

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 an 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 MLlib.

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:

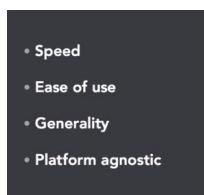
<https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/>

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

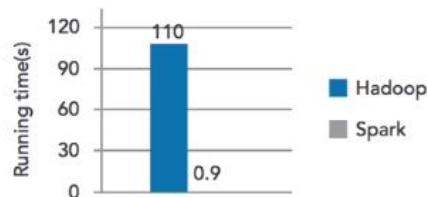
- Spark - fast and general engine for large scale data processing
- Designed for data science, supports data processing and streaming.
- In-memory distributed processing



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):

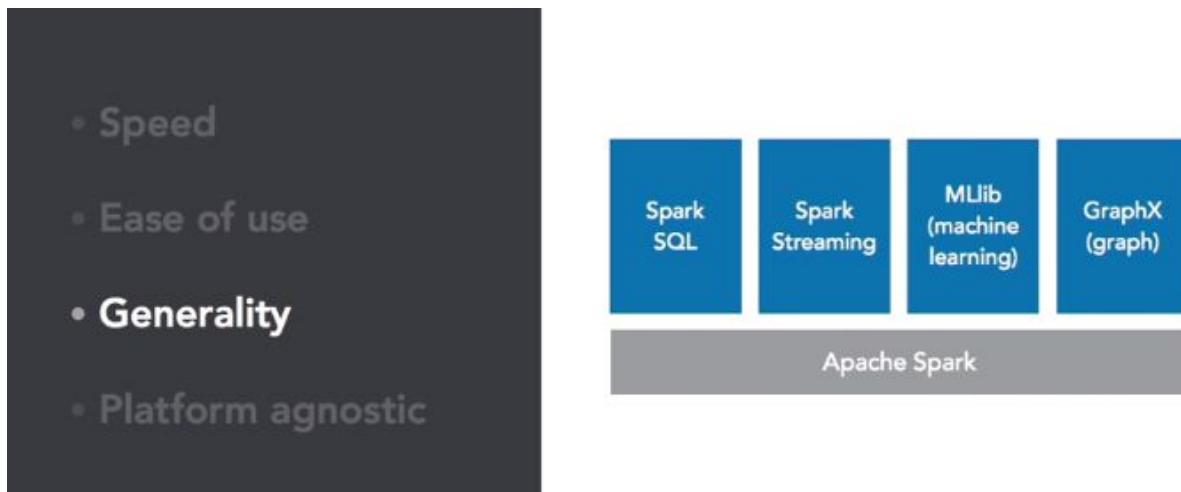
- Speed
- Ease of use
- Generality
- Platform agnostic

```
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.

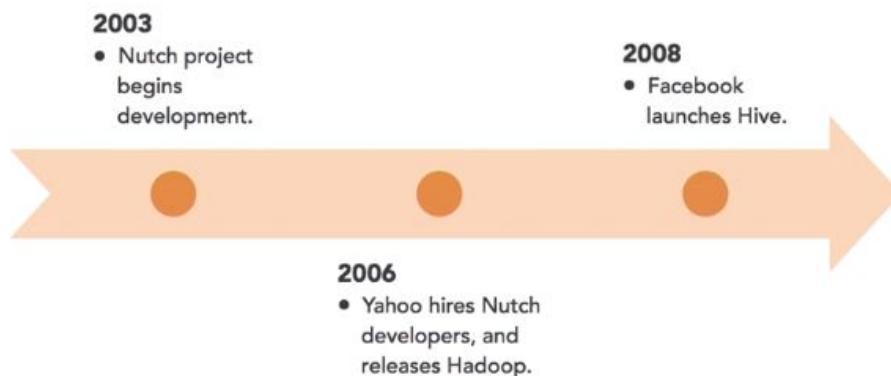
Spark has been designed to work well with other tools.



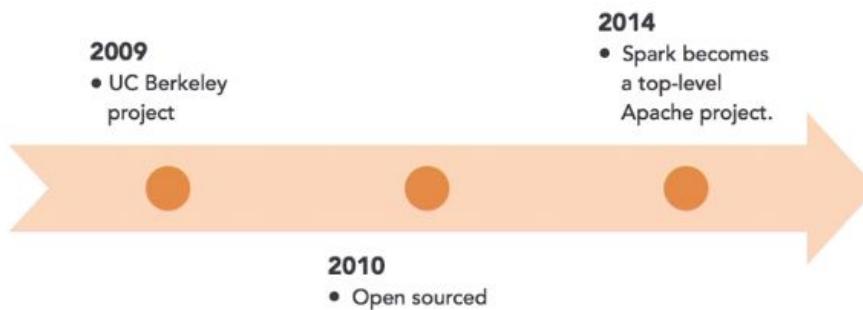
Origins of Spark

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.



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

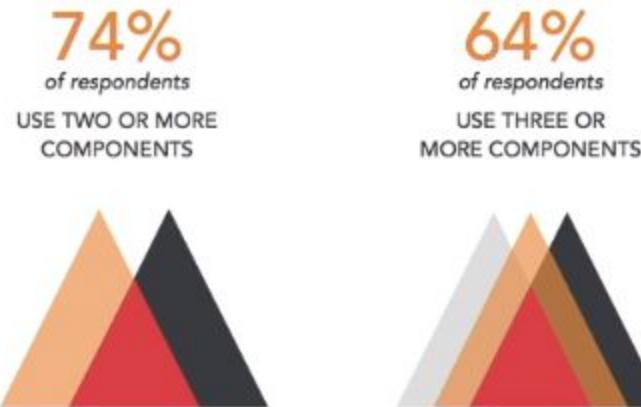
- R package for spark
- Distributed DataFrames
- R Studio integration

Where Spark shines

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

Spark is typically used as a collection of tools

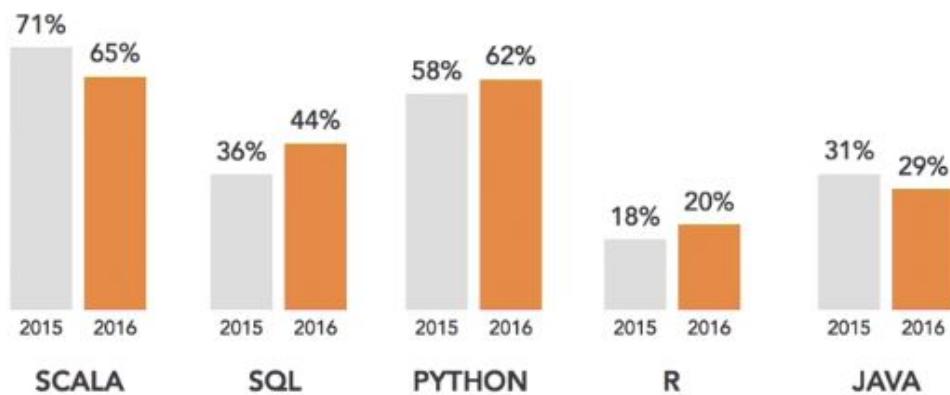
NUMBER OF COMPONENTS USED



What languages are being used?

Q: WHICH LANGUAGES DO YOU USE SPARK IN?

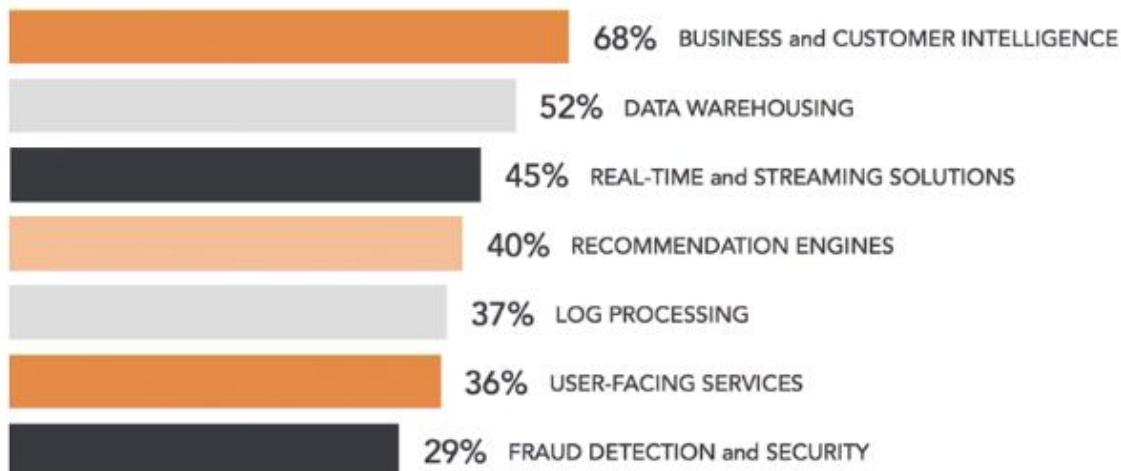
% of respondents who use each language (more than one language could be selected)



Types of applications

TYPES OF PRODUCTS BUILT

% of respondents who use Spark to create each product (more than one product could be selected)



Overview of Databricks

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



A cloud-based managed platform for running
Apache Spark

*Simplifies setup and makes learning easier

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>

A screenshot of the 'Account Owner Login' page. It features a header with the Databricks logo and the title 'Account Owner Login'. Below the header is a sub-header: 'Login to manage or upgrade your Databricks deployment. Note that this is the login information you used to initially setup Databricks account.' There are two input fields: 'Email Address' containing 'jay.urbain@gmail.com' and 'Password' containing '*****'. Below the password field is a 'Forgot Password?' link. A note at the bottom says 'Looking for Community Edition? [Sign in here.](#)' To the right of the input fields is a teal 'Login' button.

You can review their featured tutorials later.

Select Home (left vertical navigation bar)

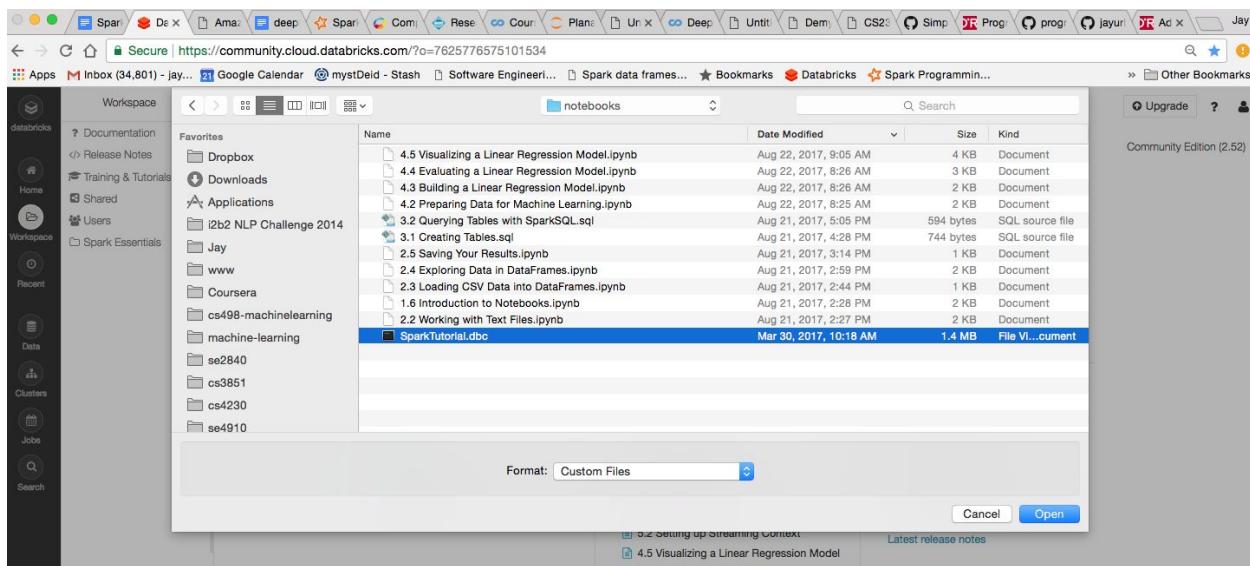
Notebooks and PySpark

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

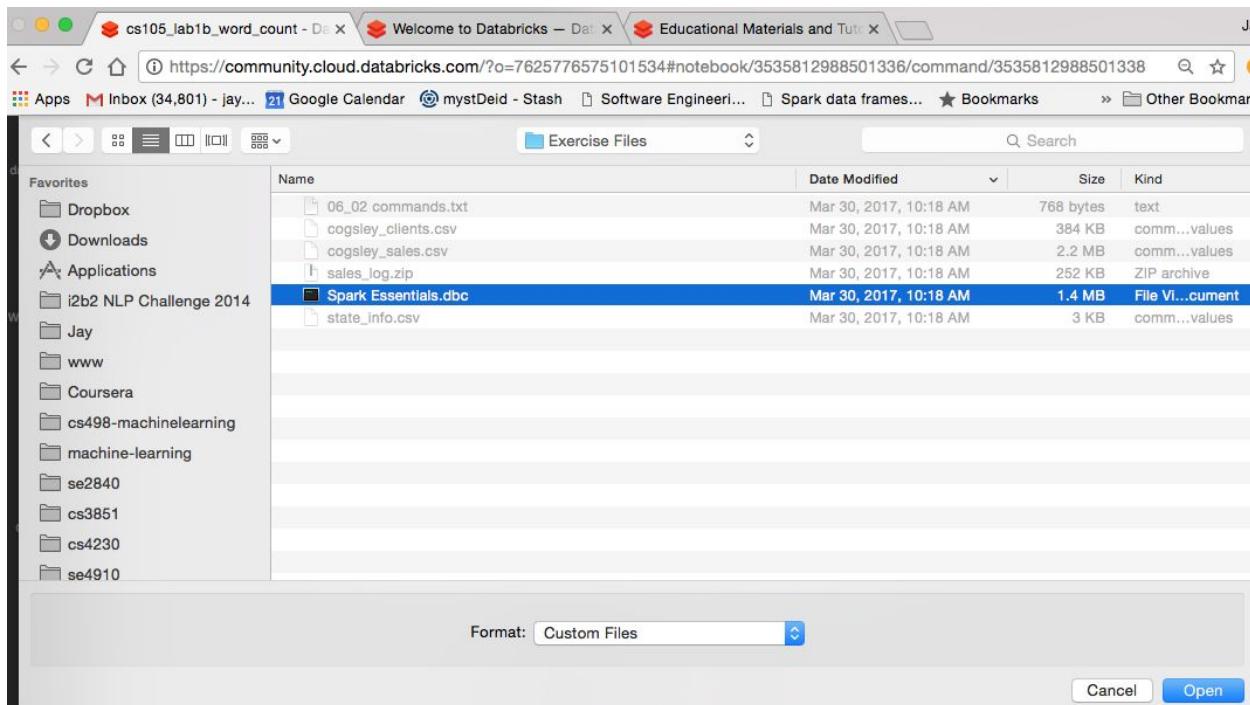


Select Import files

- Right mouse click on Workspace



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



Select Home

The screenshot shows the Databricks web interface. On the left is a sidebar with icons for Workspace, Apps, Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area shows the 'Spark Essentials' workspace under 'Documentation'. A notebook titled '1.6 Introduction to Notebooks (Python)' is open. The first cell, 'Cmd 1', contains the text 'Markdown Content' and examples of lists and links. The second cell, 'Cmd 2', contains Python code to generate a list of integers. The third cell, 'Cmd 3', shows the output 'Out[2]: 10000'. The fourth cell, 'Cmd 4', contains a comment about using Spark.

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.

This screenshot shows a Databricks workspace interface. On the left is a sidebar with icons for Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area shows a tree view under 'Spark Essentials' with various documentation and tutorial sections. A modal dialog box is open in the center, asking if you want to launch a new cluster (6 GB, 3.0) to start running commands. Below the modal, there are three command cells (Cmd 1, Cmd 2, Cmd 3) containing Python code. Cmd 2 shows the execution of the code, indicating it took 0.11 seconds.

This screenshot shows the 'Clusters' page in the Databricks interface. The sidebar on the left includes icons for Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main content area displays the 'Active Clusters' section, which lists a single cluster named 'My Cluster' with 6 GB of memory. The cluster is described as being 'Community Optimized, 3.0 (includes Apache Spark 2.2.0, Scala 2.11)'. Below this is the 'Terminated Clusters' section, which is currently empty.

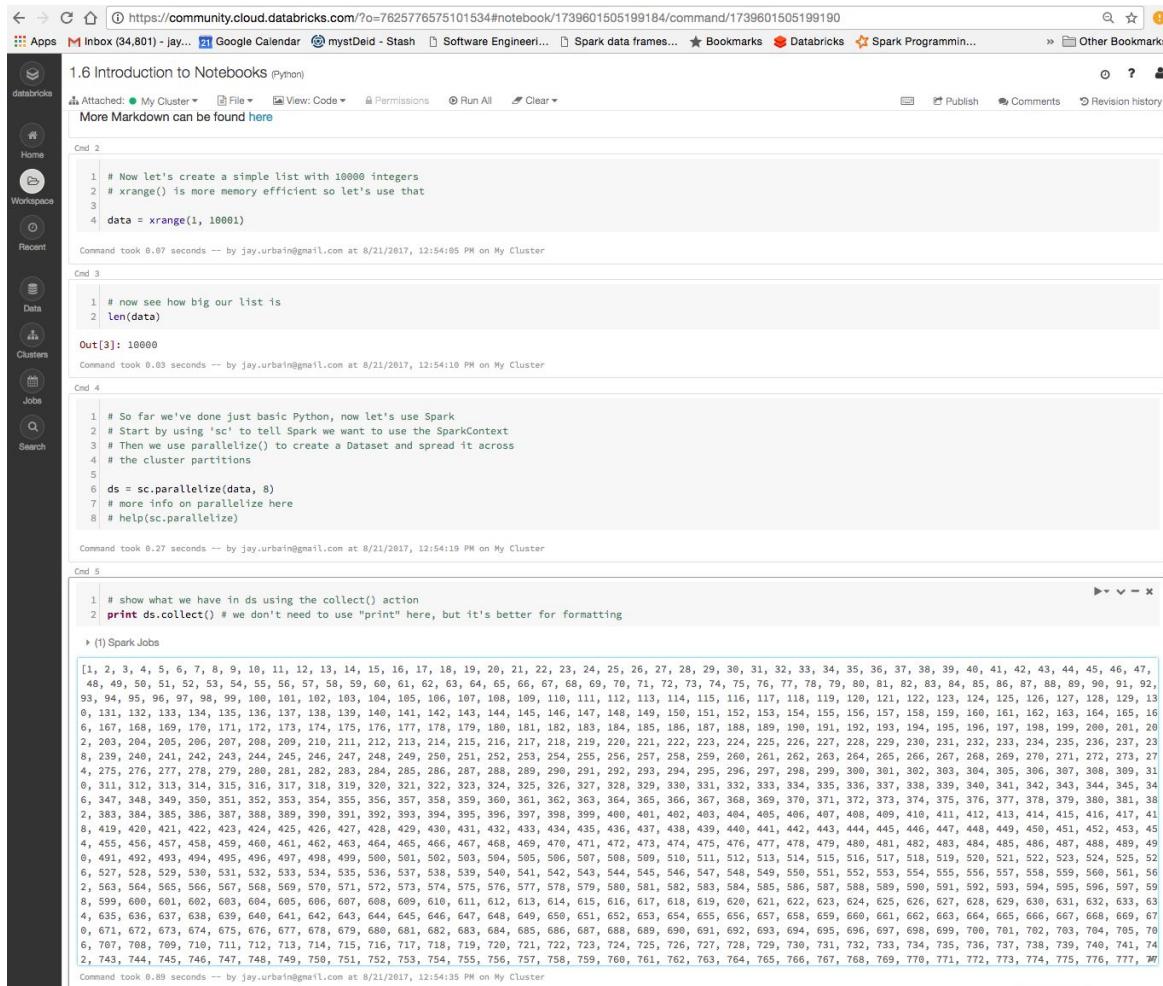
Return to workspace and the Introduction to Notebooks tutorial

- 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.



The screenshot shows a Databricks notebook titled "1.6 Introduction to Notebooks (Python)". The notebook contains five code cells:

- Cell 1:** A simple list comprehension creating a list of 10000 integers from 1 to 10000. It includes a note about using xrange() for memory efficiency.
- Cell 2:** Prints the length of the list, which is 10000.
- Cell 3:** Shows how to use sc.parallelize() to create a Dataset and spread it across partitions.
- Cell 4:** Prints the first 10000 numbers from the parallelized dataset.
- Cell 5:** Prints the first 10000 numbers again, demonstrating the collect() action.

Each cell has a timestamp indicating it was run at 8/21/2017, 12:54:08 PM on My Cluster. The notebook also includes a sidebar with icons for Apps, Home, Workspace, Recents, Data, Clusters, Jobs, and Search, and a top navigation bar with links to Google Calendar, mystDedi - Stash, Software Engineer..., Spark data frames..., Bookmarks, Databricks, Spark Programming..., and Other Bookmarks.

Terminate your cluster:

The screenshot shows the Databricks Clusters page. On the left sidebar, there are icons for Home, Workspace, Recent, Data, and Jobs. The main area is titled 'Clusters' and contains two sections: 'Active Clusters' and 'Terminated Clusters'. In the 'Active Clusters' section, there is a table with one row for 'My Cluster'. The table columns are: Name, Memory, Type, State, Nodes, Spark, Libraries, Notebooks, Default Cluster, and Actions. The 'Actions' column for 'My Cluster' has a button labeled 'Terminate'. In the 'Terminated Clusters' section, there is an empty table.

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

The screenshot shows the Databricks 'Create Cluster' page. On the left sidebar, there are icons for Home, Workspace, Recent, Data, and Jobs. The main area is titled 'Create Cluster' and contains a 'New Cluster' form. The 'Cluster Name' field is filled with 'SparkIntroClusX1'. The 'Databricks Runtime Version' dropdown is set to '3.0 (includes Apache Spark 2.2.0, Scala 2.11)'. Below the form, a note states: 'Free 6GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For more configuration options, please upgrade your Databricks subscription.' At the bottom of the form, there are tabs for 'Instances' and 'Spark', with 'Spark' selected. The 'Availability Zone' dropdown is set to 'us-west-2c'.

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.

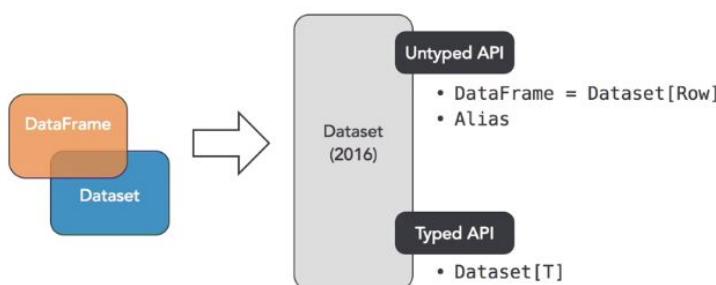
DataFrame

- 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, only executed when an action is run

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

Working with Text Files



Unstructured
(schema-never)



Semistructured
(schema-later)



Structured
(schema-first)

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 ls
```

path	name	size
dbfs:/FileStore/	FileStore/	0
dbfs:/databricks-datasets/	databricks-datasets/	0
dbfs:/databricks-results/	databricks-results/	0
dbfs:/tmp/	tmp/	0
dbfs:/user/	user/	0

```
%fs ls /databricks-datasets/bikeSharing/README.md
```

path	name	size
dbfs:/databricks-datasets/bikeSharing/README.md	README.md	5016

%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

2.3 Loading CSV Data into DataFrames (Python)

Attached: SparkIntroClusX1

Cmd 1

Find a Directory with CSVs

```
%fs ls /databricks-datasets/online_retail/data-001/
```

path	name	size
dbfs/databricks-datasets/online_retail/data-001/data.csv	data.csv	5357240

Command took 0.41 seconds -- by jay.urbain@gmail.com at 8/21/2017, 2:34:23 PM on SparkIntroClusX1

Cmd 2

```
# specify path
path = "/databricks-datasets/online_retail/data-001/data.csv"
# load as text
data = spark.read.csv(path)
# show sample
data.take(20)
```

(2) Spark Jobs

Out[1]:

```
[Row(_c0=u'InvoiceNo', _c1=u'StockCode', _c2=u'Description', _c3=u'Quantity', _c4=u'InvoiceDate', _c5=u'UnitPrice', _c6=u'CustomerID', _c7=u'Country'), Row(_c0=u'536365', _c1=u'85123A', _c2=u'WHITE HANGING HEART T-LIGHT HOLDER', _c3=u'6', _c4=u'12/1/10 8:26', _c5=u'2.55', _c6=u'17850', _c7=u'United Kingdom'), Row(_c0=u'536365', _c1=u'71053', _c2=u'WHITE METAL LANTERN', _c3=u'6', _c4=u'12/1/10 8:26', _c5=u'3.39', _c6=u'17850', _c7=u'United Kingdom'), Row(_c0=u'536365', _c1=u'844068', _c2=u'CREAM CUPID HEARTS COAT HANGER', _c3=u'8', _c4=u'12/1/10 8:26', _c5=u'2.75', _c6=u'17850', _c7=u'United Kingdom'), Row(_c0=u'536365', _c1=u'840296', _c2=u'KNITTED UNION FLAG HOT WATER BOTTLE', _c3=u'6', _c4=u'12/1/10 8:26', _c5=u'3.39', _c6=u'17850', _c7=u'United Kingdom')]
```

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

The screenshot shows a Databricks notebook interface. The title bar says "2.4 Exploring Data in DataFrames (Python)". The left sidebar has icons for Home, Workspace, Data, Clusters, Jobs, and Search. The main area has a command history and a code editor.

Command History:

```
1 %fs ls /databricks-datasets/online_retail/data-001/
```

Code Editor (Cmd 2):

```
1 # specify path
2 path = "/databricks-datasets/online_retail/data-001/data.csv"
3
4 # read in file using csv format
5 df = spark.read.load(path,
6   format='com.databricks.spark.csv',
7   header='true',
8   inferSchema='true')
9
10 # show 20 rows
11 display(df)
```

Data Preview:

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26	2.55	17850	United Kingdom
536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39	17850	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26	2.75	17850	United Kingdom
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26	3.39	17850	United Kingdom
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26	3.39	17850	United Kingdom
536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/10 8:26	7.65	17850	United Kingdom
536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/10 8:26	4.25	17850	United Kingdom
536366	22633	HAND WARMER UNION JACK	6	12/1/10 8:28	1.85	17850	United Kingdom
536366	22632	HAND WARMER RED POLKA DOT	6	12/1/10 8:28	1.85	17850	United Kingdom

Showing the first 1000 rows.

```
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:

The screenshot shows the Databricks SQL workspace interface. On the left is a sidebar with icons for Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area has two tabs open:

- 3.1 Creating Tables (SQL)**: This tab shows a command-line interface (CLI) window titled "Find a CSV". It contains the command "%fs ls /databricks-datasets/samples/population-vs-price/" and a table showing the result: a single file named "data_geo.csv" located at "dbfs:/databricks-datasets/samples/population-vs-price/data_geo.csv" with a size of 10952. Below the table are download and refresh buttons.
- Create Table w/o Schema**: This tab shows a code editor with the following SQL script:

```
1 CREATE TABLE IF NOT EXISTS population_v_price
2 USING CSV
3 OPTIONS (path "/databricks-datasets/samples/population-vs-price/data_geo.csv", header "true", inferSchema "true");
4
5 /* check results */
6 select * from population_v_price limit 100;
```

Below the code is a table titled "2014 rank" containing data for cities in Alabama and Arizona. The table has columns: 2014 rank, City, State, State Code, 2014 Population estimate, and 2015 median sales price. The data includes rows for Birmingham, Huntsville, Mobile, Montgomery, Anchorage, Chandler, Gilbert, Glendale, and Mesa.

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:

2014 rank	city	state	State Code
101	Birmingham	Alabama	AL
125	Huntsville	Alabama	AL
122	Mobile	Alabama	AL
114	Montgomery	Alabama	AL
64	Anchorage[19]	Alaska	AK
78	Chandler	Arizona	AZ
86	Gilbert[20]	Arizona	AZ
88	Glendale	Arizona	AZ
38	Mesa	Arizona	AZ

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")

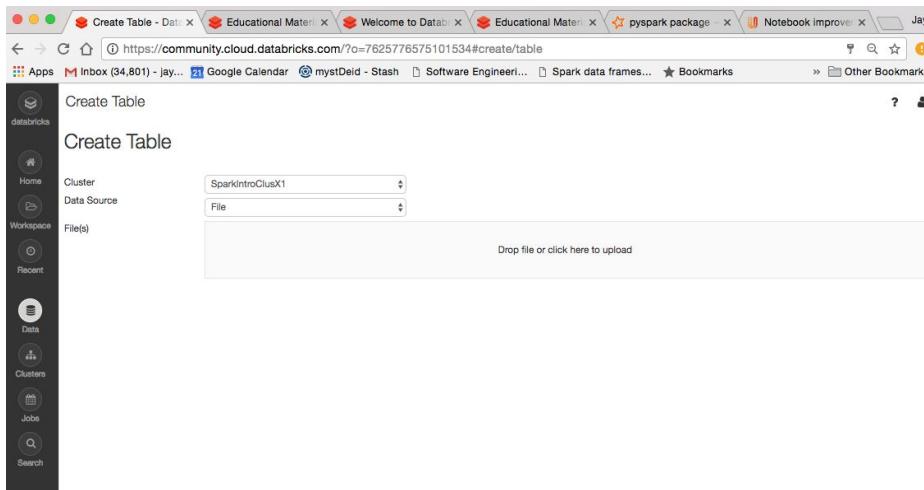
sqlContext.sql("SELECT * FROM schema_patient_mother limit 100").show(5)
```

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

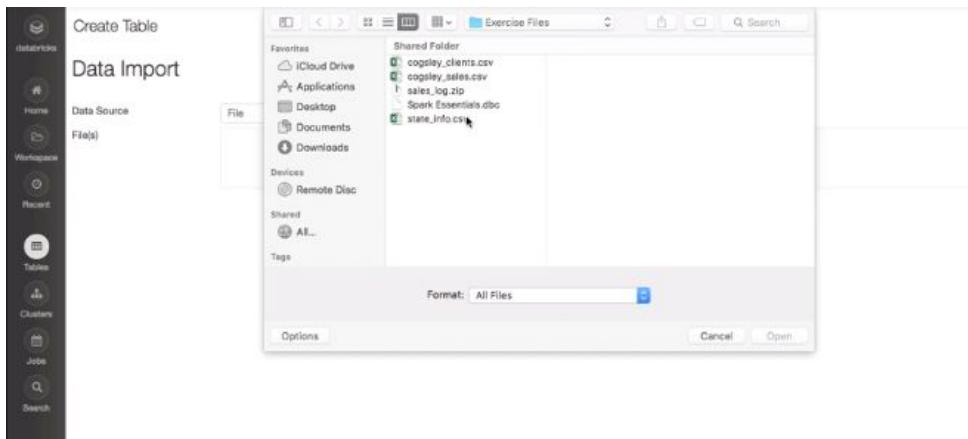
```
# Register the DataFrame as a table.  
  
sc.sql("DROP TABLE IF EXISTS schema_patient_mother")  
sc.registerDataFrameAsTable(schema_patient_mother, "schema_patient_mother")  
  
sc.sql("SELECT * FROM schema_patient_mother limit 100").show(5)
```

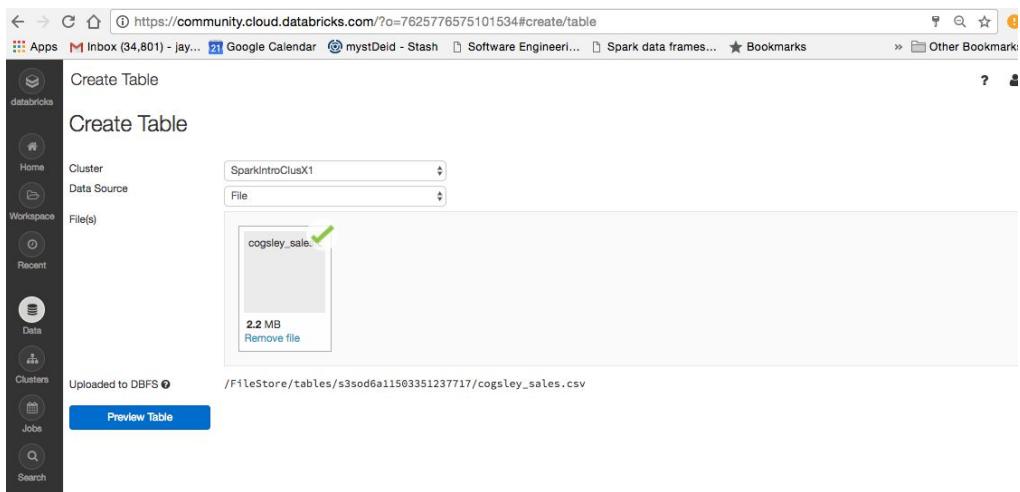
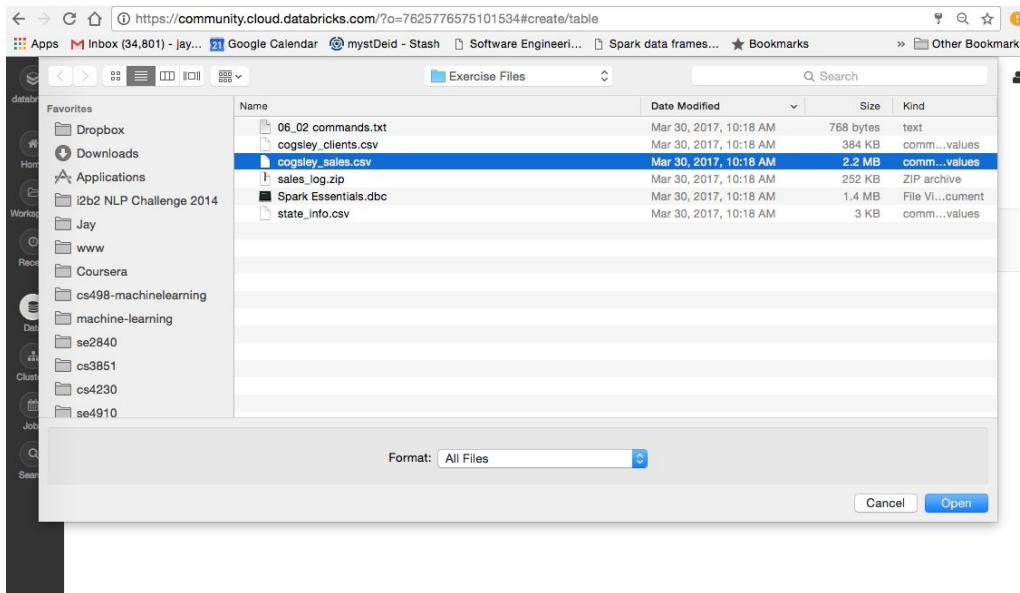
Manually creating a table from a csv file:

Select Data -> Tables -> '+'



Note: data is in tutorial data sub-directory.





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

The screenshot shows the Databricks Create Table interface. A file named 'cogsley_clients.csv' has been uploaded to DBFS. The table details are as follows:

Name	Symbol	LastSale	MarketCapLabel	MarketCapAmount	IPOYear	Sector	Industry	Summary Quot
1347 Capital Corp.	TFSC	8.43	\$56.09M	56090000	2014	Finance	Business Services	http://www.nasdaq
1347 Capital Corp.	TFSCR	0.37	n/a	0	2014	Finance	Business Services	http://www.nasdaq
1347 Capital Corp.	TFSCU	9.97	\$41.67M	41670000	2014	n/a	n/a	http://www.nasdaq

The 'File type' is set to CSV and 'Column Delimiter' is a comma (,). The 'First row is header' checkbox is checked. A 'Create Table' button is visible at the bottom left.

Select Create Table

The screenshot shows the Databricks Table view for 'cogsley_sales'. The schema is defined as follows:

col_name	data_type	comment
RowID	string	null
OrderID	string	null
OrderDate	string	null
OrderMonthYear	string	null
Quantity	string	null
Quote	string	null
DiscountPct	string	null
Rate	string	null
SaleAmount	string	null
CustomerName	string	null
CompanyName	string	null
Sector	string	null
Industry	string	null
City	string	null
ZipCode	string	null
State	string	null
Region	string	null

The sample data is as follows:

RowID	OrderID	OrderDate	OrderMonthYear	Quantity	Quote	DiscountPct	Rate	SaleAmount	CustomerName	CompanyName	Sector	Industry	City	ZipCode	State
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	60035	Illinois
2	6	2012-02-20	2012-02-01	2	280	0.01	140	277.20	Ruben Staebel	Celgene Corporation	Health Care	Major Pharmaceuticals	Edmonds	98026	Washington
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	98387	Washington

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:

Create Table

Cluster: SparkIntroClusX1

File(s): cogsley_clients.csv (0.4 MB)

Uploaded to DBFS /FileStore/tables/047gsx901503351568831/cogsley_clients.csv

Preview Table

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

Create Table

Cluster: SparkIntroClusX1

File(s): cogsley_clients (0.4 MB)

Table Details

Table name	Name	Symbol	LastSale	MarketCapLabel	MarketCapAmount	IPOyear	Sector	industry	Summary Quote
cogsley_clients	STRING	STRING	FLOAT	STRING	BIGINT	STRING	STRING	STRING	STRING
File type	CSV								
Column Delimiter	,								
<input checked="" type="checkbox"/> First row is header									
<button>Create Table</button>	1347 Capital Corp.	TFSC	9.43	\$56.09M	56090000	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfsc
	1347 Capital Corp.	TFSCR	0.37	n/a	0	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfscr
	1347 Capital Corp.	TFSCU	9.97	\$41.67M	41670000	2014	n/a	n/a	http://www.nasdaq.com/symbol/tfscu

Create table.

Table: cogsley_clients

cogsley_clients | Refresh

Cluster: SparkIntroClusX1

Schema:

col_name	data_type	comment
Name	string	null
Symbol	string	null
LastSale	string	null
MarketCapLabel	string	null
MarketCapAmount	string	null
IPOyear	string	null
Sector	string	null
industry	string	null
Summary Quote	string	null

Sample Data:

Name	Symbol	LastSale	MarketCapLabel	MarketCapAmount	IPOyear	Sector	industry	Summary Quote
1347 Capital Corp.	TFSC	9.43	\$56.09M	56090000	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfsc
1347 Capital Corp.	TFSCR	0.37	n/a	0	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfscr
1347 Capital Corp.	TFSCU	9.97	\$41.67M	41670000	2014	n/a	n/a	http://www.nasdaq.com/symbol/tfscu
1347 Capital Corp.	TFSCW	0.2	n/a	0	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfscw
1347 Property Insurance Holdings, Inc.	PIH	7.66	\$48.7M	48700000	2014	Finance	Property-Casualty Insurers	http://www.nasdaq.com/symbol/pih
1-800 FLOWERS.COM, Inc.	FLWS	10.32	\$667.78M	667780000	1999	Consumer Services	Other Specialty Stores	http://www.nasdaq.com/symbol/flws
1st Century Bancshares, Inc	FCTY	6.774	\$68.73M	68730000	n/a	Finance	Major Banks	http://www.nasdaq.com/symbol/fcty
1st Constitution Bancorp (NJ)	FCCY	11.18	\$79.77M	79770000	n/a	Finance	Savings Institutions	http://www.nasdaq.com/symbol/fccy

Create state_info table:

The screenshot shows the Databricks interface for creating a new table. The left sidebar includes icons for Home, Cluster, Data Source, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area is titled 'Create Table' with 'Cluster' set to 'SparkIntroClusX1' and 'File(s)' selected. A file named 'state_info.csv' is uploaded, showing a preview of 2.8 KB. The table details section shows the table name 'state_info' and columns: State (STRING), StateCode (STRING), Capital (STRING), LargestCity (STRING), PopulationEst2015 (BIGINT), HouseSeats (STRING), and Statehood (STRING). The preview table shows data for Alabama, Alaska, and Arizona.

State	StateCode	Capital	LargestCity	PopulationEst2015	HouseSeats	Statehood
Alabama	AL	Montgomery	Birmingham	4858979	7	1819-12-14
Alaska	AK	Juneau	Anchorage	738432	1	1959-01-03
Arizona	AZ	Phoenix	Phoenix	6826065	9	1912-02-14

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.

The screenshot shows the Databricks interface for creating a new table. The left sidebar includes icons for Home, Cluster, Data Source, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area is titled 'Create Table' with 'Cluster' set to 'SparkIntroClusX1' and 'File(s)' selected. A file named 'cogsley_sales.csv' is uploaded, showing a preview of 2.2 MB. The table details section shows the table name 'cogsley_sales' and columns: RowID (INT), OrderID (STRING), OrderDate (STRING), OrderMonthYear (STRING), Quantity (INT), Quote (INT), DiscountPct (DOUBLE), Rate (INT), and SaleAmount (DOUBLE). The preview table shows data for three rows (17, 18, 19).

RowID	OrderID	OrderDate	OrderMonthYear	Quantity	Quote	DiscountPct	Rate	SaleAmount
17	97	2010-01-28	2010-01	26	5200	0.03	200	5044.00
18	129	2012-11-18	2012-11	4	440	0.09	110	400.40
19	130	2012-05-07	2012-05	3	450	0.05	150	427.50

Table: cogsley_sales

The screenshot shows the Databricks interface with the following details:

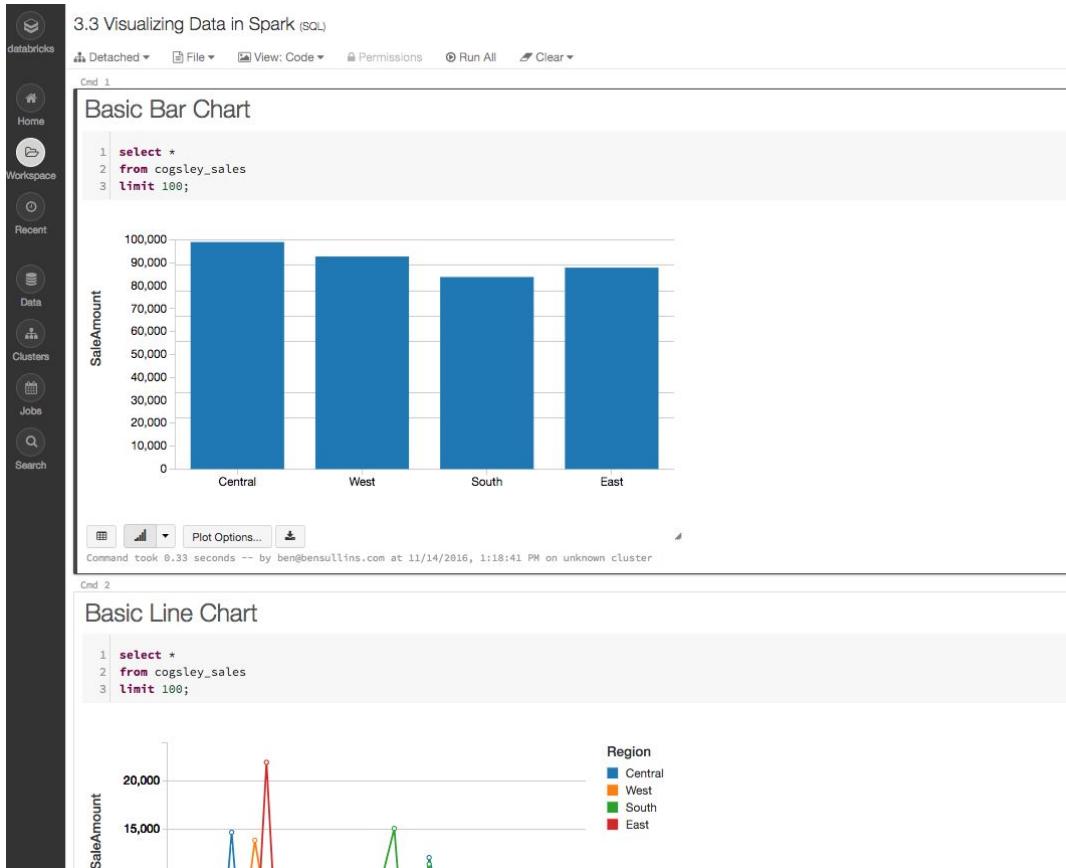
- Left Sidebar:** Contains icons for Home, Workspace, Recent, Data, Clusters, Jobs, and Search.
- Header:** Shows the table name "cogsley_sales" and a "Refresh" button.
- Cluster Selection:** Set to "SparkIntroClusX1".
- Schema:** A table showing the column names, data types, and comments for the "cogsley_sales" table.

col_name	data_type	comment
RowID	int	null
OrderID	string	null
OrderDate	string	null
OrderMonthYear	string	null
Quantity	int	null
Quote	int	null
DiscountPct	double	null
Rate	int	null
SaleAmount	double	null
CustomerName	string	null
CompanyName	string	null
Sector	string	null
Industry	string	null
City	string	null
ZipCode	string	null
State	string	null
Region	string	null

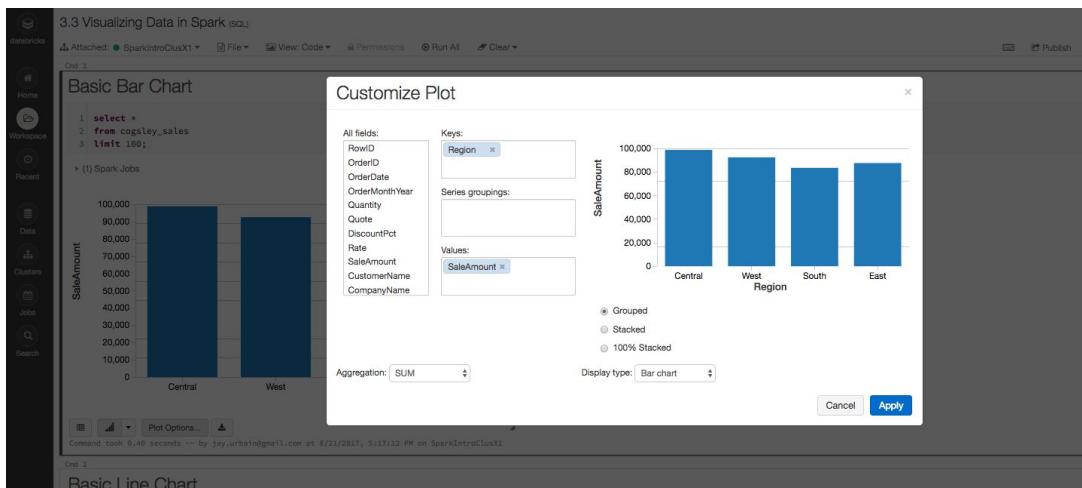
- Sample Data:** A table showing three rows of sample data from the "cogsley_sales" table.

RowID	OrderID	OrderDate	OrderMonthYear	Quantity	Quote	DiscountPct	Rate	SaleAmount	CustomerName	CompanyName
1	3	2010-10-13	2010-10-01	6	1200	0.04	200	1152	Muhammed MacIntyre	CA Inc.
2	6	2012-02-20	2012-02-01	2	280	0.01	140	277.2	Ruben Staebel	Celgene Corporation
3	32	2011-07-15	2011-07-01	26	3250	0.07	125	3022.5	Liz Greer	Twenty-First Century Fox,

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+

The screenshot shows the Databricks interface for creating a new cluster. On the left is a sidebar with icons for Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area is titled "Create Cluster" and "New Cluster". It includes fields for "Cluster Name" (with a placeholder "Please enter a cluster name") and "Databricks Runtime Version" (with a dropdown menu). The dropdown menu lists various versions, with "Spark 1.6.3-db2 (Hadoop 2, Scala 2.10)" highlighted with a blue background. Other listed versions include 3.0, 3.1, 3.2 alpha, Spark 1.3.0, 1.4.1, 1.5.2, and 1.6.3-db2 (Hadoop 1, Scala 2.10). To the right of the dropdown, there is a note about automatic termination after two hours.

Attach to your 1.6.3 Cluster

```

4.2 Preparing Data for Machine Learning (Python)

Attached: ● Spark 1.6.3 ▾ File ▾ View: Code ▾ Permissions ▾ Run All ▾ Clear ▾
100 2111k 100 2111k 0 0 3701k 0 3698k
Command took 0.77 seconds -- by jay.urbain@gmail.com at 8/22/2017, 5:36:37 AM on Spark 1.6.3

Cmd 2
Read in and Cleanse Data

1 path = 'file:/databricks/driver/CogsleyServices-SalesData-US.csv'
2 # path = "/databricks-datasets/samples/population-vs-price/data_geo.csv"
3
4 # Use the Spark CSV datasource with options specifying:
5 # - First line of file is a header
6 # - Automatically infer the schema of the data
7 data = sqlContext.read.format("csv")\
8   .option("header", "true")\
9   .option("inferSchema", "true")\
10  .load(path)
11
12 data.cache() # Cache data for faster reuse
13 data = data.dropna() # drop rows with missing values
14
15 # Register table so it is accessible via SQL Context
16 # For Apache Spark = 2.0
17 # data.createOrReplaceTempView("data_geo")
18
19 display(data)

▶ (3) Spark Jobs

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|RowID|OrderID|OrderDate|OrderMonthYear|Quantity|Quote|DiscountPct|Rate|SaleAmount|CustomerName|CompanyName|Sector|Industry|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|1914 |13729 |2009-01-01|2009-01-01|9|1800|0.08|200|1640.96|Matt Bertelsons|The Priceline Group Inc.|Miscellaneous|Business Services|
|4031 |28774 |2009-01-01|2009-01-01|32|6400|0.1|200|5707.87|Jessica Thornton|Garmin Ltd.|Capital Goods|Industrial Machinery/Components|
|1279 |9285 |2009-01-02|2009-01-01|3|480|0.06|160|447.11|David O'Rourke|Wynn Resorts Limited|Consumer Services|Hotels/Resorts|
|5272 |37537 |2009-01-02|2009-01-01|4|500|0|125|495.47|Alan Brumley|Bed Bath & Beyond Inc.|Consumer Services|Home Furnishings|
|5273 |37537 |2009-01-02|2009-01-01|43|5375|0.07|125|4953.46|Alan Brumley|Bed Bath & Beyond Inc.|Consumer Services|Home Furnishings|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Showing the first 1000 rows.

```

`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")\n    .map(lambda r: LabeledPoint(r[1], [r[0]]))\n    .toDF()
```

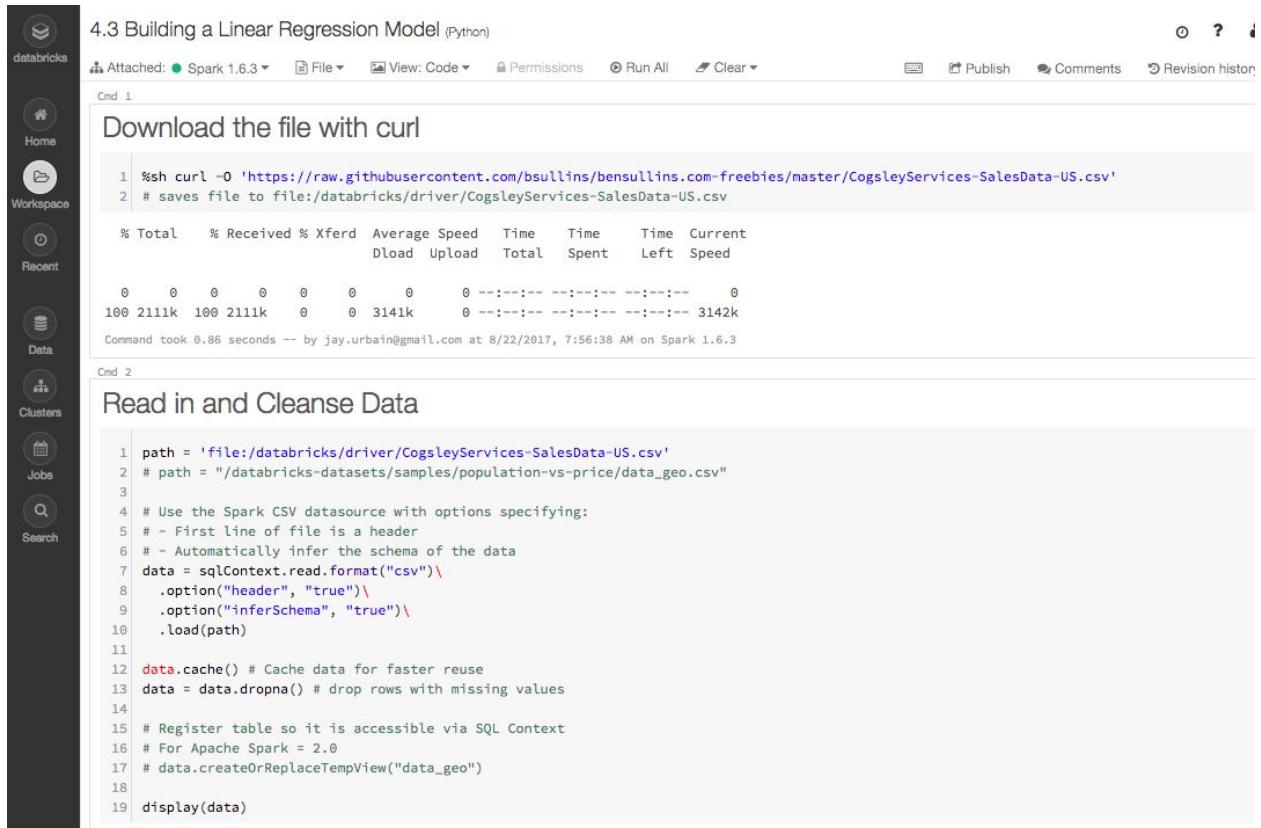
- By applying map to a dataframe, you get a result in the form of an RDD

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



4.3 Building a Linear Regression Model (Python)

Attached: ● Spark 1.6.3 ▾ File ▾ View: Code ▾ Permissions Run All Clear

Cmd 1

Download the file with curl

```
1 %sh curl -O 'https://raw.githubusercontent.com/benjamins/bensullins.com-freebies/master/CogsleyServices-SalesData-US.csv'
2 # saves file to file:/databricks/driver/CogsleyServices-SalesData-US.csv

% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total Spent    Left  Speed
0       0      0      0      0      0      0      0      0      0
100 2111k  100 2111k      0      0  3141k      0      0      0      0      0
Command took 0.88 seconds -- by jay.urban@gmail.com at 8/22/2017, 7:56:38 AM on Spark 1.6.3
```

Cmd 2

Read in and Cleanse Data

```
1 path = 'file:/databricks/driver/CogsleyServices-SalesData-US.csv'
2 # path = "/databricks-datasets/samples/population-vs-price/data_geo.csv"
3
4 # Use the Spark CSV datasource with options specifying:
5 # - First line of file is a header
6 # - Automatically infer the schema of the data
7 data = sqlContext.read.format("csv")\
8   .option("header", "true")\
9   .option("inferSchema", "true")\
10  .load(path)
11
12 data.cache() # Cache data for faster reuse
13 data = data.dropna() # drop rows with missing values
14
15 # Register table so it is accessible via SQL Context
16 # For Apache Spark = 2.0
17 # data.createOrReplaceTempView("data_geo")
18
19 display(data)
```

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 regularization
```

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

```
# Make predictions on data
```

```
predictionsA = modelA.transform(data)
predictionsB = modelB.transform(data)
```

```
display(predictionsA)
```

The screenshot shows a Jupyter Notebook cell with the title "Build Linear Regression Model". The cell contains Python code for fitting two linear regression models with different regularization parameters (0.0 and 100.0) and then displaying the predictions. Below the code is a table showing the features, label, and prediction for each data point. The command took 9.81 seconds to run.

```
1 # Import LinearRegression class
2 from pyspark.ml.regression import LinearRegression
3
4 # Define LinearRegression algorithm
5 lr = LinearRegression()
6
7 # Fit 2 models, using different regularization parameters
8 modelA = lr.fit(data, {lr.regParam:0.0})
9 modelB = lr.fit(data, {lr.regParam:100.0})
10
11 # Make predictions
12 predictionsA = modelA.transform(data)
13 predictionsB = modelB.transform(data)
14
15 display(predictionsA)
```

features	label	prediction
[{"type":1,"size":1,"indices":[],"values":[20090101]}]	734559.3599999996	604138.9286542917
[{"type":1,"size":1,"indices":[],"values":[20090201]}]	539887.7999999998	604174.610447512
[{"type":1,"size":1,"indices":[],"values":[20090301]}]	559449.9600000002	604210.2922407314
[{"type":1,"size":1,"indices":[],"values":[20090401]}]	614983.31	604245.9740339518
[{"type":1,"size":1,"indices":[],"values":[20090501]}]	637481.3899999998	604281.6558271721
[{"type":1,"size":1,"indices":[],"values":[20090601]}]	555516.2199999999	604317.3376203915
[{"type":1,"size":1,"indices":[],"values":[20090701]}]	670807.13	604353.0194136119
[{"type":1,"size":1,"indices":[],"values":[20090801]}]	660353.6399999995	604388.7012068313
[{"type":1,"size":1,"indices":[],"values":[20090901]}]	648992.75	604424.3830000516

Command took 9.81 seconds -- by ben@bensullins.com at 11/14/2016, 2:52:17 PM on unknown cluster

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.

The screenshot shows a Databricks notebook interface. The sidebar on the left includes icons for Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area has a title bar "4.4 Evaluating a Linear Regression Model (Python)". Below the title are navigation buttons for Attached (Spark 1.6.3), File, View, Permissions, Run All, Clear, Publish, Comments, and Revision history. The notebook content is divided into two cells:

Cmd 1: Download the file with curl

```
1 %sh curl -O 'https://raw.githubusercontent.com/bensullins/bensullins.com-freebies/master/CogsleyServices-SalesData-US.csv'
2 # saves file to file:/databricks/driver/CogsleyServices-SalesData-US.csv

% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total   Spent    Left  Speed
0       0      0      0      0      0      0 --::-- --::-- --::-- 0
0       0      0      0      0      0      0 --::-- --::-- --::-- 0
100 2111k 100 2111k 0      0 3163k 0 --::-- --::-- --::-- 3161k

Command took 0.79 seconds -- by jay.urbain@gmail.com at 8/22/2017, 8:18:11 AM on Spark 1.6.3
```

Cmd 2: Read in and Cleanse Data

```
1 path = 'file:/databricks/driver/CogsleyServices-SalesData-US.csv'
2 # path = "/databricks-datasets/samples/population-vs-price/data_geo.csv"
3
4 # Use the Spark CSV datasource with options specifying:
5 # - First line of file is a header
6 # - Automatically infer the schema of the data
7 data = sqlContext.read.format("csv")\
8 .option("header", "true")\
9 .option("inferSchema", "true")\
10 .load(path)
11
12 data.cache() # Cache data for faster reuse
13 data = data.dropna() # drop rows with missing values
14
15 # Register table so it is accessible via SQL Context
16 # For Apache Spark = 2.0
17 # data.createOrReplaceTempView("data_geo")
18
19 display(data)
```

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

Check Models for Accuracy

```
1 from pyspark.ml.evaluation import RegressionEvaluator
2 evaluator = RegressionEvaluator(metricName="rmse")
3
4 RMSE = evaluator.evaluate(predictionsA)
5 print("ModelA: Root Mean Squared Error = " + str(RMSE))
6
7 RMSE = evaluator.evaluate(predictionsB)
8 print("ModelB: Root Mean Squared Error = " + str(RMSE))
9
```

▶ ▷ ⏪ ⏴ ⏵ ⏴ ×

▶ (2) Spark Jobs

```
ModelA: Root Mean Squared Error = 70038.3200417
ModelB: Root Mean Squared Error = 70038.3202721
```

Command took 3.02 seconds -- by jay.urbain@gmail.com at 8/22/2017, 8:19:24 AM on Spark 1.6.3

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:

4.5 Visualizing a Linear Regression Model (Python)

Attached: ● Spark 1.6.3 ▾ File ▾ View: Code ▾ Permissions ■ Stop Execution ⚡ Clear ▾

Cmd 1

Download the file with curl

```
1 ssh curl -O 'https://raw.githubusercontent.com/bensullins/bensullins.com-freebies/master/CogsleyServices-SalesData-US.csv'
2 # saves file to file:/databricks/driver/CogsleyServices-SalesData-US.csv
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
Dload	Upload	Total	Spent	Left	Speed		
0	0	0	0	0	0	0	0
100	2111k	100	2111k	0	0	4035k	0

Command took 0.67 seconds -- by jay.urbain@gmail.com at 8/22/2017, 8:27:01 AM on Spark 1.6.3

Cmd 2

Read in and Cleanse Data

```
1 path = 'file:/databricks/driver/CogsleyServices-SalesData-US.csv'
2 # path = "/databricks-datasets/samples/population-vs-price/data_geo.csv"
3
4 # Use the Spark CSV datasource with options specifying:
5 # - First line of file is a header
6 # - Automatically infer the schema of the data
7 data = sqlContext.read.format("csv")\
8     .option("header", "true")\
9     .option("inferSchema", "true")\
10    .load(path)
11
12 data.cache() # Cache data for faster reuse
13 data = data.dropna() # drop rows with missing values
14
15 # Register table so it is accessible via SQL Context
16 # For Apache Spark = 2.0
17 # data.createOrReplaceTempView("data_geo")
18
19 display(data)
```

▶ ▷ ⏪ ⏴ ⏵ ⏴ ×

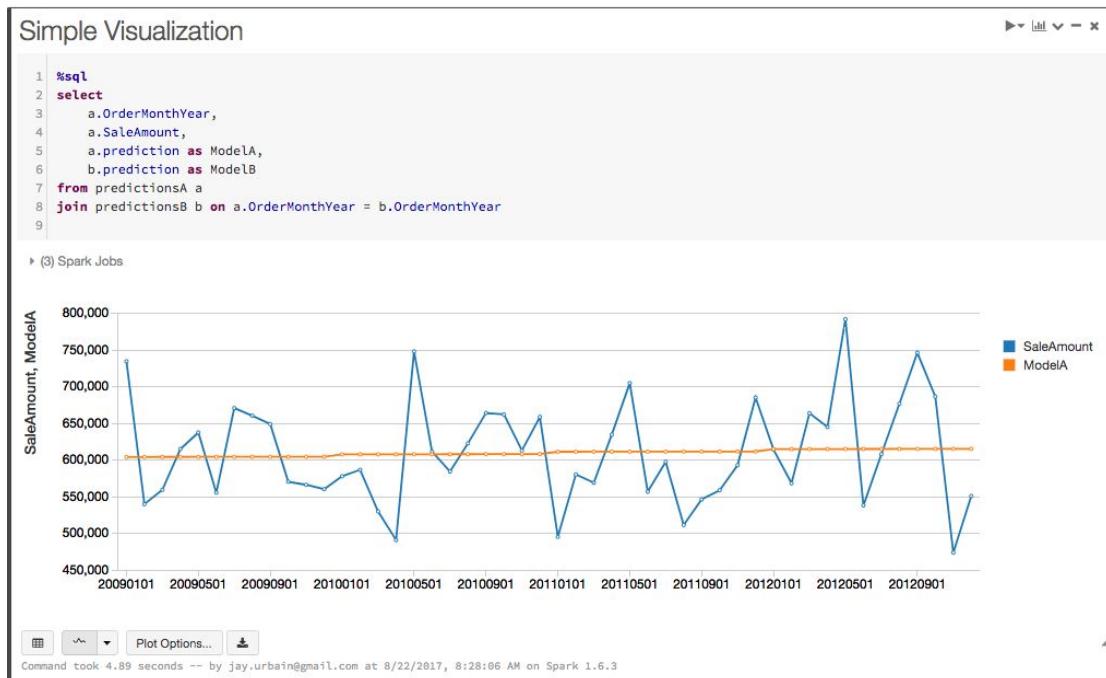
Execute each cell down to Create Tables with Predictions

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

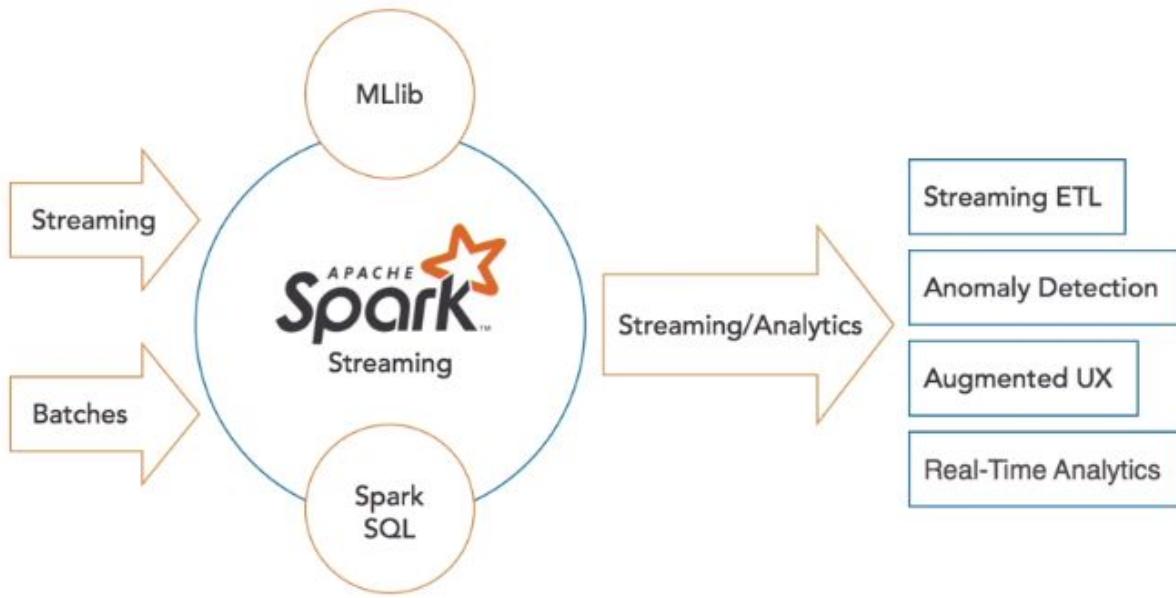
Simple Visualizations

%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 MLlib
- Setup streaming job

Streaming context setup

Load the Setting of Streaming Context notebook.

Use a 2.2 cluster for Spark Streaming

5.2 Setting up Streaming Context (Python)

Attached: SparkIntroClusX1 File View: Code Permissions Run All Clear Publish Comments Revision hist

Cmd 1

Download Sales Data

```
1 %sh
2 curl -O 'https://raw.githubusercontent.com/bsullins/bensullins.com-freebies/master/sales_log.zip'
3 file sales_log.zip

% Total    % Received % Xferd  Average Speed   Time   Time   Time Current
          Dload  Upload Total Spent   Left Speed
0       0     0      0      0      0      0 --::-- --::-- --::-- 0
100  245k  100  245k      0      0  685k      0 --::-- --::-- --::-- 686k
sales_log.zip: Zip archive data, at least v1.0 to extract
Command took 0.53 seconds -- by jay.urbain@gmail.com at 8/22/2017, 9:21:57 AM on Spark 1.6.3
```

Cmd 2

```
1 %sh unzip sales_log.zip
Archive: sales_log.zip
```

Cmd 3

```
1 %fs ls 'file:/databricks/driver/sales_log/'
```

path	name	size
file:/databricks/driver/sales_log/sales-79.csv	sales-79.csv	9276
file:/databricks/driver/sales_log/sales-65.csv	sales-65.csv	9215
file:/databricks/driver/sales_log/sales-13.csv	sales-13.csv	9089
file:/databricks/driver/sales_log/sales-51.csv	sales-51.csv	9206
file:/databricks/driver/sales_log/sales-9.csv	sales-9.csv	9180
file:/databricks/driver/sales_log/sales-14.csv	sales-14.csv	9315
file:/databricks/driver/sales_log/sales-26.csv	sales-26.csv	9206
file:/databricks/driver/sales_log/sales-27.csv	sales-27.csv	9189
file:/databricks/driver/sales_log/sales-21.csv	sales-21.csv	9242

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

Downloads sales data, nzip 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

```
1 from pyspark.sql.types import *
2
3 path = "file:/databricks/driver/sales_log/"
4
5 # create schema for data so stream processing is faster
6 salesSchema = StructType([
7     StructField("OrderID", DoubleType(), True),
8     StructField("OrderDate", StringType(), True),
9     StructField("Quantity", DoubleType(), True),
10    StructField("DiscountPct", DoubleType(), True),
11    StructField("Rate", DoubleType(), True),
12    StructField("SaleAmount", DoubleType(), True),
13    StructField("CustomerName", StringType(), True),
14    StructField("State", StringType(), True),
15    StructField("Region", StringType(), True),
16    StructField("ProductKey", StringType(), True),
17    StructField("RowCount", DoubleType(), True),
18    StructField("ProfitMargin", DoubleType(), True)])
19
20 # Static DataFrame containing all the files in sales_log
21 data = (
22     spark
23         .read
24         .schema(salesSchema)
25         .csv(path)
26 )
27
28
29 # create table so we can use SQL
30 data.createOrReplaceTempView("sales")
31
32 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

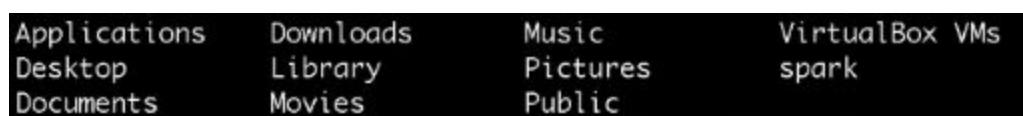
Use prebuilt version

The screenshot shows the Apache Spark download page. At the top, there's a navigation bar with links for 'Download', 'Libraries', 'Documentation', 'Examples', 'Community', and 'FAQ'. To the right of the navigation bar is a link to 'Apache Software Foundation'. Below the navigation bar, there's a section titled 'Download Apache Spark™' with a sub-section for 'Our latest stable version is Apache Spark 2.0.1, released on Oct 3, 2016'. There's a numbered list of steps: 1. Choose a Spark release: 1.6.3 (Nov 07 2016), 2. Choose a package type: Pre-built for Hadoop 2.6, 3. Choose a download type: Direct Download, 4. Download Spark: spark-1.6.3-bin-hadoop2.6.tgz, and 5. Verify this release using the 1.6.3 signatures and checksums and project release KEYS. Below this list is a note: 'Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build with Scala 2.10 support.' On the right side of the page, there's a sidebar with 'Latest News' (listing releases 1.6.3, 2.0.1, 2.0.0, and 1.6.2) and a 'Download Spark' button. Below the news is a section for 'Built-in Libraries' with links to 'SQL and DataFrames', 'Spark Streaming', 'MLlib (machine learning)', 'GraphX (graph)', and 'Third-Party Packages'.

Download Java SDK if not present

Unzip Spark

Open terminal window in root of spark directory.



```
~ $ cd spark/
spark $ ls
CHANGES.txt      R           bin          ec2          licenses
LICENSE          README.md    conf         examples    python
NOTICE          RELEASE      data         lib         sbin
```

Run pyspark:

```
spark $ ./bin/pyspark
Python 2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
16/11/14 14:17:10 INFO SparkContext: Running Spark version 1.6.3
16/11/14 14:17:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
16/11/14 14:17:11 INFO SecurityManager: Changing view acls to: bensullins
16/11/14 14:17:11 INFO SecurityManager: Changing modify acls to: bensullins
16/11/14 14:17:11 INFO SecurityManager: securityManager disabled; ui acls disabled; users with view permissions: Set(bensullins); users with modify permissions: Set
```

Verify:

```
Welcome to
   _/ \
  / \  _ \  / \  _ \  / \
 / \ / . \ \ , / \ / / \ \ \  version 1.6.3
 / \
 / \ 

Using Python version 2.7.10 (default, Oct 23 2015 19:19:21)
SparkContext available as sc, HiveContext available as sqlContext.
>>> sc
<pyspark.context.SparkContext object at 0x10baf3450>
>>>
```

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