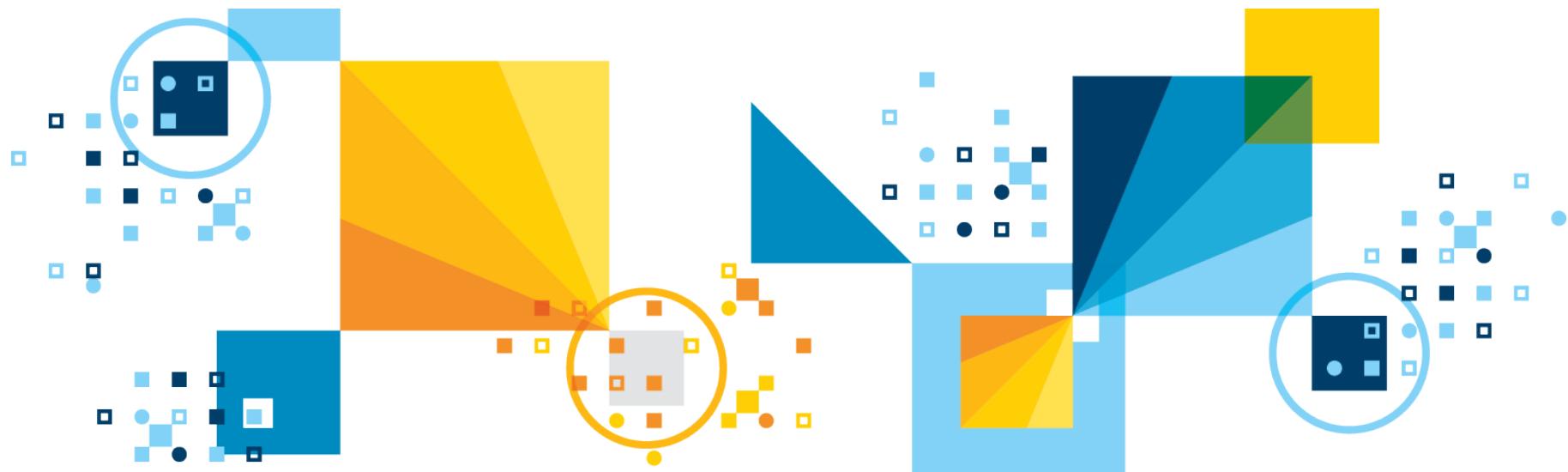


Tanaka Y.P
2016-09-03

SparkとJupyter Notebookを使った分析処理

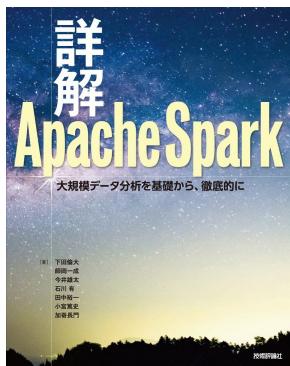


自己紹介

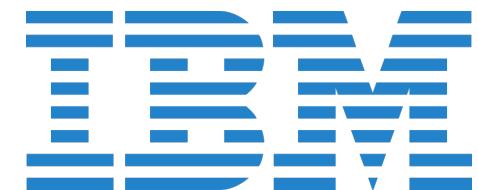
田中裕一 (yuichi tanaka)



主にアーキテクチャとサーバーサイドプログラムを担当することが多い。Hadoop/Spark周りをよく触ります。Node.js、Python、最近はSpark周りの仕事でScalaを書くことが多い気がします。
休日はOSS周りで遊んだり。



詳解 Apache Spark



アジェンダ

- Sparkの概要
- Jupyter notebookの概要
- 本日のサンプル（分類器）
- [Notebook](#)

Sparkの概要

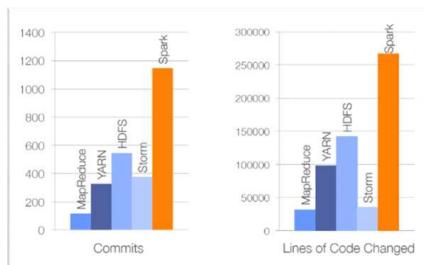


Sparkの歴史

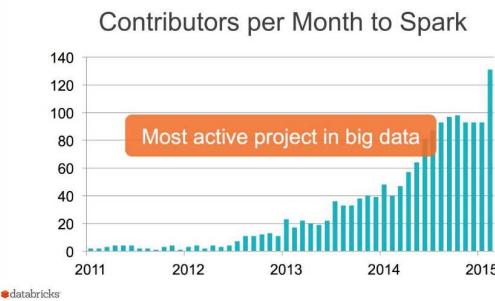
All Releases

- Spark 1.5.1 (Oct 02 2015)
- Spark 1.5.0 (Sep 09 2015)
 - Spark 1.4.1 (Jul 15 2015)
 - Spark 1.4.0 (Jun 11 2015)
 - Spark 1.3.1 (Apr 17 2015)
 - Spark 1.3.0 (Mar 13 2015)
 - Spark 1.2.2 (Apr 17 2015)
 - Spark 1.2.1 (Feb 09 2015)
 - Spark 1.2.0 (Dec 18 2014)
 - Spark 1.1.1 (Nov 26 2014)
 - Spark 1.1.0 (Sep 11 2014)
 - Spark 1.0.2 (Aug 05 2014)
 - Spark 1.0.1 (Jul 11 2014)
 - Spark 1.0.0 (May 30 2014)
 - Spark 0.9.2 (Jul 23 2014)
 - Spark 0.9.1 (Apr 09 2014)
 - Spark 0.9.0 (Feb 02 2014)
 - Spark 0.8.1 (Dec 19 2013)
 - Spark 0.8.0 (Sep 25 2013)
 - Spark 0.7.3 (Jul 16 2013)
 - Spark 0.7.2 (Feb 06 2013)
 - Spark 0.7.0 (Feb 27 2013)

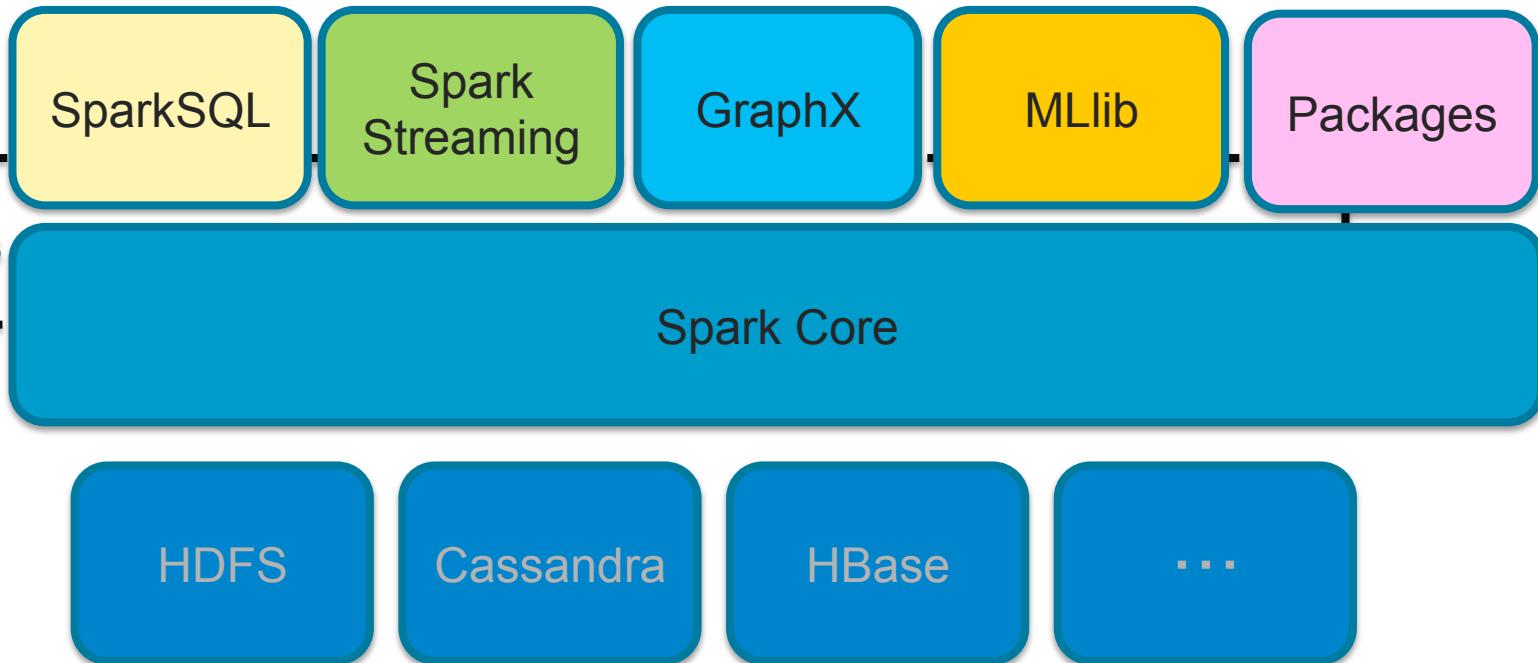
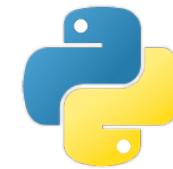
- Sparkは、2009年にUC Berkeley AMPLabのプロジェクトとしてスタート、2010年にオープンソース化され、現在では、Apache Software Foundationで現在最もアクティブなプロジェクトの一つ。
- AMPLabで開発していたメンバーが中心となって Databricks社を設立し、コミュニティをリード。



Activity for 6 months in 2014
(from Matei Zaharia - 2014 Spark Summit)

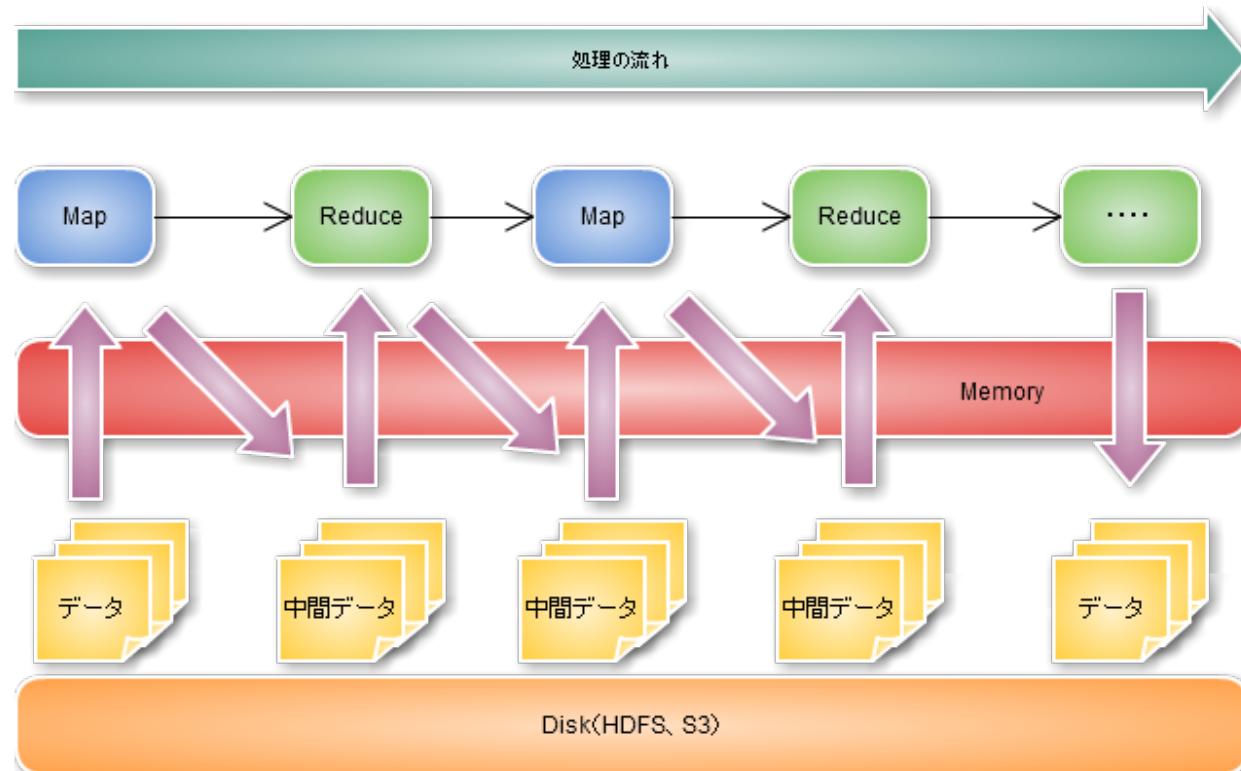


Sparkのテクノロジースタック



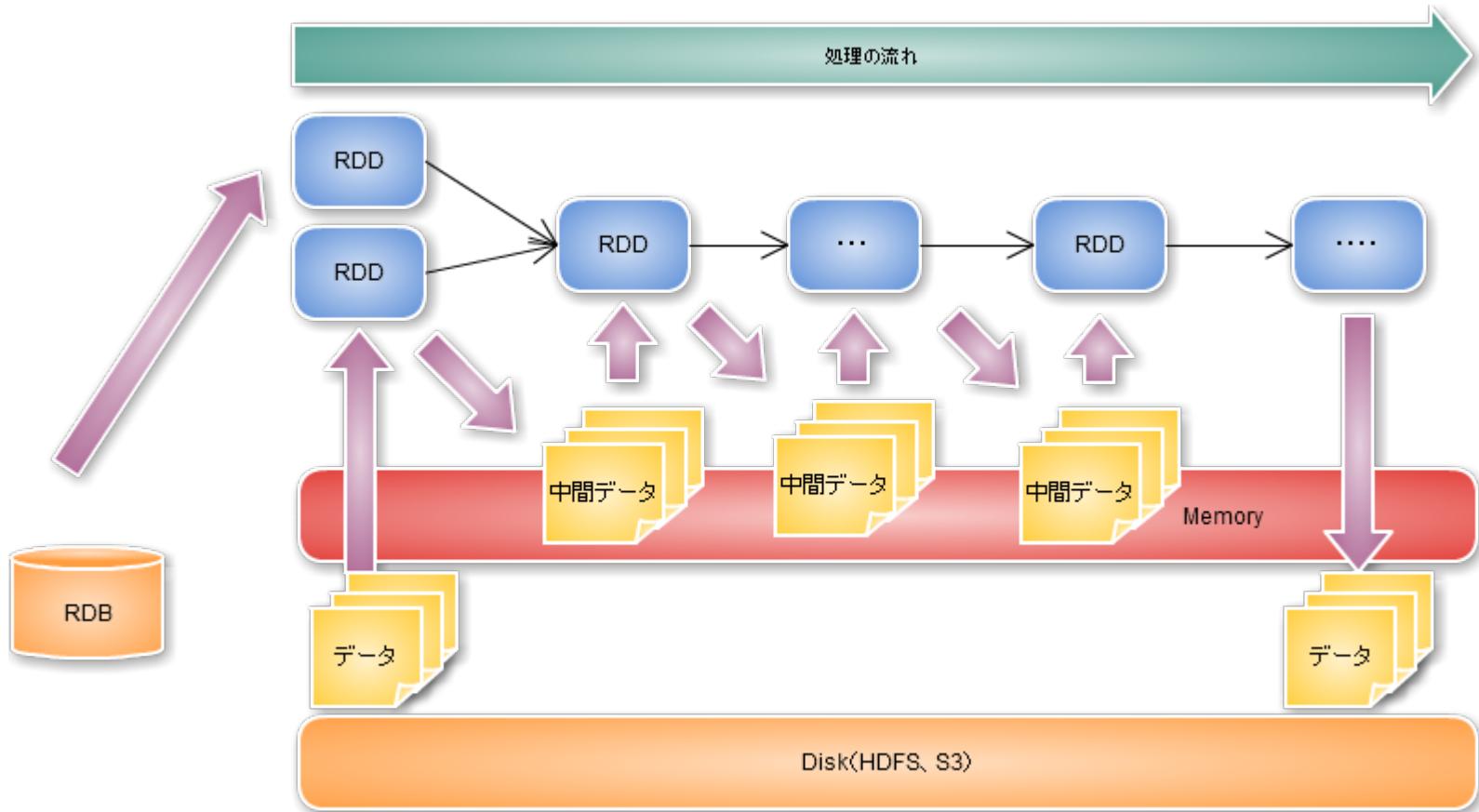
Apache SparkとHadoop

HadoopでのMapReduceの処理例



Apache Sparkの処理概要

SparkでのRDD&DAGの処理例

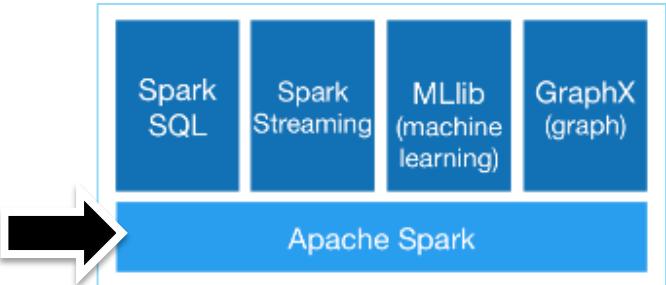


Spark Core

▪ Spark CoreはSparkのエンジン

- Java, Scala, Pythonを利用してETLを実行可能
- RDD(Reslient Distributed Datasets)はScalaのコレクションのSeqのようなもので、データを順番に保持
- RDDの内部はパーティションに分かれている。パーティション毎にデータを保持(HDFSブロック数に依存)

- 分散処理する際にはパーティション毎に並列に処理
- mapやfilter等の基本的な操作の場合、データの順序は変わらない。



例：平均気温の計算

```
20150614 22:00:00,0,1,8,20.9,8,3.0,8,南,南,西,8,85,8
20150614 23:00:00,0,1,8,20.9,8,2.6,8,南,南,西,8,86,8
20150615 00:00:00,0,1,8,20.5,8,1.0,8,南,8,86,8
20150615 1:00:00,0,1,8,20.4,8,0.7,8,南,8,88,8
```



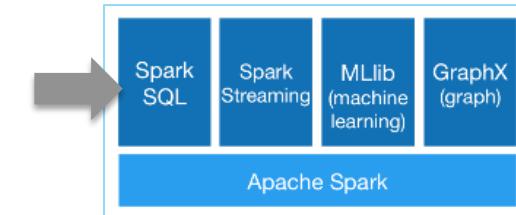
```
val csv = spark.textFile("tokyo.csv")
val pairs = csv.map(line => (line.split(",")))
    .map(x => (x(0).take(8), (x(4).toFloat, 1)))
    .reduceByKey( (x,y) => (x._1 + y._1, x._2 + y._2) )
    .map(x => (x._1, x._2._1/x._2._2) )
    .sortByKey()
```



```
(2015/6/14,22.565218)
(2015/6/15,24.550001)
(2015/6/16,23.358332)
(2015/6/17,21.583334)
```

Spark SQL

- SparkSQLによるデータ操作
 - **SQLを利用したデータ操作が可能**
 - トランザクションなし
 - Parquet、Json、Hive だけでなく JDBCやODBCもサポート
 - Thrift JDBC/ODBCによる外部からの接続
 - 後述のDataFrameをラップする形で実装



例：住所データ(json)からの特定データの抽出

```
{"name": "貝嶋", "address": {"city": "川崎", "state": "神奈川"}}  
{"name": "土屋", "address": {"city": "豊洲", "state": "東京"}}  
{"name": "山田", "address": {"city": "横浜", "state": "神奈川"}}  
{"name": "岸代", "address": {"city": "後楽園", "state": "東京"}}
```



```
val people = sqlContext.jsonFile("test.json")  
people.registerTempTable("people")  
val nameAndAddress =  
    sqlContext.sql("SELECT name, address.city, address.state FROM  
people WHERE address.state=%s")  
nameAndAddress.collect.foreach(println)
```

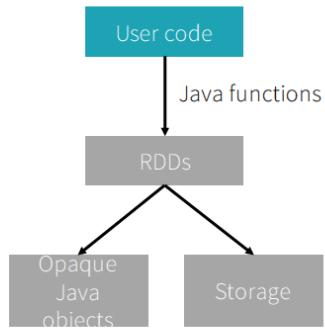


```
{"name": "貝嶋", "address": {"city": "川崎", "state": "神奈川"}}  
{"name": "山田", "address": {"city": "横浜", "state": "神奈川"}}
```

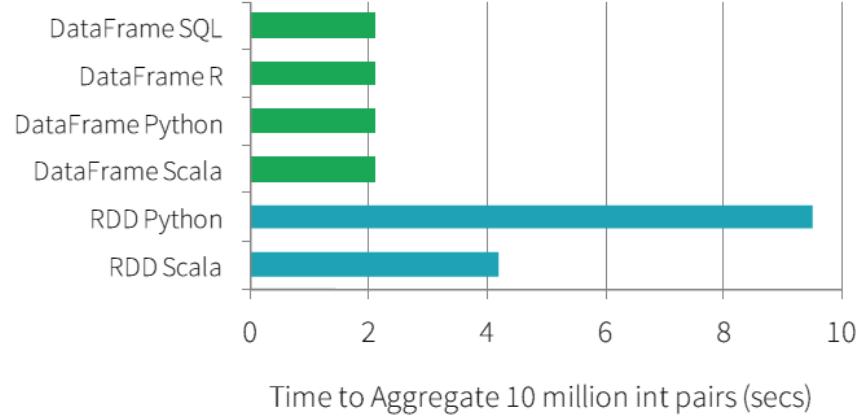
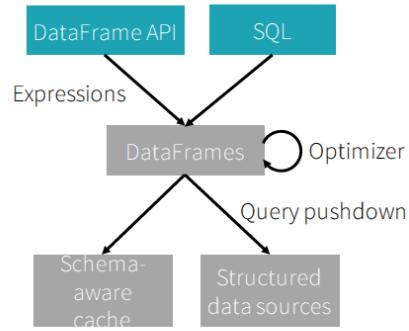
DataFrame API

- **Performance**
- **Less Code**

Traditional Spark



DataFrames



 databricks

DataFrame API

- Performance
- Less Code



```
private IntWritable one =
    new IntWritable(1)
private IntWritable output =
    new IntWritable()
protected void map(
    LongWritable key,
    Text value,
    Context context) {
    String[] fields = value.split("\t")
    output.set(Integer.parseInt(fields[1]))
    context.write(one, output)
}

IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable()

protected void reduce(
    IntWritable key,
    Iterable<IntWritable> values,
    Context context) {
    int sum = 0
    int count = 0
    for(IntWritable value : values) {
        sum += value.get()
        count++
    }
    average.set(sum / (double) count)
    context.write(key, average)
}
```



Spark RDD

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

SQL

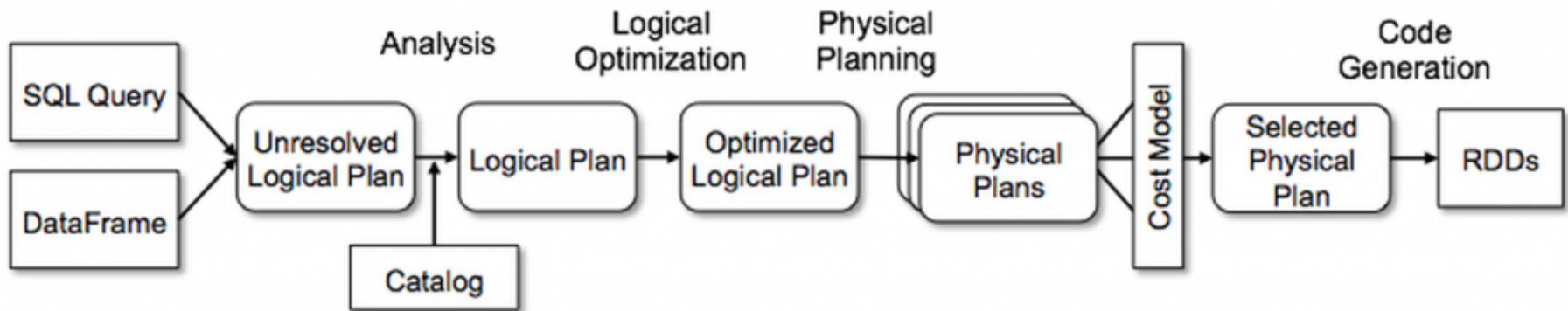
```
SELECT name, avg(age)
FROM people
GROUP BY name
```

Spark DataFrame API

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

Catalyst Optimizaについて補足

CatalystはSQL QueryやDataFrameAPIで作られた処理を最適化し
RDDベースで処理するためのコードを生成します

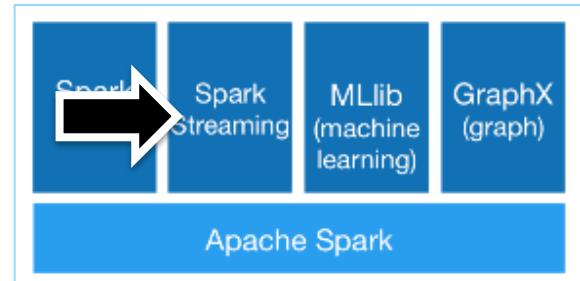


Spark Streaming

▪ Sparkでストリーム処理

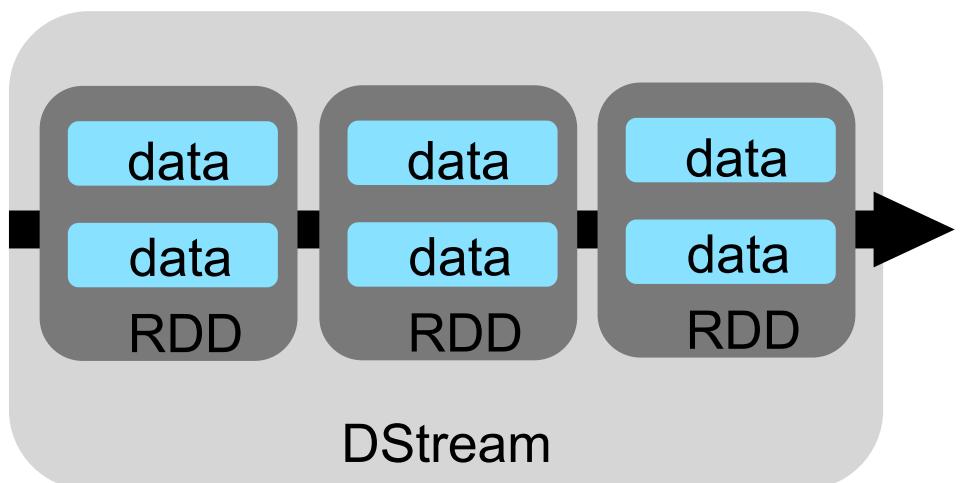
- Sparkによるミニ（マイクロ）バッチの実行
- DStreamと呼ばれるRDDを操作
 - 指定間隔ごとにまとめられたRDDを処理
(Windows処理も可能)
 - 通常のSparkプログラミングとほぼ同様

たとえば、定期的に流入するデータの「移動平均値」の連続計算



例：センサーデータの出力値変更時にアラート

センサーデータ：
 (Dev1, 201501010000, 0)
 (Dev2, 201501010000, 0)
 (Dev1, 201501010001, 1)



```
val tstream = ssc.socketTextStream(hostname, port)
var mdtxt = tstream.map(x => x.split(","))
  .map(x => (x(0), (x(0), x(1), x(2).toInt) ))
  .updateStateByKey(updateFunc _)
mdtxt.print()
```



Alert: Dev1 Status changed : 1



SparkR, MLLib

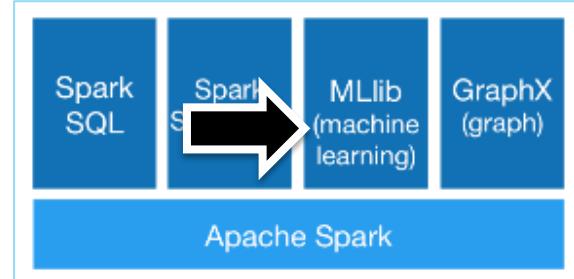
- Sparkで機械学習

- MLlibとRが利用可能

MLlibはScalaで、SparkRはRで

記述可能

- アルゴリズム(MLlib)
- SVM、ロジスティック回帰、決定木、
K-means、ALSなど
- IBMはSystemMLをSparkに提供

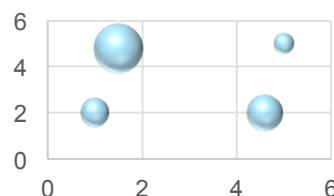


例：顧客のクラスタ分け

データ：(直近購買月[n日前], 期間内購買回数)
 $(5,1),(4,2),(5,3),(1,2),(2,4),(2,5),(2,6),(1,4),(1,5),(1,2),(1,5),(5,5)$

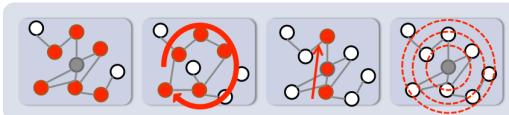
```
val data = spark.textFile("kdata.txt")
val parsedData = data.map(x =>
    Vectors.dense(x.split(',').map(_.toDouble))).cache()
val numClusters = 3
val numIterations = 10
val clusters = KMeans.train(parsedData, numClusters, numIterations)
```

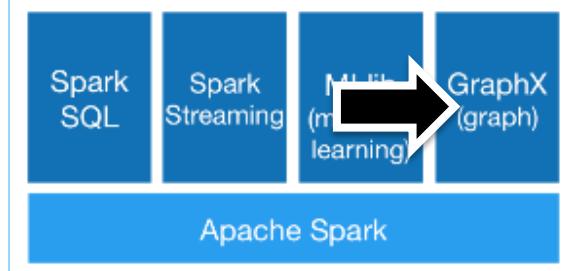
クラスタ結果：([中心], 人数)
 $([1.0, 2.0], 2), ([1.5, 4.833333333333333], 6), ([4.666666666666666, 2.0], 3), ([5.0, 5.0], 1)$



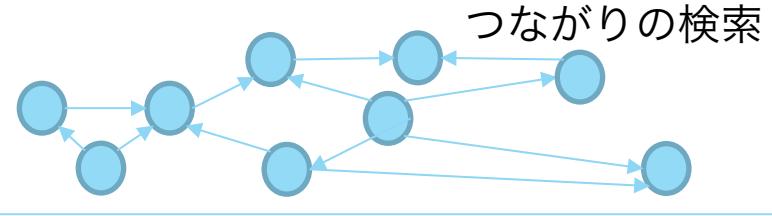
Spark GraphX

▪ Sparkでグラフ処理を

- グラフデータを並列分散環境で処理するためのフレームワーク
- グラフ構造データを用いた解析を行う
 - 「点」と「辺」からなるデータ
 - SNSでのつながり、データ間の関連性など
- 表構造では扱うことが難しい関係を見つける
 - データ間のつながりの抽出
 - 輪の抽出
 - 距離の計測
 - 影響の計測
- グラフDBとの兼ね合い（これから）



例： つながりと距離を見つけ出す



```
val graphWithDistance = Pregel(
  graph.mapVertices((id:VertexId, attr:Int) => List((id, 0))),
  List[(VertexId, Int)](),
  Int.MaxValue, EdgeDirection.Out)((id, attr, msg) =>
  mergeVertexRoute(attr, msg.map(a=> (a._1, a._2 + 1))), edge => {
    val isCyclic = edge.srcAttr.filter(_._1 == edge.dstId).nonEmpty
    if(isCyclic) Iterator.empty
    else Iterator((edge.dstId, edge.srcAttr))
  },(m1, m2) => m1 ++ m2
)
```

1,((1,0), (6,1), (9,1), (7,1), (4,2))

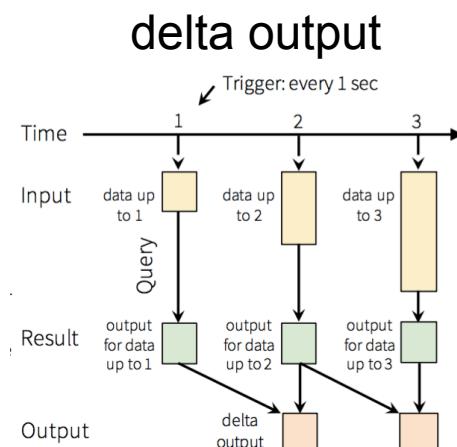
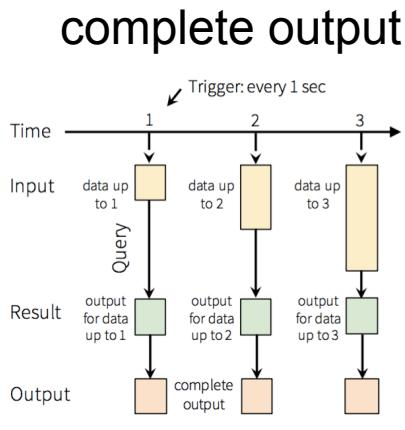
DataSet API

- Spark **v1.6**で追加された新しいAPI
 - まだ**実験的**な実装であることに注意
- 登場背景
 - RDDとDataFrameという二つの抽象概念ができてしまった。
 - RDDとDataFrameにそれぞれ長所があること
 - 2つの抽象概念を行き来する為のコストがかかる
- 二つの抽象概念をいいとこ取りしたDataSetAPIの登場
 - DataFrameの速さはそのまま
 - オブジェクト・メソッドはコンパイル時のタイプセーフ提供
 - DataFrameとのシームレス変換

Structured Streaming

- Spark v2.0で追加される予定の新しいAPI
- Datasetの上に実装されたHigh-levelのStreaming API
- Streamingデータを構造化データとして

継続的に処理可能



```
Jacek Warszawa, Polska, 42,true  
Jacek Warszawa, Polska, 42,true
```

```
val in =  
spark.readStream .schema(schemaImp) .format("csv") .option("header",  
true) .option("maxFilesPerTrigger", 1) .load("csv-logs")
```

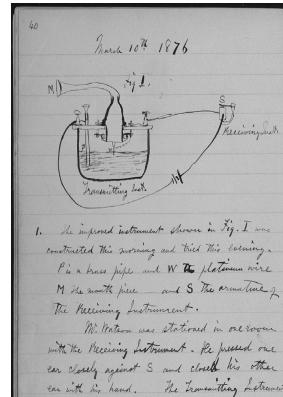
Batch: 0

name	city	country	age	alive
Jacek	Warszawa	Polska	42	true

JupyterはNotebook… “Notebook”とは?

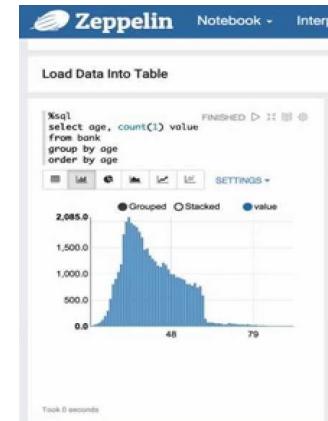
• 紙と鉛筆

- 紙と鉛筆は、これまで長い間、科学者がメモや図面を通して進捗状況を文書化するための重要なツールである:
 - 表現力
 - 累積した情報
 - コラボレーション



• Notebooks

- Notebooks は、これまでの紙と鉛筆のデジタル版であり、再現性のある分析と文書化を可能にする:
 - マークダウンとグラフ化
 - 反復探索
 - 共有が容易

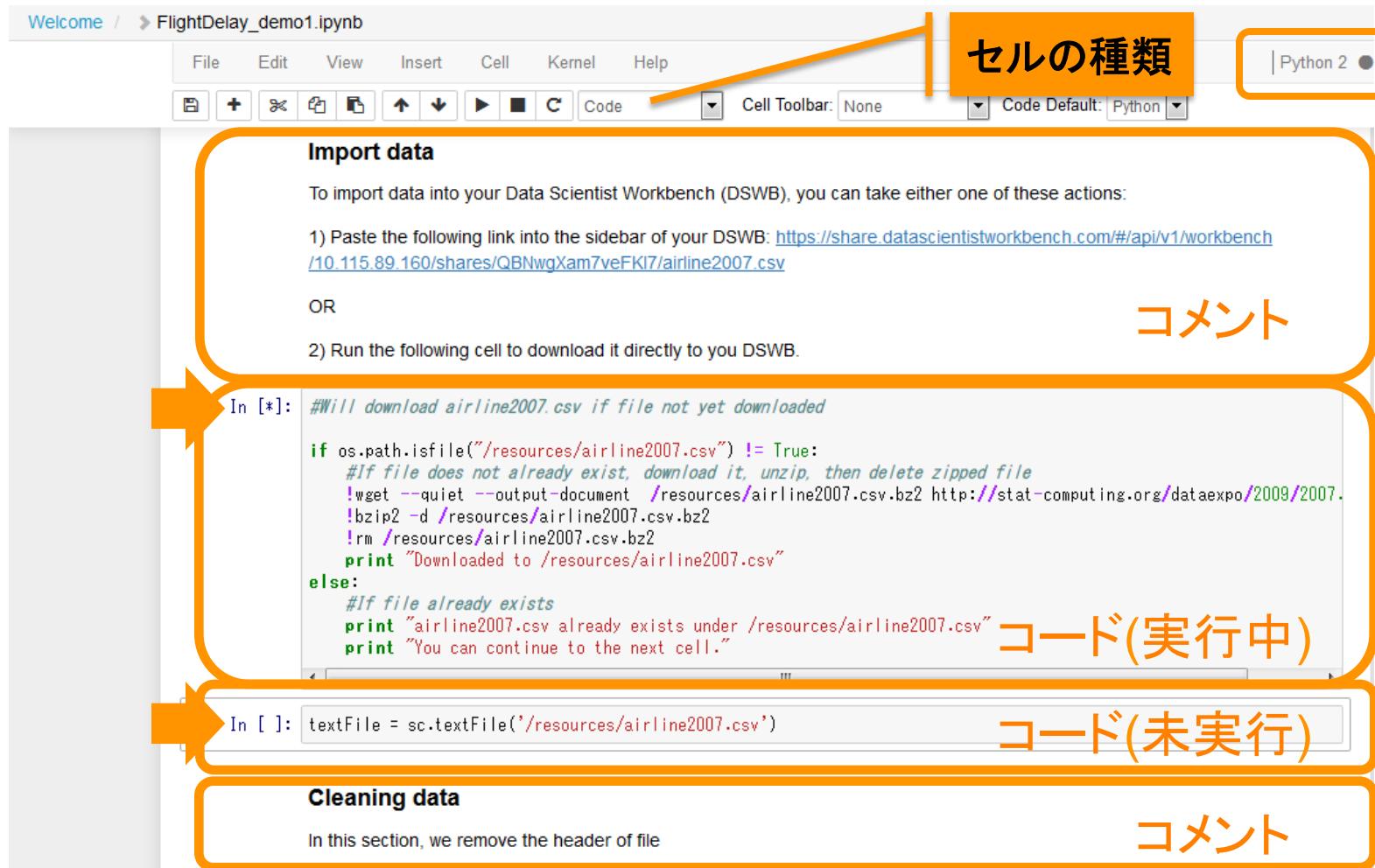


データ整形と分析の実行 「Jupyter Notebook」



- リリース
 - 2001年にリリースされたIPythonをベースに、2015年にJupyterとしてリリース
- ノートブック
 - WebブラウザからのGUI操作可能
 - コード実行、コメント記述、グラフの描画を実行可能
- カーネル
 - Data Scientist Workbenchでは、Scala, Python, Rを実行可能

Jupyterにおけるセル・コメント・コード

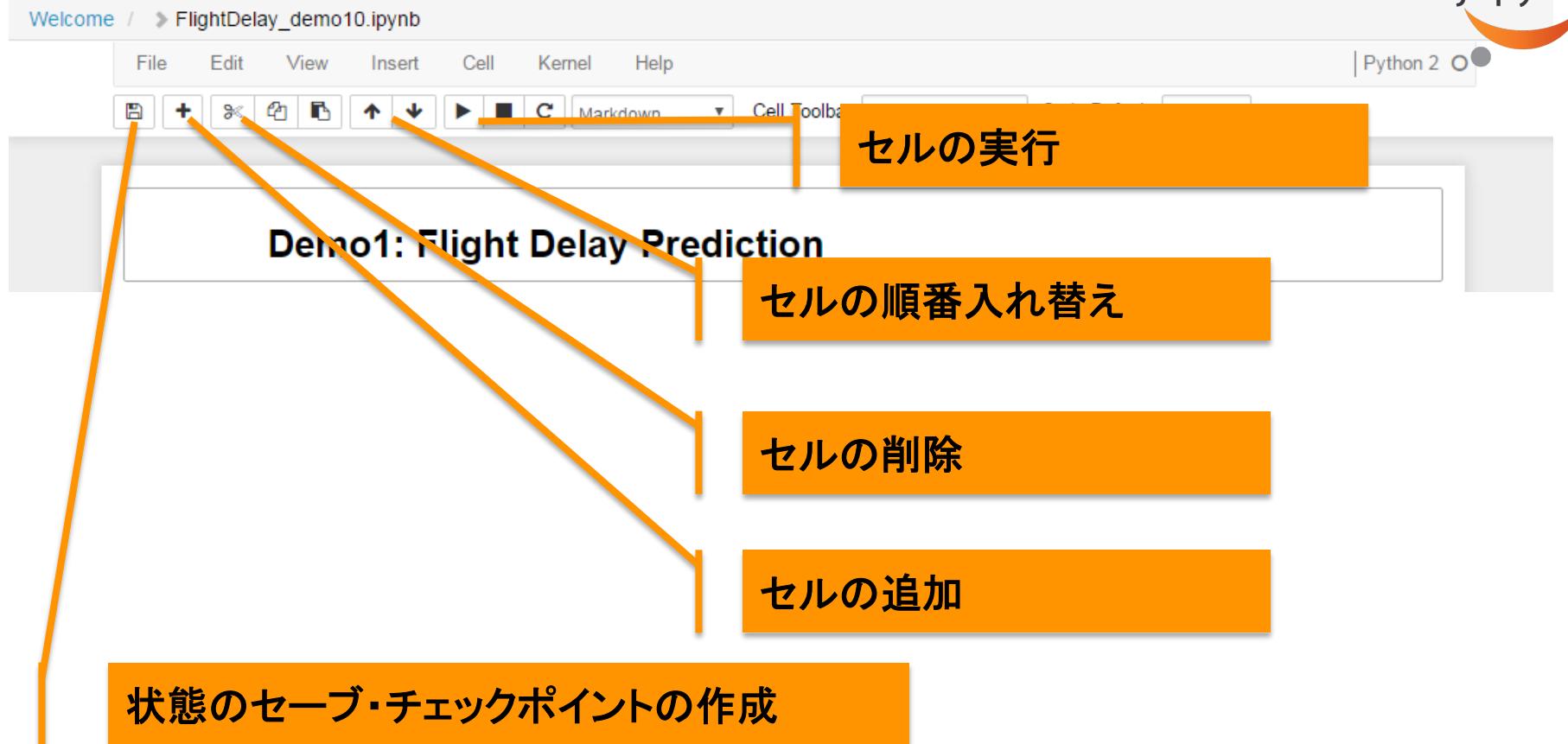


The screenshot shows the Jupyter Notebook interface with the following elements highlighted:

- セルの種類**: A callout pointing to the "Cell" dropdown in the top menu bar.
- Python 2**: A callout pointing to the kernel selection dropdown in the top right corner.
- Import data**: A section header in the sidebar.
- コメント**: A callout pointing to the explanatory text in the sidebar about importing data.
- In [*]:** A code cell containing Python code for downloading a CSV file. The code is annotated with comments explaining its purpose.
- コード(実行中)**: A callout pointing to the status bar at the bottom of the code cell.
- In []:** An empty code cell.
- コード(未実行)**: A callout pointing to the status bar of the empty code cell.
- Cleaning data**: A section header in the sidebar.
- コメント**: A callout pointing to the explanatory text in the sidebar about cleaning data.

```
#Will download airline2007.csv if file not yet downloaded
if os.path.isfile("/resources/airline2007.csv") != True:
    #If file does not already exist, download it, unzip, then delete zipped file
    !wget --quiet --output-document /resources/airline2007.csv.bz2 http://stat-computing.org/dataexpo/2009/2007.
    !bzip2 -d /resources/airline2007.csv.bz2
    !rm /resources/airline2007.csv.bz2
    print "Downloaded to /resources/airline2007.csv"
else:
    #If file already exists
    print "airline2007.csv already exists under /resources/airline2007.csv"
    print "You can continue to the next cell."
```

よく使うJupyterのアイコン



修正したコード(セル)からの再実行



Welcome / > FlightDelay_demo10.ipynb

File Edit View Insert Cell Kernel Help

Cell Toolbar: None Code Default: Python

In [1]: `#!!! download airline2007.csv if file not yet downloaded`

```
if os.path.isfile("/resources/airline2007.csv") != True:
    #If file does not already exist, download it, unzip, then delete zipped file
    !wget --quiet --output-document /resources/airline2007.csv.bz2 http://stat-computing.org/dataexpo/2009/2007.csv.bz2
    !bzip2 -d /resources/airline2007.csv.bz2
    !rm /resources/airline2007.csv.bz2
    print "Downloaded to /resources/airline2007.csv"
else:
    #If file already exists
    print "airline2007.csv already exists under /resources/airline2007.csv"
    print "You can continue to the next cell."
```

airline2007.csv already exists under /resources/airline2007.csv
You can continue to the next cell.

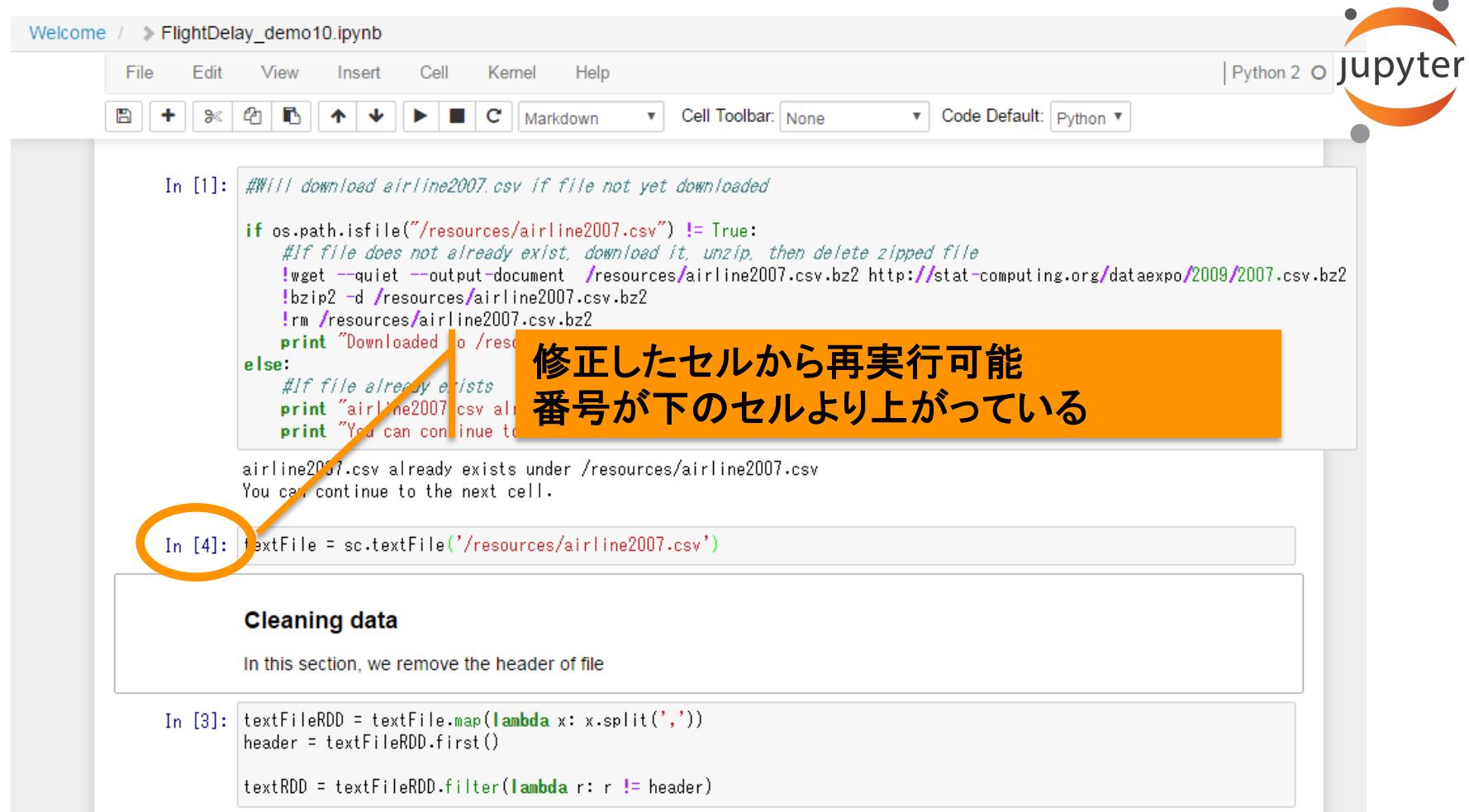
In [2]: `textFile = sc.textFile('/resources/airline2000.csv')`

Cleaning data

In this section, we remove the header of file

In [3]: `textFileRDD = textFile.map(lambda x: x.split(','))`
`header = textFileRDD.first()`
`textRDD = textFileRDD.filter(lambda r: r != header)`

修正したコード(セル)からの再実行



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Welcome / FlightDelay_demo10.ipynb
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Help; Python 2 selected.
- In [1]:** A code cell containing Python script to download and unzip a CSV file if it doesn't exist. It includes comments explaining the steps: checking if the file exists, downloading it if not, unzipping it, and then deleting the zipped file. It also prints messages indicating the download and existence of the file.
- In [4]:** A code cell starting with `textFile = sc.textFile('/resources/airline2007.csv')`. This cell is circled in orange.
- Cleaning data:** A section title followed by a note: "In this section, we remove the header of file".
- In [3]:** A code cell containing two lines of code: `textFileRDD = textFile.map(lambda x: x.split(','))` and `header = textFileRDD.first()`, followed by `textRDD = textFileRDD.filter(lambda r: r != header)`.
- Annotation:** A yellow box with black text: "修正したセルから再実行可能 番号が下のセルより上がっている" (Execution from modified cell is possible, number is higher than the next cell).

コード補完



```
In [ ]: %matplotlib inline  
import matplotlib as plt  
import matplotlib.pyplot as plt  
import numpy as np  
from pyspark.mllib.clustering import KMeans,   
from numpy import array  
from math import sqrt
```

Tabを押下することでコード補完が行われる

お客様情報(csv)を読み込む

```
In [ ]: rawdata = sc.|
```

最初の1行

```
In [ ]: print '[|
```

csvの情報

```
In [ ]: data = rawdata.map(lambda line: np.array([float(x) for x in line.split(',')]))
```

視覚的に理解するためグラフにプロットする

キーボードショートカット

キーボードショートカットによるviライクな操作が可能 (j, kでセル間を移動)
 Enterでセル毎の編集モードに入り、EscでNotebookへのコマンドモードに変更



Keyboard shortcuts

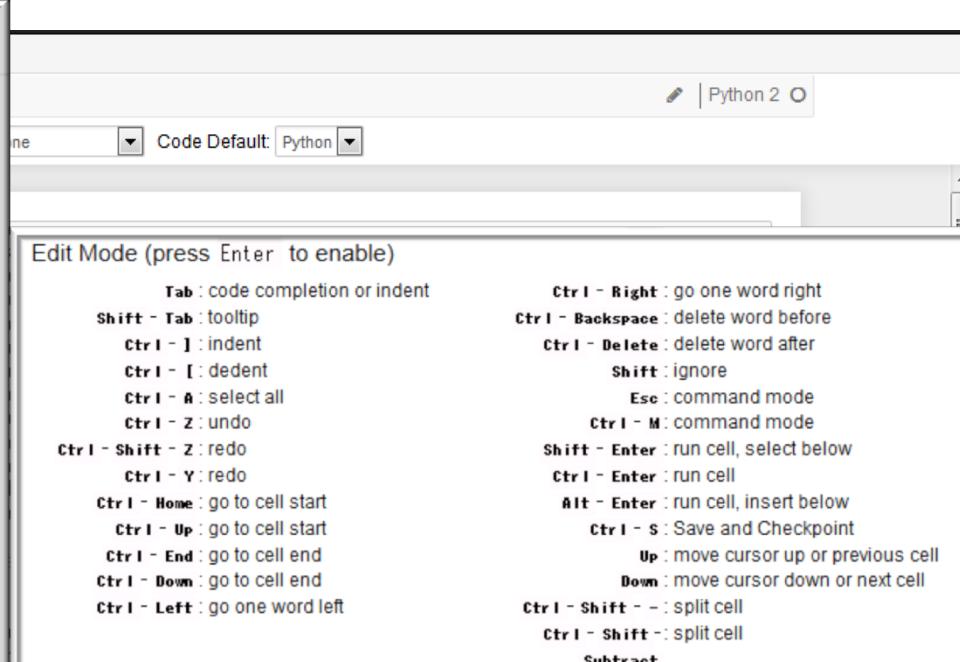
The IPython Notebook has two different keyboard input modes. Edit mode allows you to type code/text into a cell and is indicated by a green cell border. Command mode binds the keyboard to notebook level actions and is indicated by a grey cell border.

Command Mode (press Esc to enable)

Y	: to code
M	: to markdown
R	: to raw
1	: to heading 1
2	: to heading 2
3	: to heading 3
4	: to heading 4
5	: to heading 5
6	: to heading 6
J	: select cell below
A	: insert cell above
B	: insert cell below
X	: cut selected cell
C	: copy selected cell
V	: paste cell below
Z	: undo last cell deletion
S	: Save and Checkpoint
L	: toggle line numbers
O	: toggle output

Edit Mode (press Enter to enable)

Q	: close pager
H	: show keyboard shortcut help dialog
0,0	: restart kernel
Shift	: ignore
Shift - Enter	: run cell, select below
Ctrl - Enter	: run cell
Alt - Enter	: run cell, insert below
Ctrl - S	: Save and Checkpoint
Shift - Space	: scroll up
Enter	: enter edit mode
Shift - V	: paste cell above
Shift - M	: merge cell below
Shift - O	: toggle output scrolling
Space	: scroll down
Up	: select cell above
K	: select cell above
Down	: select cell below
D, D	: delete selected cell
Esc	: close pager
I, I	: interrupt kernel



The screenshot shows the Jupyter Notebook interface with a floating "Keyboard shortcuts" window. The window title is "Keyboard shortcuts". It contains two sections: "Command Mode (press Esc to enable)" and "Edit Mode (press Enter to enable)". The "Edit Mode" section is currently active, displaying a list of keyboard shortcuts for navigating cells and performing operations like saving and pasting. The background shows a notebook cell with a "Python 2" code cell and a toolbar with various icons.

Welcome / ➤ Handson2_noheader.ipynb / ➤ FlightDelay_demo10.ipynb / Search - type:notebook

File Edit View Insert Cell Kernel Help

None Code Default: Python

1) Paste the following link into the side
[/QBNwgXam7veFKI7/airline2007.csv](#)

OR

2) Run the following cell to download it

In []: #Will download airline2007.csv

```
if os.path.exists('airline2007.csv'):
    if file_size('airline2007.csv') > 0:
        !get_ipython().run_line_magic('rm', 'resources/airline2007.csv')
        !bz2 -d resources/airline2007.csv
        print "Downloaded to /resources/airline2007.csv"
else:
    #If file already exists
    print "airline2007.csv already exists"
    print "You can continue to the next cell."
```

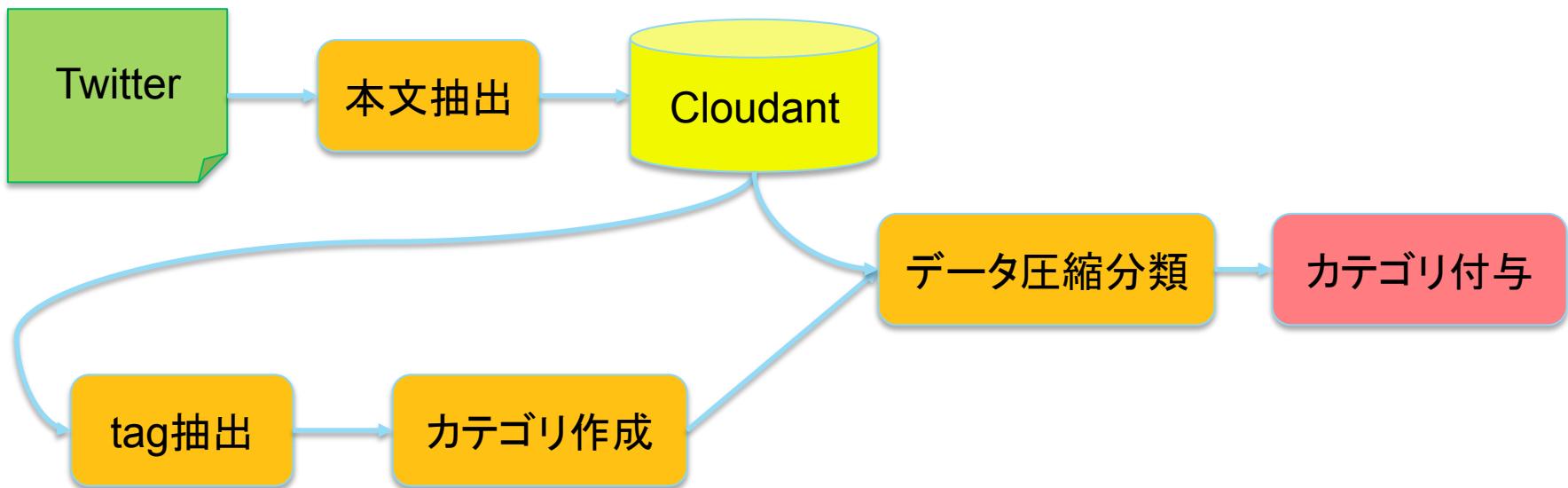
ユーザインターフェースの
細かい解説を確認可能

本日のサンプル（分類器の作成）

- Sparkを使ったTweetの分類を行います。（自動タグリング）
 - SNS分析などで利用される分析
 - 例えば評判の取得や調査の為の関連tweetの抽出などで利用可能
- 短文(tweetなど)の課題
 - kuromojiを使って形態素解析を行う
 - 新語・略語・間違いなどに弱い（というか辞書にないのはできない）
 - kMeansを用いた分類
 - 辞書の精度に依存してしまう
 - 日本語以外の形態素解析が難しい
- データ圧縮による分類
 - 似たような短文を複数集め、それぞれデータ圧縮し分類器を作る
 - 判定したい短文と複数集めた短文をデータ圧縮する
 - どの程度圧縮されたか圧縮比を計算し、圧縮比の高いものと同じ分類とする
→
新語や略語に強い
言語を問わない・画像や映像もいける

本日のサンプル（分類器の作成）

- Sparkを使ったTweetの分類を行います。（自動タグリング）
 - Tweetを取得し、Cloudant(CouchDB)に格納
 - Sparkでtagを抽出し、tag毎のtweet_listを作成
 - 今回タグ付けを行いたいカテゴリとtagのマッピングを作成
 - カテゴリ毎のtweetデータをデータ圧縮分類
 - 分類に基づいてカテゴリ付与



本日のサンプルの反省

1. もうちょっと学習用データのtagを考えれば良かった・・・

- 学習用のデータ収集

- #economy,#経済
- #seiji,#政治
- #sports,#スポーツ
- #anime,#アニメ
- #IT,#エンジニア
- #science,#科学

→

これらのタグは各分野の「ニュース」が大量に含まれてしまい、ワードが似通ってしまった。

経済のタグは実はFX関連が多いなど、タグだけで機械的に学習データを集めると分類結果が・・・

2. Sparkでの実装

1. SparkSQLや前処理部分での恩恵は大きい（バス）
2. 肝心の圧縮部分が分散処理されていない

3. 今日のサンプルの応用

1. Twetterなどの短文、新語や略語が多く含まれる分類には有効
2. 同様にブログ記事など同じ性質を持つものにも有効そう
3. EC系の説明文などを使った自動タグリングにも応用が利きそう

参考

- <http://db-event.jpn.org/deim2011/proceedings/pdf/a1-6.pdf>
- <http://dbsj.org/wp-content/uploads/journal/vol10/no1/dbsj-journal-10-01-001.pdf>