames...
Bookmarks
Other Bookmark
databricks
Home
Workspace
Recent
Data
Clusters
Jobs
Search
Create Table
Create Table
Cluster
SparkIntroClusX1
Data Source
File
File(s)
cogsley_sale
2.2 MB
Remove file
Uploaded to DBFS
/FileStore/tables/s3sod6a1150351237717/cogsley_sales.csv
Preview Table

Preview table, provide name and select *First row as header.* Adjust String to FLOAT or BIGINT where appropriate.

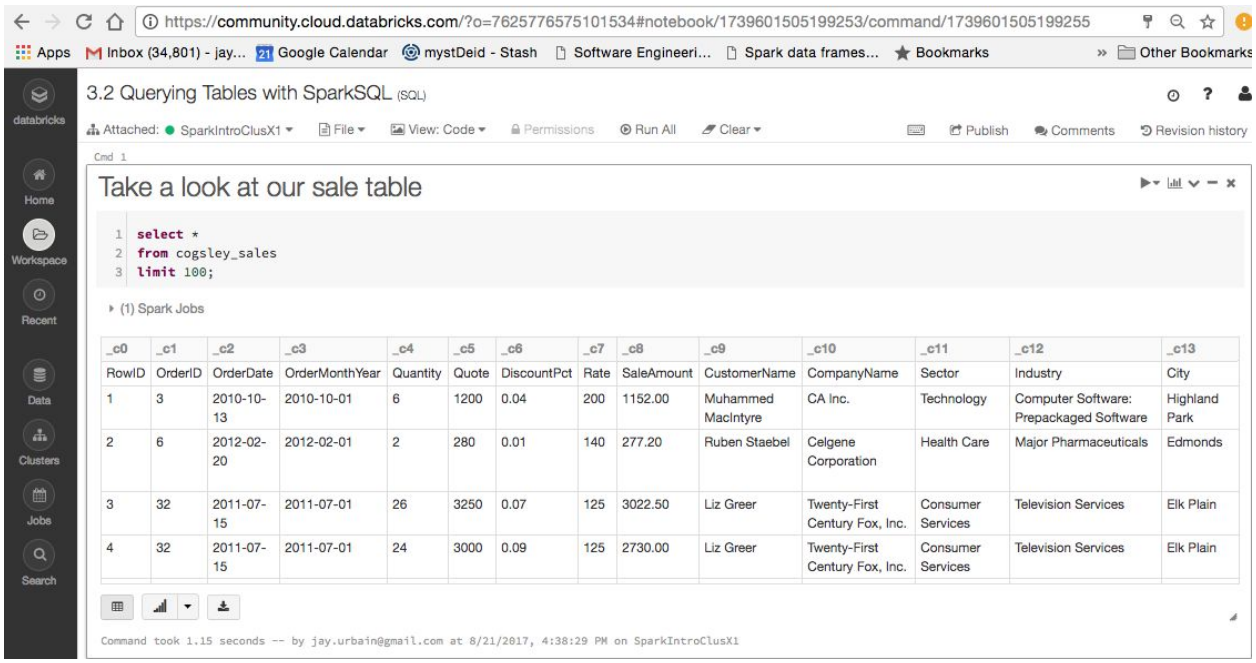


Select Create Table



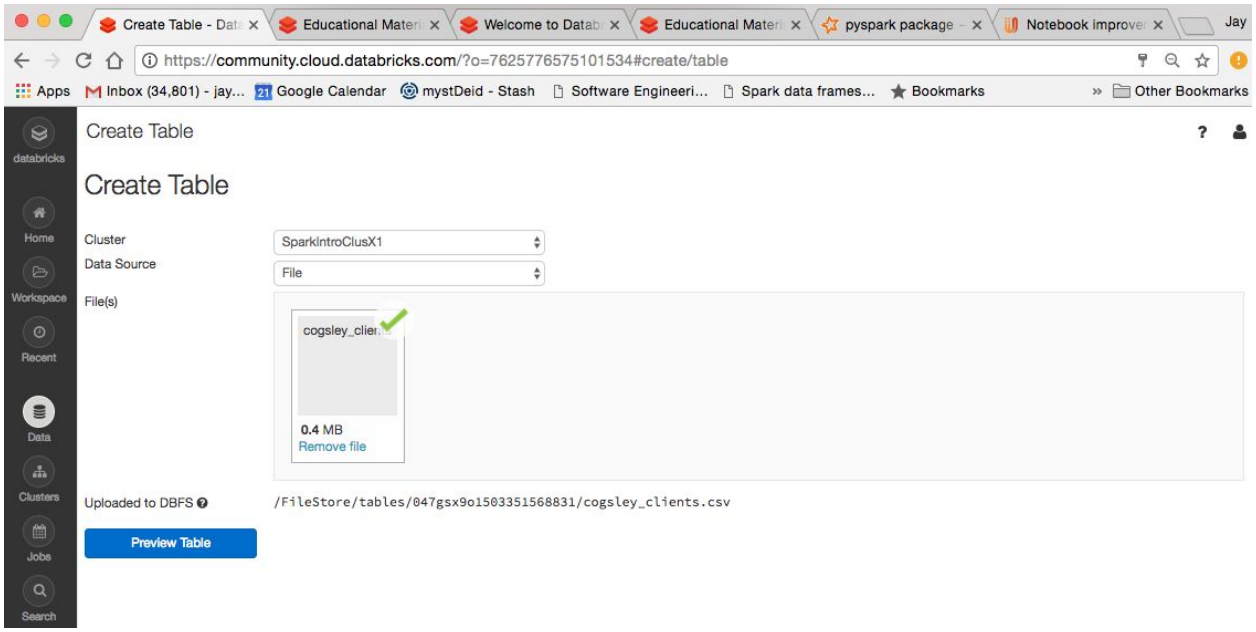
Querying Tables with SparkSQL

Load the “Querying Tables with SparkSQL” SQL notebook

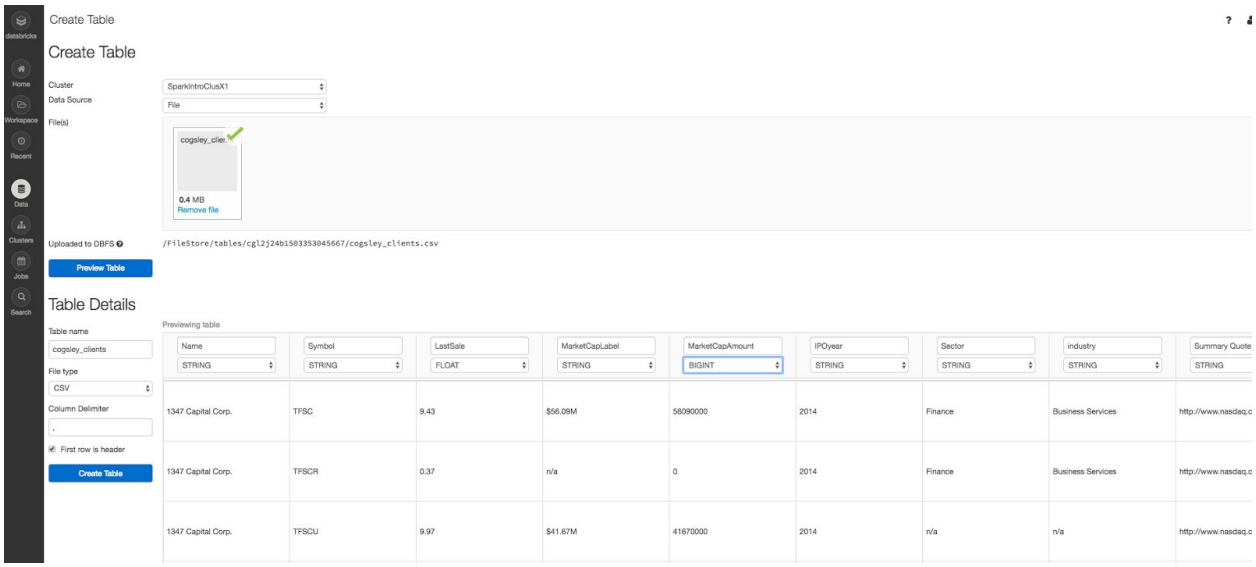


_c0	_c1	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9	_c10	_c11	_c12	_c13
RowID	OrderID	OrderDate	OrderMonthYear	Quantity	Quote	DiscountPct	Rate	SaleAmount	CustomerName	CompanyName	Sector	Industry	City
1	3	2010-10-13	2010-10-01	6	1200	0.04	200	1152.00	Muhammed MacIntyre	CA Inc.	Technology	Computer Software: Prepackaged Software	Highland Park
2	6	2012-02-20	2012-02-01	2	280	0.01	140	277.20	Ruben Staebel	Celgene Corporation	Health Care	Major Pharmaceuticals	Edmonds
3	32	2011-07-15	2011-07-01	26	3250	0.07	125	3022.50	Liz Greer	Twenty-First Century Fox, Inc.	Consumer Services	Television Services	Elk Plain
4	32	2011-07-15	2011-07-01	24	3000	0.09	125	2730.00	Liz Greer	Twenty-First Century Fox, Inc.	Consumer Services	Television Services	Elk Plain

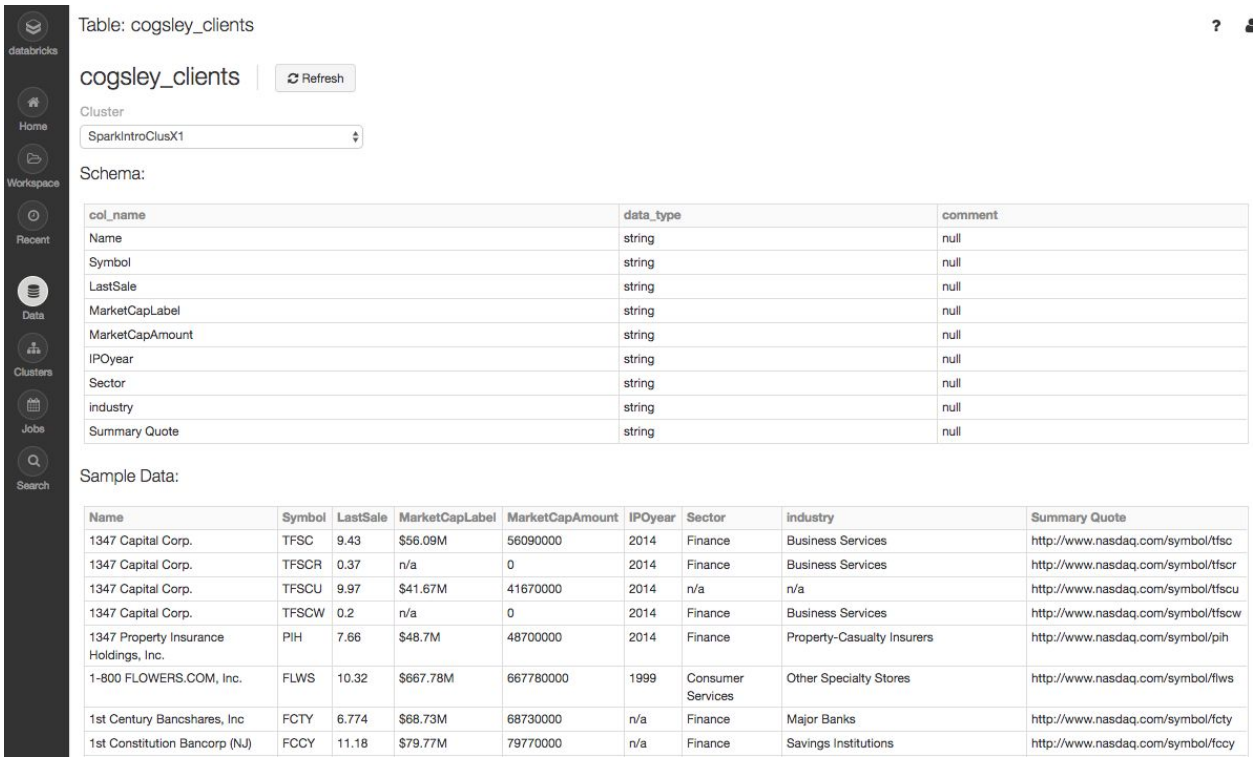
Load the cogsley_clients.csv into a table:



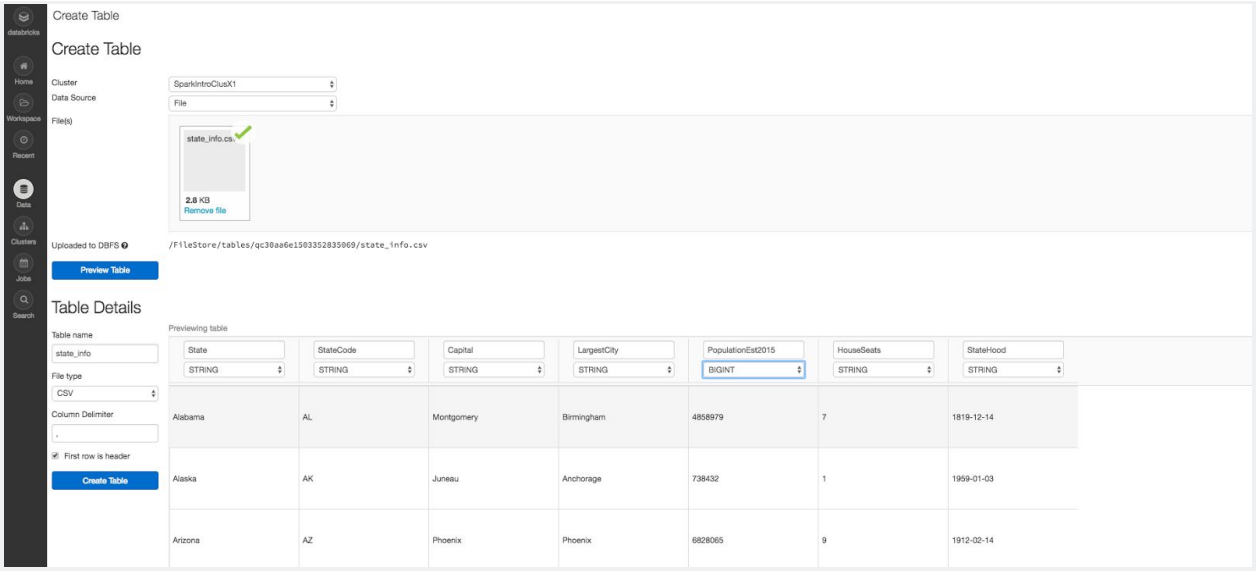
Preview the table, provide name and select First row is header.



Create table.



Create state_info table:



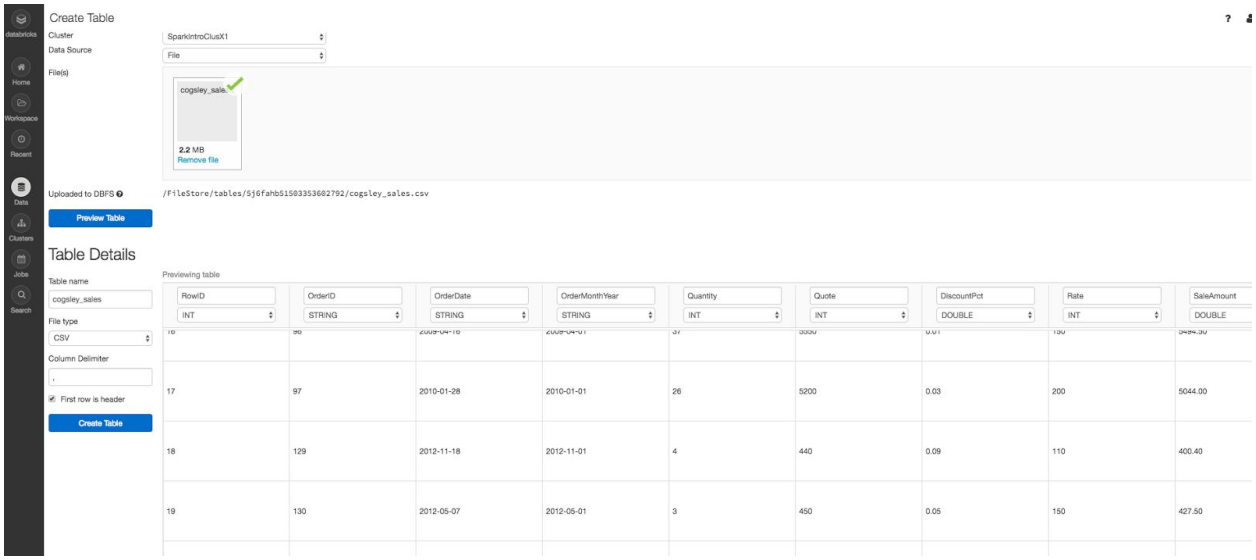
Note: refresh and clear notebook if the queries do not work.

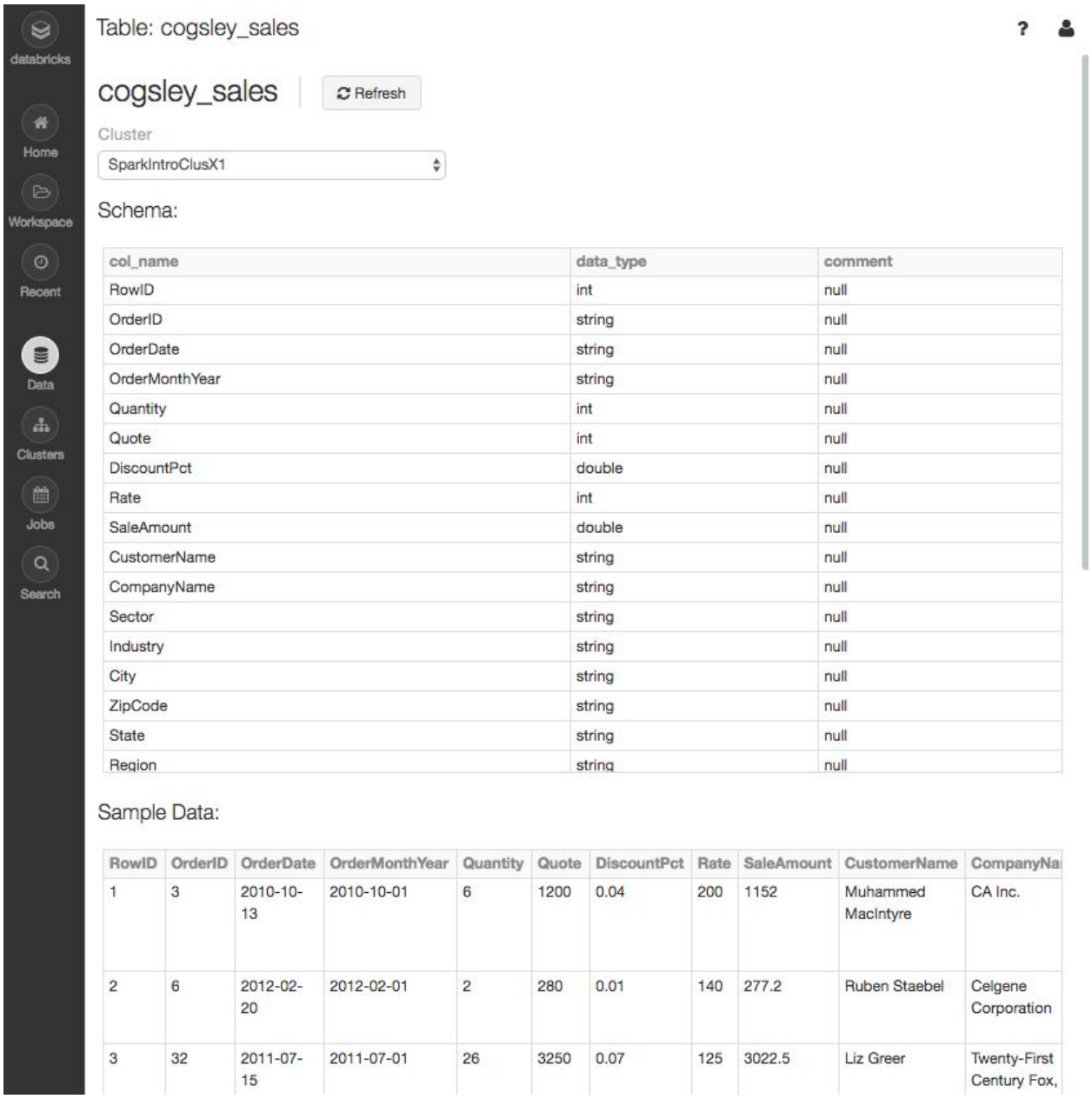
Complete the cells that illustrate table joins.

Visualizing data in Databricks notebooks

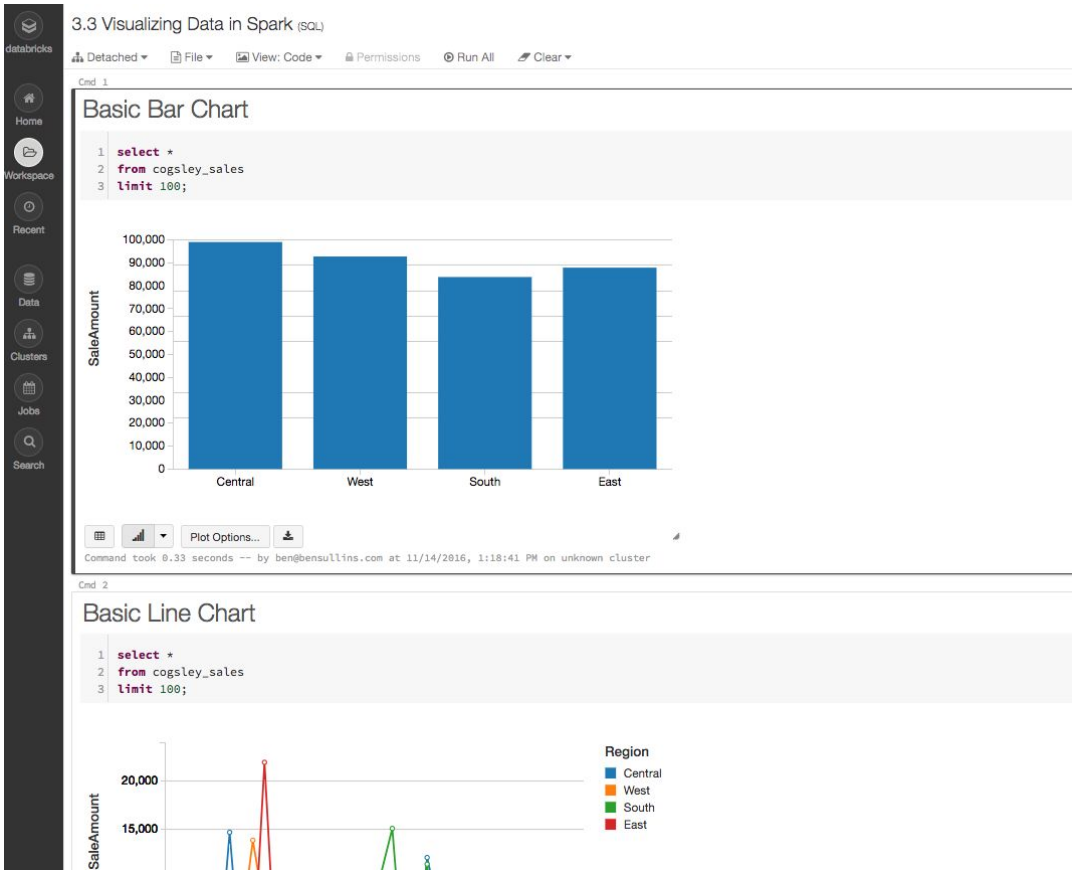
Convert text-based data into graphical representations to better inform

Delete and re-create the cogsley_sales table. Make sure all numeric columns are properly typed to int or float.

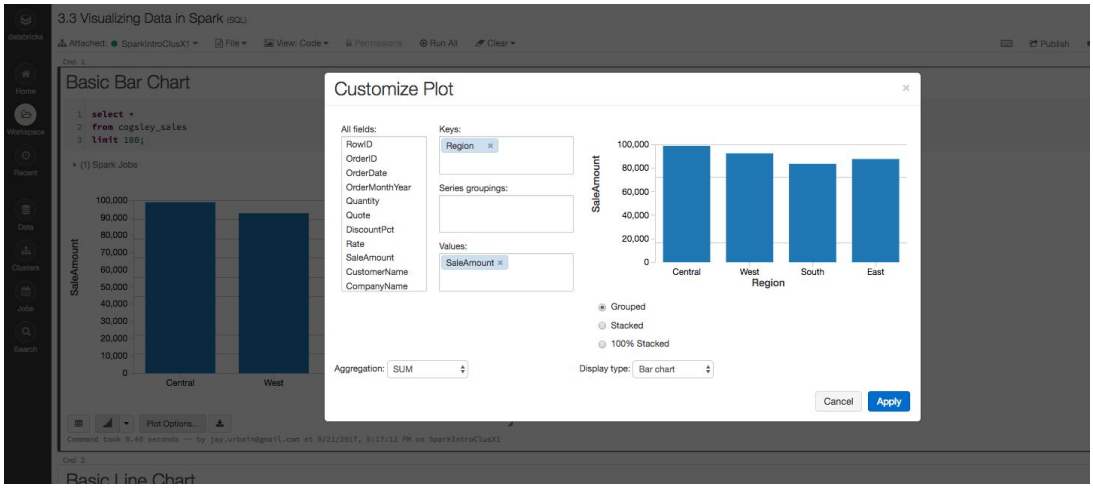




Load the “Visualizing Data in Spark” notebook.



Can adjust plot parameters dynamically by selecting options at the bottom of each plot.



Note: Note: this is Databricks specific functionality.
In non-databricks environments you need to use matplotlib or similar python package.

Running Machine Learning Algorithms Using MLib

Introduction to MLib with Spark

Machine Learning

- Repeatable and automated processes for producing expected output from given data.
- Typically used to find hidden patterns in data or make predictions.

Types

- Supervised - build predictive models from labeled training data
- Unsupervised - find structure/patterns in unlabeled data

Supervised learning classification

- Given labeled data
- Categorize data into groups
- Spam vs. not spam
- Groups provided by user

Supervised learning regression

- Variable relationship
- Prediction and forecasting
- Given answer build model to predict answer

Unsupervised learning clustering

- Inputs into groups
- Groups are not known

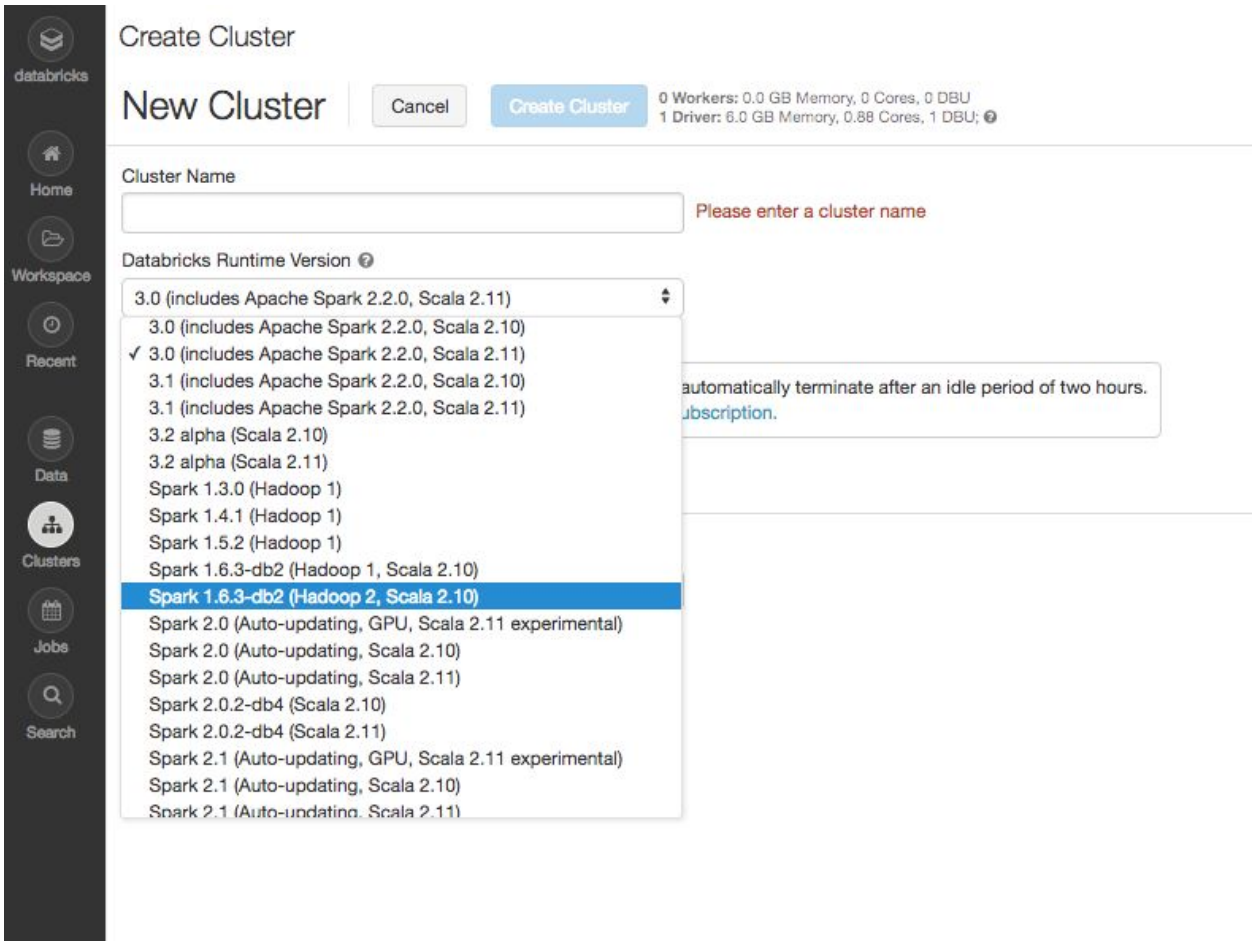
Preparing data for machine learning

- Download external data
- Read and cleanse data
- Aggregate and convert
- Build features and labels

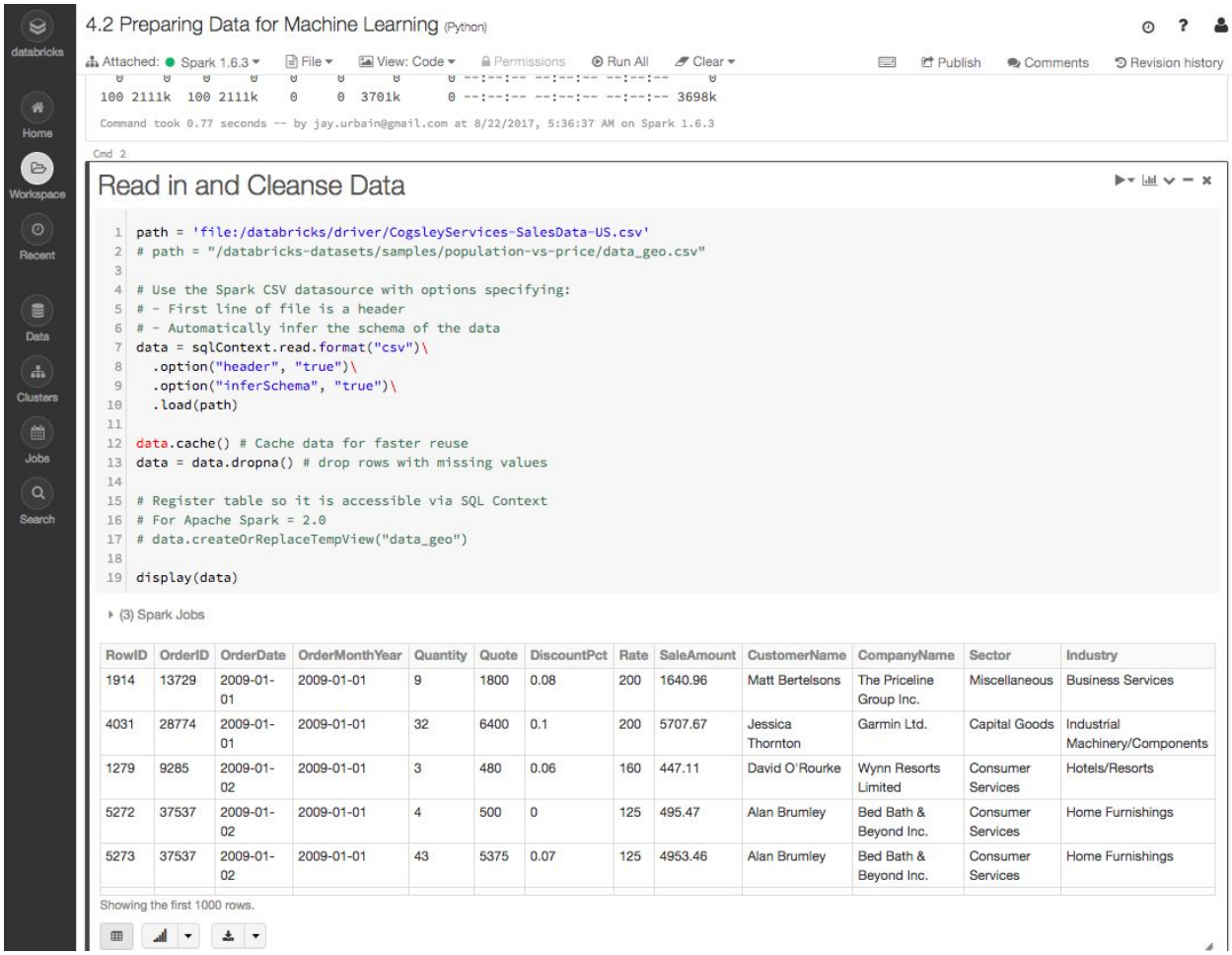
Download external data

Load the “Preparing Data for Machine Learning” notebook

Select a 1.6 version of Spark. All of the machine learning models in 1.6 have not yet been ported to 2.2+



Attach to your 1.6.3 Cluster



data.cache() # Cache data for faster reuse

- Keep in memory

data = data.dropna() # drop rows with missing values

- Drop missing values

summary = data.select("OrderMonthYear", "SaleAmount").groupBy("OrderMonthYear").sum().orderBy("OrderMonthYear").toDF("OrderMonthYear","SaleAmount")

- select OrderMonthYear, SaleAmount from data group by OrderMonthYear order by OrderMonthYear

summary.printSchema()

- Display columns and types

# Convert OrderMonthYear to integer type
results = summary.map(lambda r: (int(r.OrderMonthYear.replace('-','')), r.SaleAmount)).toDF(["OrderMonthYear","SaleAmount"])

- For each row, remove hyphens from OrderMonthYear and convert to integer
- Note: This operation is not compatible in 2.2

```
from pyspark.mllib.regression import LabeledPoint
```

- Regression library

```
data = results.select("OrderMonthYear", "SaleAmount")\
  .map(lambda r: LabeledPoint(r[1], [r[0]]))\
  .toDF()
```

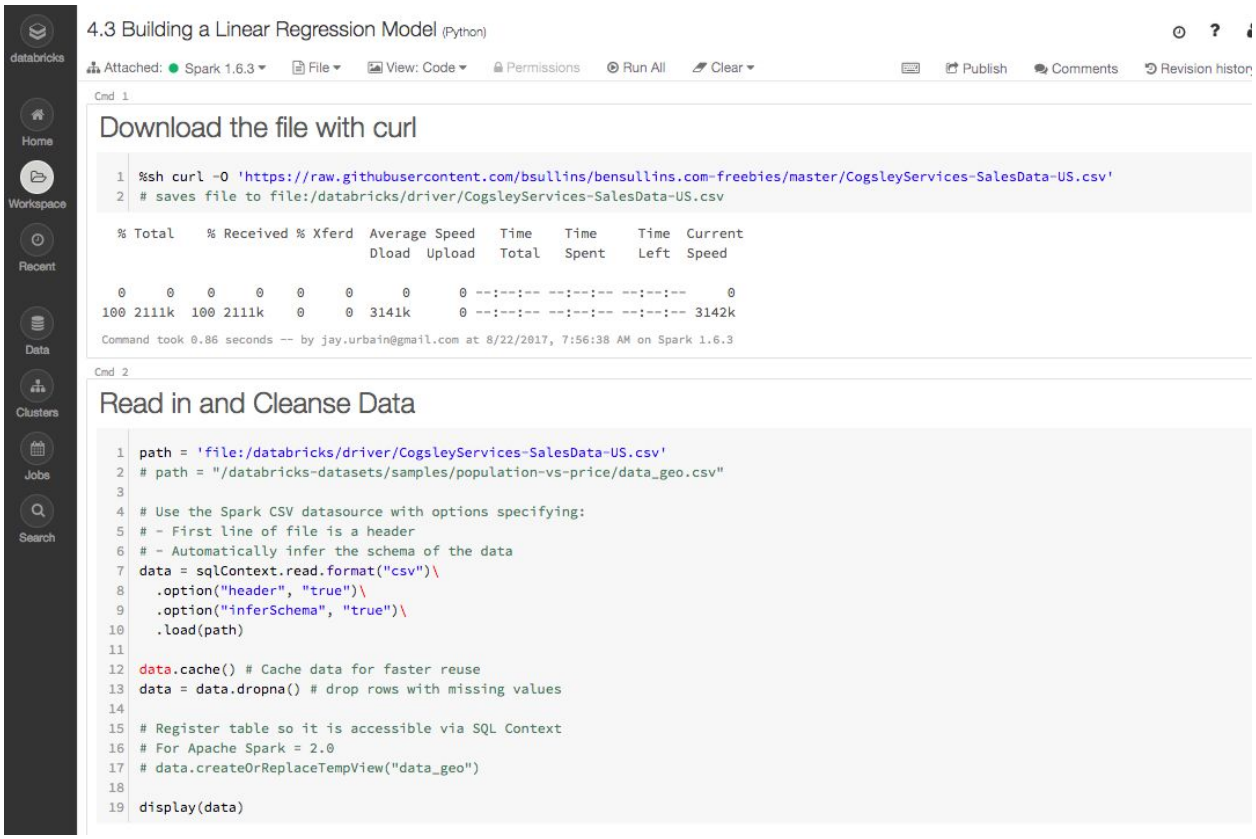
- By applying map to a dataframe, you get a result in the form of an RDD

Building a linear regression model

- Import ML library
- Define the algorithm
- Fit two models
- Make predictions

Load the “Building a Linear Regression Model” notebook.

Make sure you attach to a Spark 1.6 cluster



from pyspark.mllib.regression import LabeledPoint

- Imports the regression library

data = results.select("OrderMonthYear", "SaleAmount").map(lambda r: LabeledPoint(r[1], [r[0]]))\
.toDF()

- Select OrderMonthYear, SaleAmount from results
- Map to RDD taking the second, and first columns as labeled points (x,y)
- Convert RDD back to dataframe

Build linear regression model

https://spark.apache.org/docs/latest/mllib-linear-methods.html

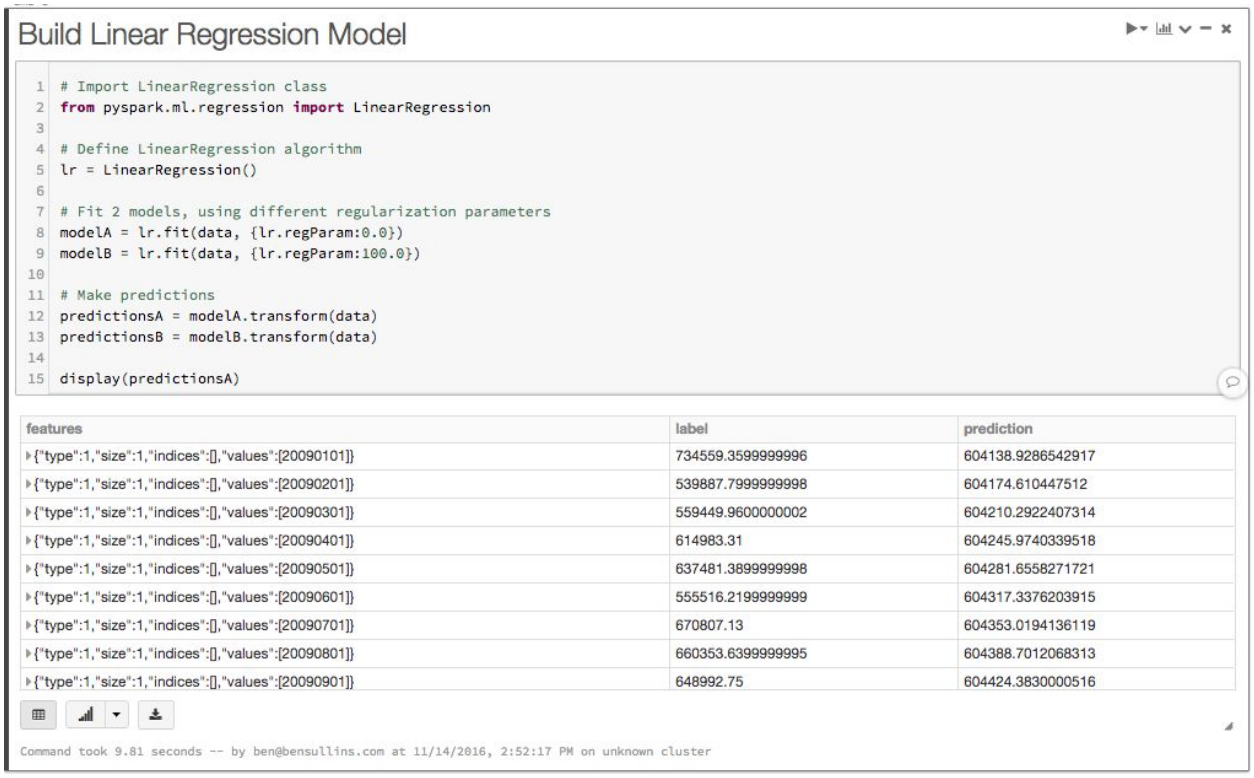
lr = LinearRegression()
Create linear regression model

modelA = lr.fit(data, {lr.regParam:0.0})

- Fit the data, i.e., with no regulariztion

modelB = lr.fit(data, {lr.regParam:100.0})

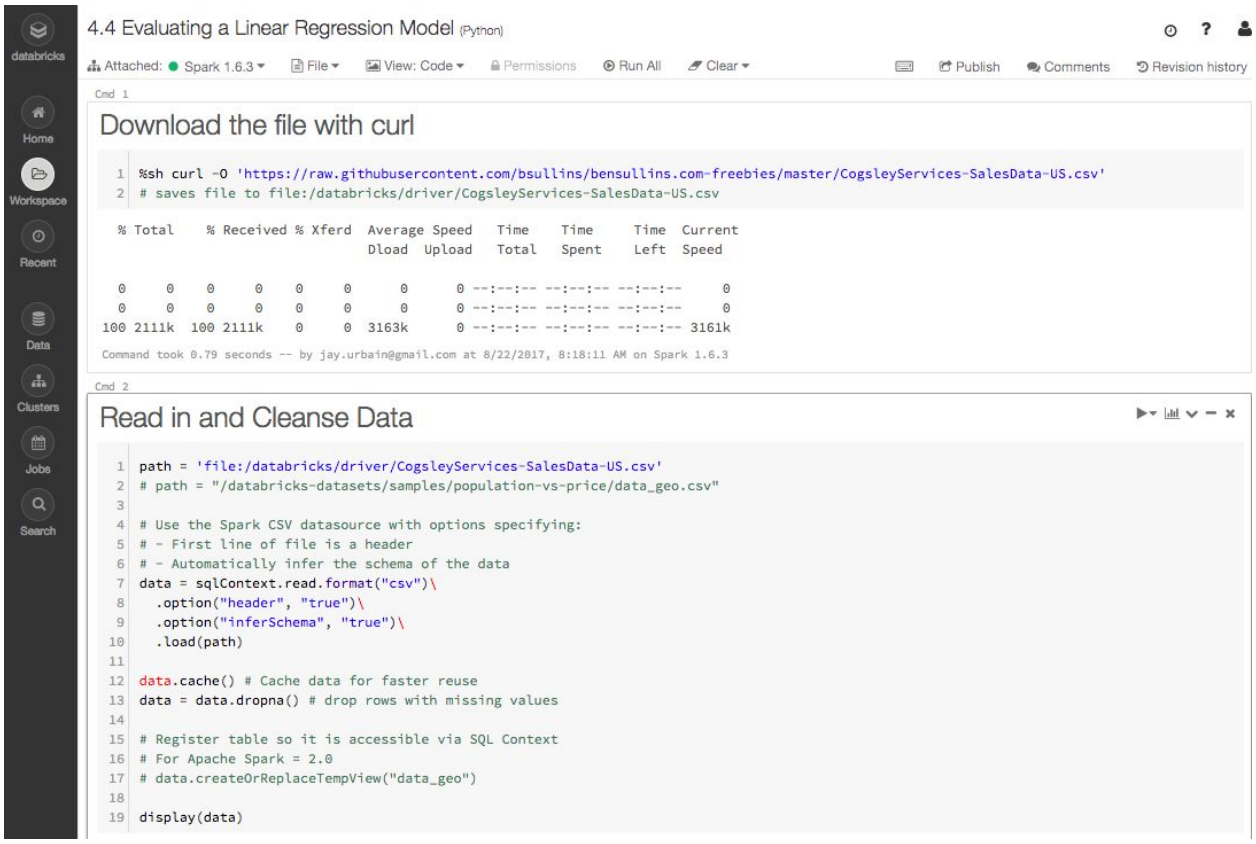
```
# Make predictions on data
predictionsA = modelA.transform(data)
predictionsB = modelB.transform(data)

display(predictionsA)
```



Evaluating a linear regression model

- Import ML evaluation library
- Choose metric: RMSE
- Calculate root-mean-square error

Open the “Evaluating a Linear Regression Model” notebook.



Run each cell.

Check Models for Accuracy cell:

from pyspark.ml.evaluation import RegressionEvaluator

- Import regression evaluator

evaluator = RegressionEvaluator(metricName="rmse")

- Create evaluator instance

RMSE = evaluator.evaluate(predictionsA)

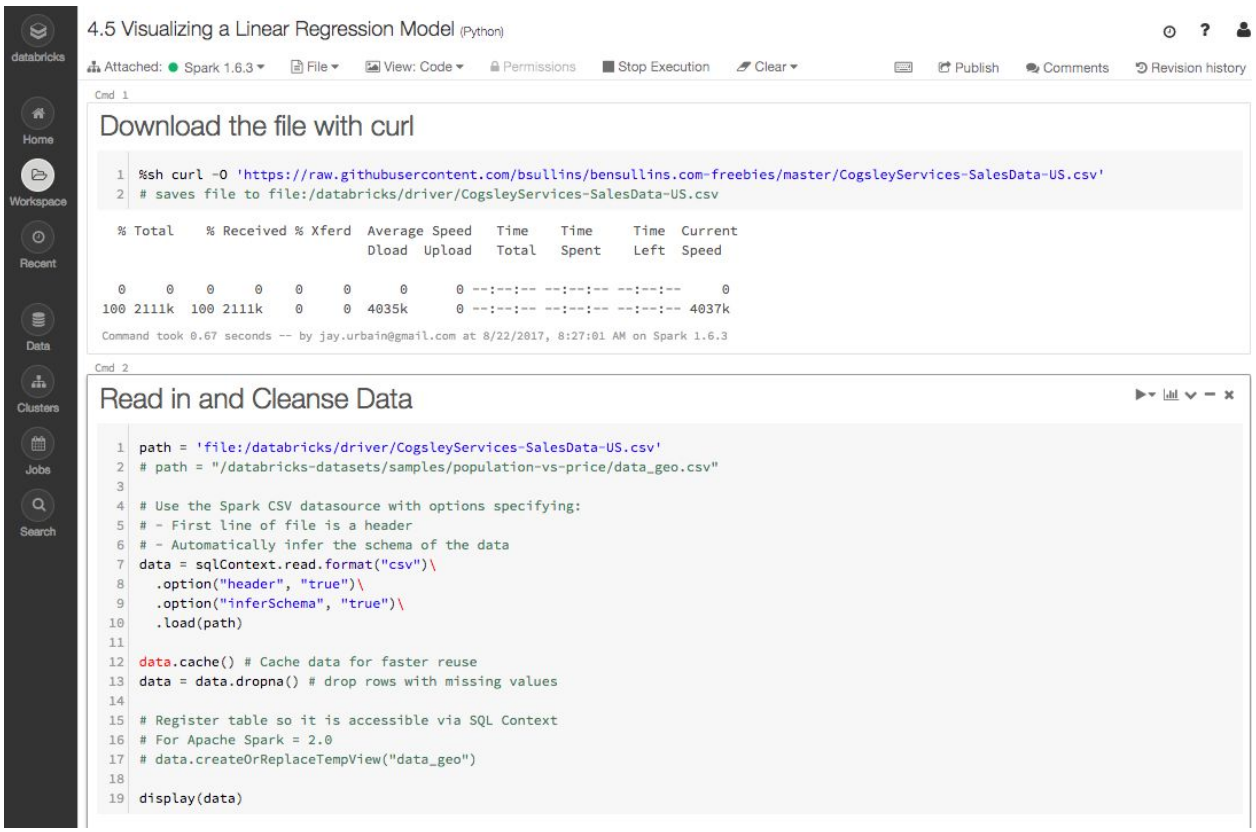
- Calculate root-mean-square error on predictionsA

Both models are about the same with respect to loss

Visualizing a linear regression model

- Create SQL table
- Join tables
- Setup visualization

Load “Visualizing a Linear Regression Model” notebook:



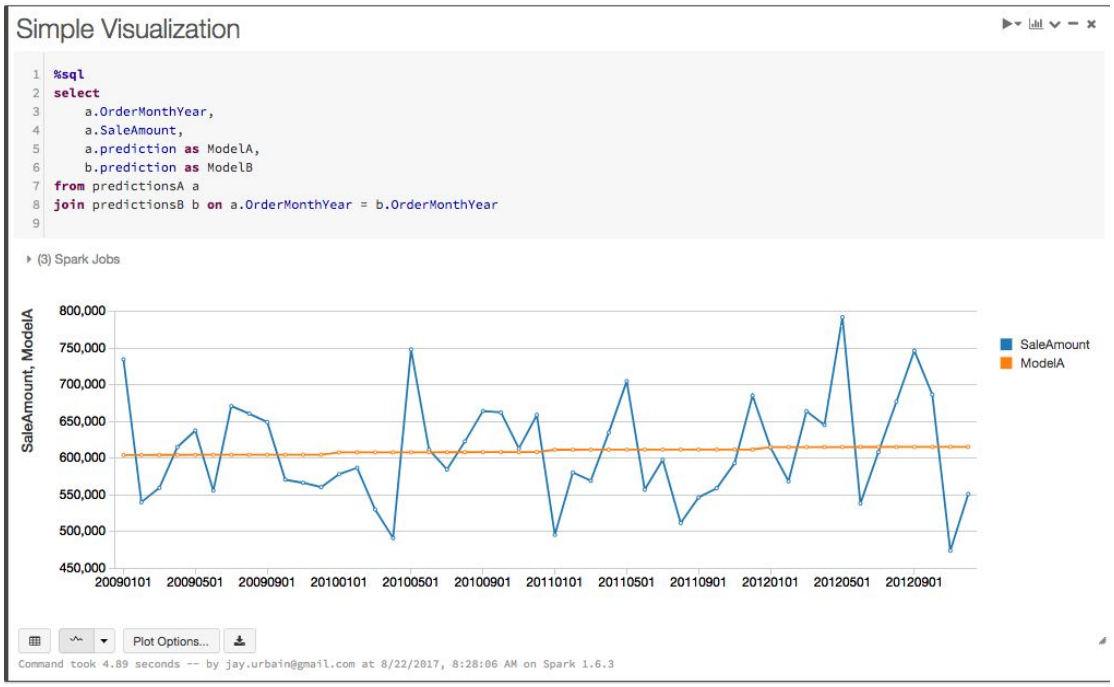
Execute each cell down to Create Tables with Predictions

```
tableA = sc.parallelize(\
        predictionsA.map(lambda r: (float(r.features[0]), r.label, r.prediction)).collect()\
    ).toDF(cols)
```

- The dataframe map function converts dataframe to an RDD, and runs the RDD map function
- Map uses lambda function to select x, y features
- collect() returns all rows in the RDD
- toDF converts RDD back to a dataframe tableA

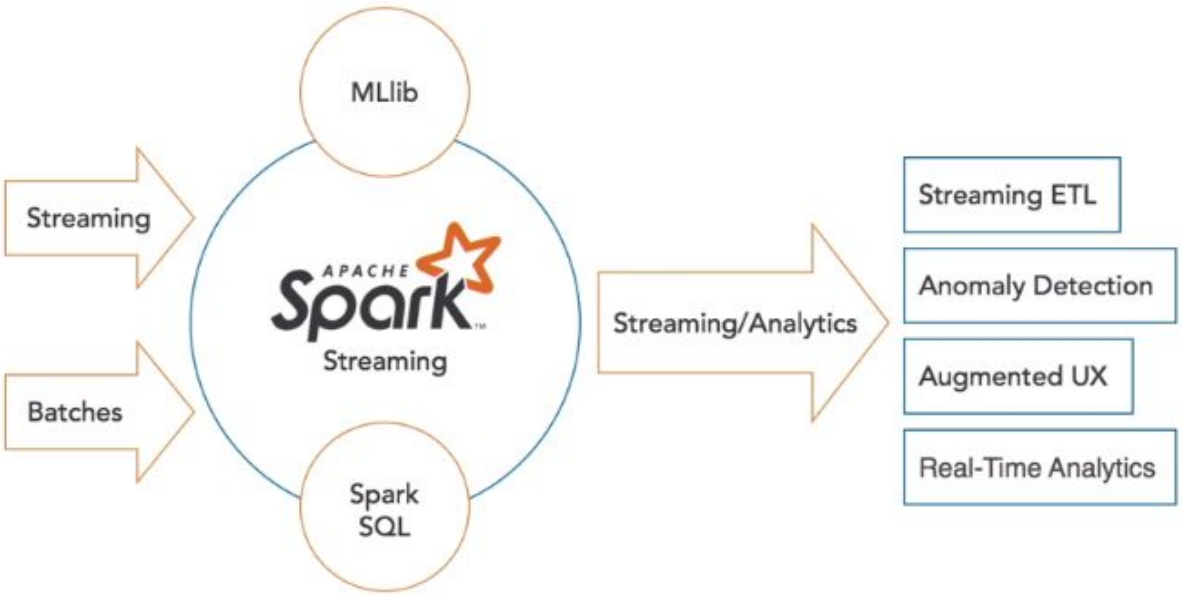
Simple Visualizations

%sql is a “magic function” that converts default python notebook cell to allow sql for this cell



Real-time Data Analysis with Spark Streaming

- Streaming analytics
- Realtime data streamed in and batched in at a regular interval
- Can run Mlib functions or SQL on batched data



Spark chunks operations into very small batches called micro-batches

- Streams of data comes in
- Data is processed by receiver into micro batches
- Spark then processes these jobs and then produces the results based on whatever the jobs were defined as

Implementing streaming

- Setup the streaming context
- Executing jobs using Spark Mlib or Spark SQL

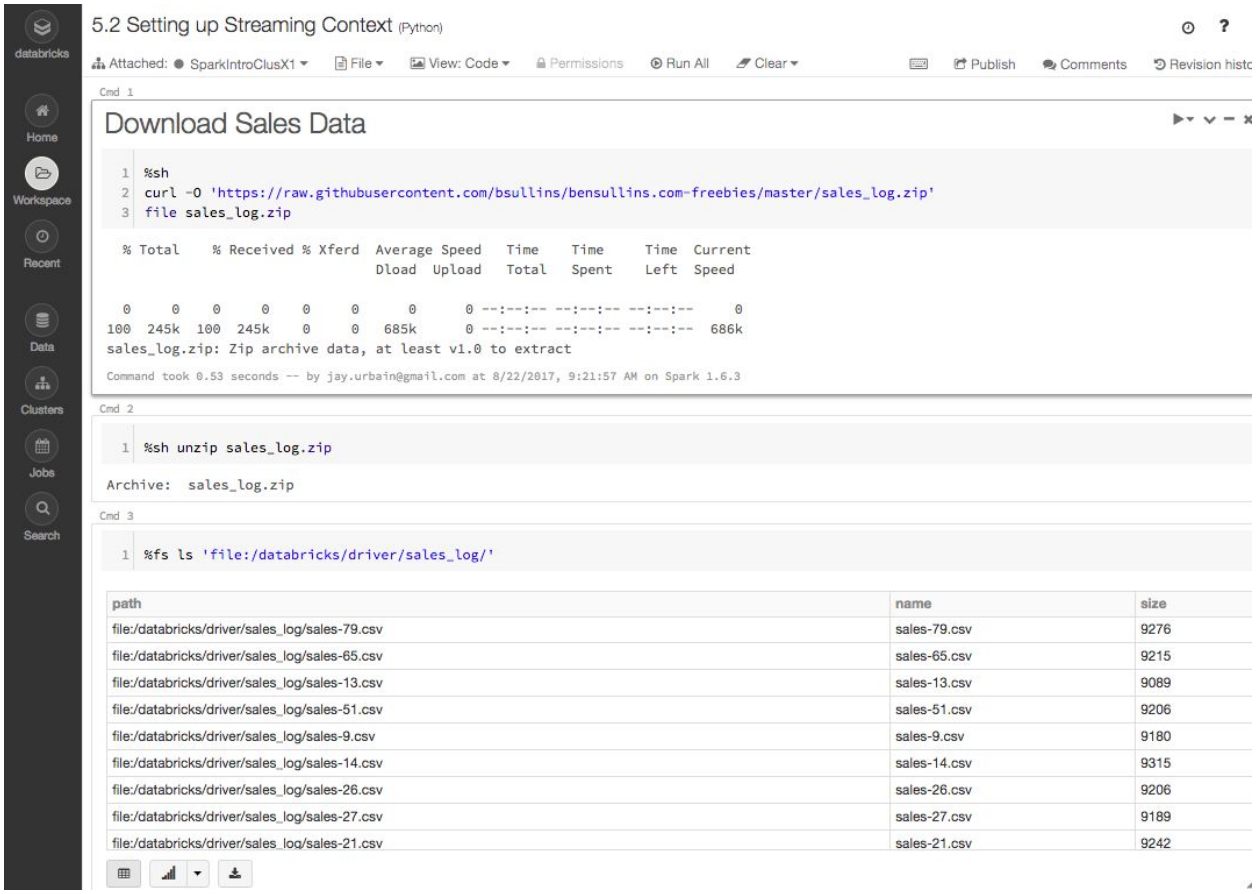
Process

- Download files to “simulate” stream
- Perform analysis with spark SQL or MLib
- Setup streaming job

Streaming context setup

Load the Setting of Streaming Context notebook.

Use a 2.2 cluster for Spark Streaming



%sh is a magic command for executing a shell (terminal) command

Downloads sales data, nzips the data, lists the files

Read data into SQL table, create schema

```
  spark
    .read
    .schema(salesSchema)
    .csv(path)
)
```

Reads csv data, applies schema into dataframe

Read in Data

```
from pyspark.sql.types import *

path = "file:/databricks/driver/sales_log/"

# create schema for data so stream processing is faster
salesSchema = StructType([
  StructField("OrderID", DoubleType(), True),
  StructField("OrderDate", StringType(), True),
  StructField("Quantity", DoubleType(), True),
  StructField("DiscountPct", DoubleType(), True),
  StructField("Rate", DoubleType(), True),
  StructField("SaleAmount", DoubleType(), True),
  StructField("CustomerName", StringType(), True),
  StructField("State", StringType(), True),
  StructField("Region", StringType(), True),
  StructField("ProductKey", StringType(), True),
  StructField("RowCount", DoubleType(), True),
  StructField("ProfitMargin", DoubleType(), True)])

# Static DataFrame containing all the files in sales_log
data = (
  spark
    .read
    .schema(salesSchema)
    .csv(path)
)


# create table so we can use SQL
data.createOrReplaceTempView("sales")

display(data)
```

OrderID	OrderDate	Quantity	DiscountPct	Rate	SaleAmount	CustomerName	State	Region	ProductKey	RowCount	ProfitMargin
44935	11/17/10	37	0.06	200	7011.4	Arthur Nelson	New Mexico	West	Development - Scala	1	0.65
44935	11/17/10	37	0.01	140	5169.04	Arthur Nelson	Florida	South	Development - Database	1	0.64
2563	11/18/10	12	0.04	200	2322.41	Brenda Hildebrand	Mississippi	South	Development - Scala	1	0.74
2563	11/18/10	33	0.01	120	3951.72	Brenda Hildebrand	Mississippi	South	Development - Business Logic	1	0.44
2752	11/18/10	30	0.03	150	4399.87	Todd Cacioppo	Wisconsin	Central	Consulting - Business Model	1	0.53
2752	11/18/10	41	0.02	140	5670.14	Todd Cacioppo	Wisconsin	Central	Development - Database	1	0.56
2752	11/18/10	10	0.03	200	1955.5	Todd Cacioppo	Wisconsin	Central	Development - Big Data	1	0.69
2752	11/18/10	10	0.04	150	1451.5	Todd Cacioppo	Wisconsin	Central	Consulting - Market Research	1	0.7
13607	11/18/10	12	0.07	150	1687.37	Matthew Lucas	New Mexico	West	Consulting - Business Model	1	0.59

Showing the first 1000 rows.

Setup the streaming context

Querying streaming data

Load Querying Streaming Data notebook.

Make sure you use 2.0+ Spark cluster.

Treat data as if it's streaming in

Need data, static Dataframe to feed data.

Will run across each file in the directory.

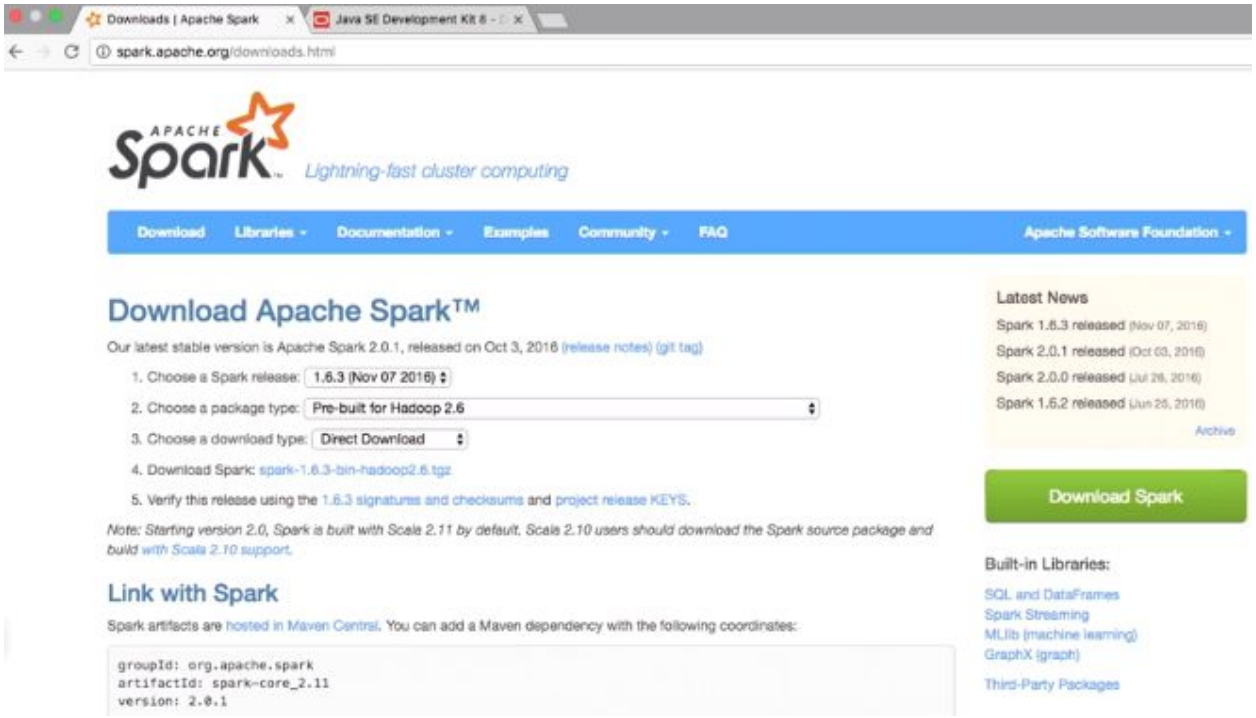
Note: this will take a while

Connecting BI Tools to Spark

Setting up Spark locally

Download Spark from spark.apache.org

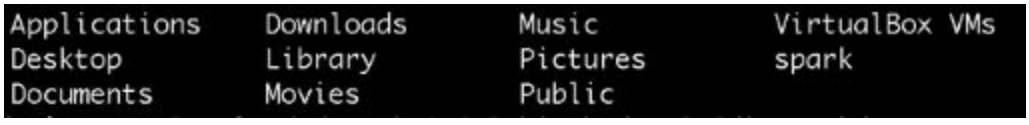
Use prebuilt version



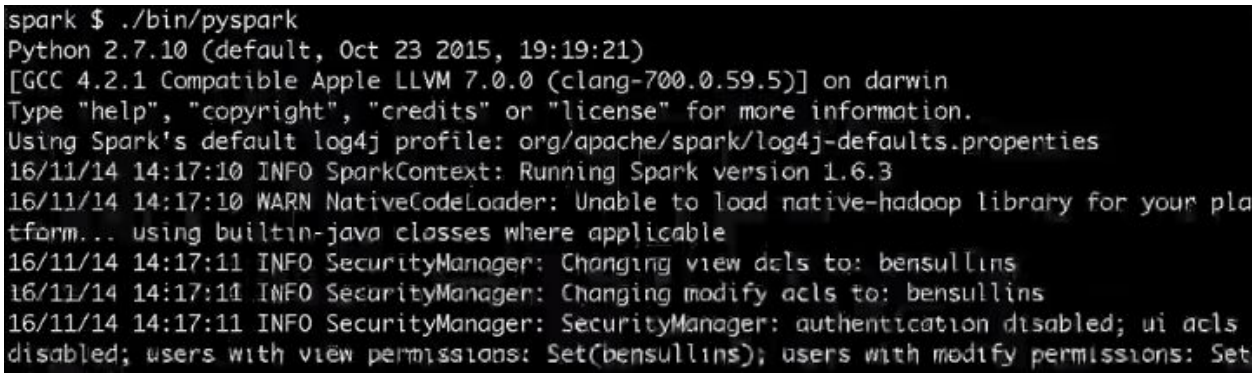
Download Java SDK if not present

Unzip Spark

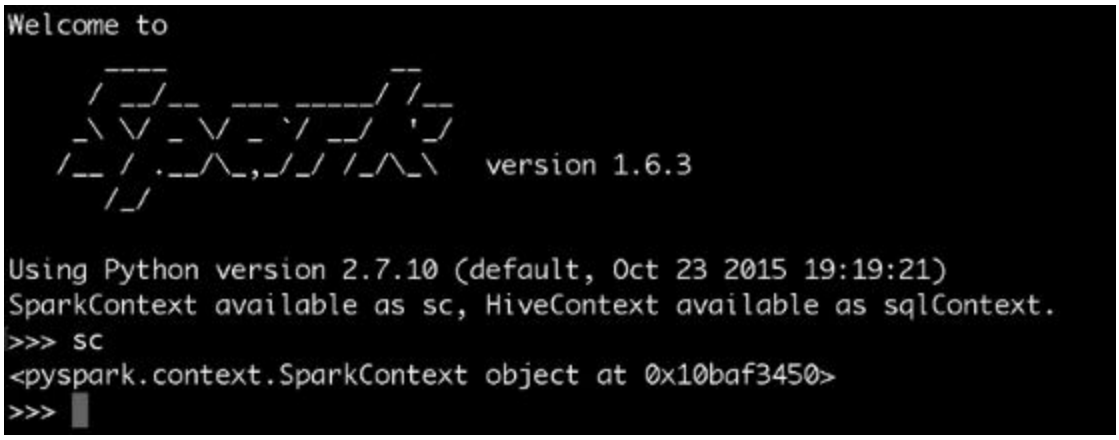
Open terminal window in root of spark directory.



Run pyspark:



Verify:



Connecting jupyter notebook to Spark.

Download Anaconda

https://www.continuum.io/downloads

Run the installer

Open a terminal window

List conda environments:
$conda env list

Select environment:
$source activate “env”

Add conda packages for Spark.
conda install jupyter
conda update jupyter

Set spark profile:

```
~ $ echo "export PATH=$PATH:~/spark/bin" >> .profile
~ $ echo "export PYSPARK_DRIVER_PYTHON=ipython" >> .profile
~ $ echo "export PYSPARK_DRIVER_PYTHON_OPTS='notebook' pyspark" >> .profile
~ $ source .profile
```

Start pyspark with jupyter notebook
$ pyspark

Other ways to connect to Spark

- BI/Tools
- JDBC/ODBC
- Spark SQL