Big Data Processing and Data Science Environment in Python

PyCon PH 2017
February 25, 2017
University of Science & Technology of Southern Philippines

PYCON PH 2017 CDO

Hello!

Rick Bahague

Co-organizer, PyData Philippines
National Coordinator, CP-Union.Org Inc
Data Scientist

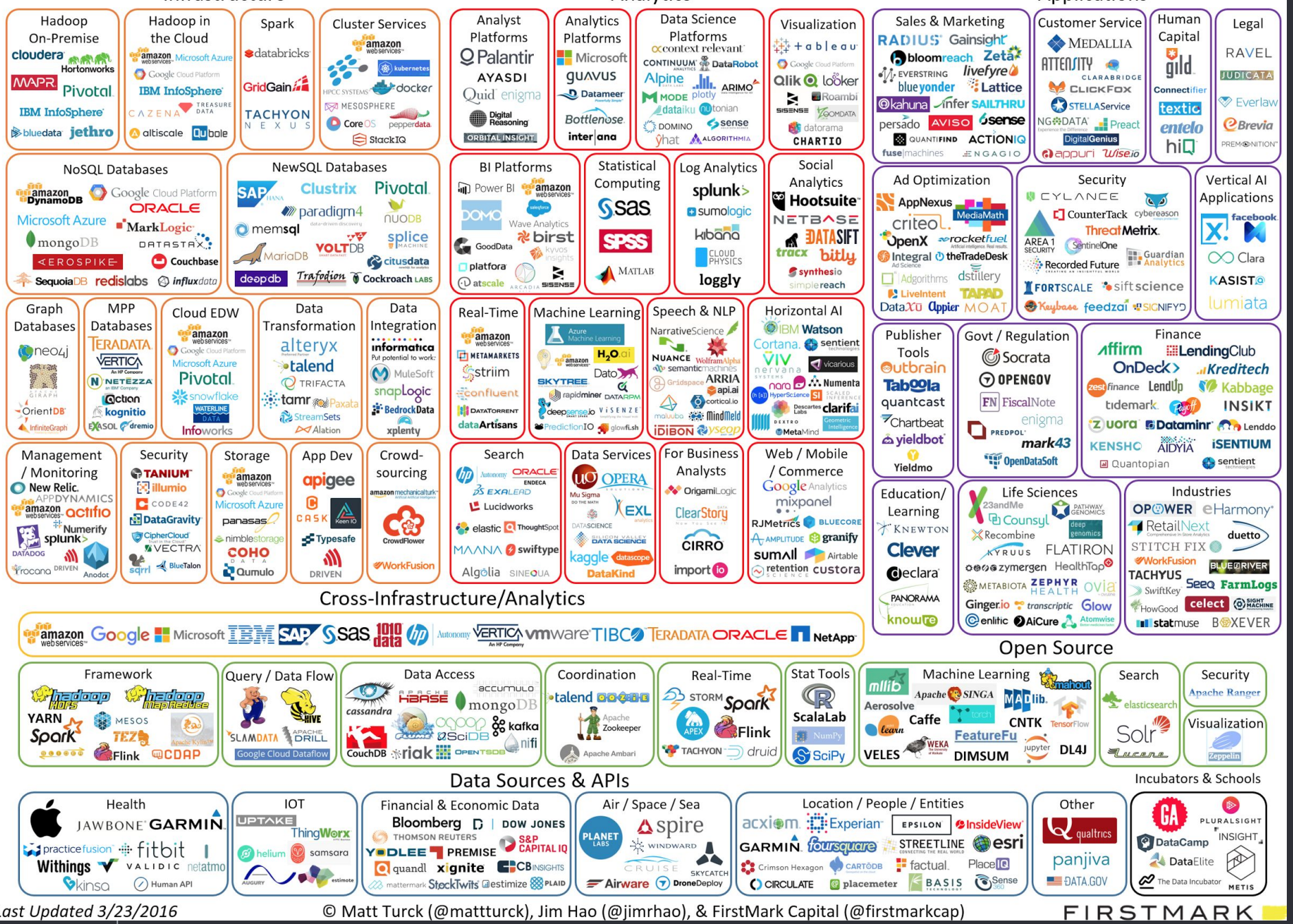
http://github.com/RickBahague

https://medium.com/@rbahaguejr

PySpark/Spark: Python for Big Data Processing

- Our topics
  - Big Data & Data Science & Python
  - Collaboration enabled by Spark/PySpark
  - Finding Pandas in Pyspark
  - Bonus: Scikit-learn on PySpark
  - Questions

Big Data Landscape 2016 (Version 3.0)



FIRSTMARK

Distributed File System

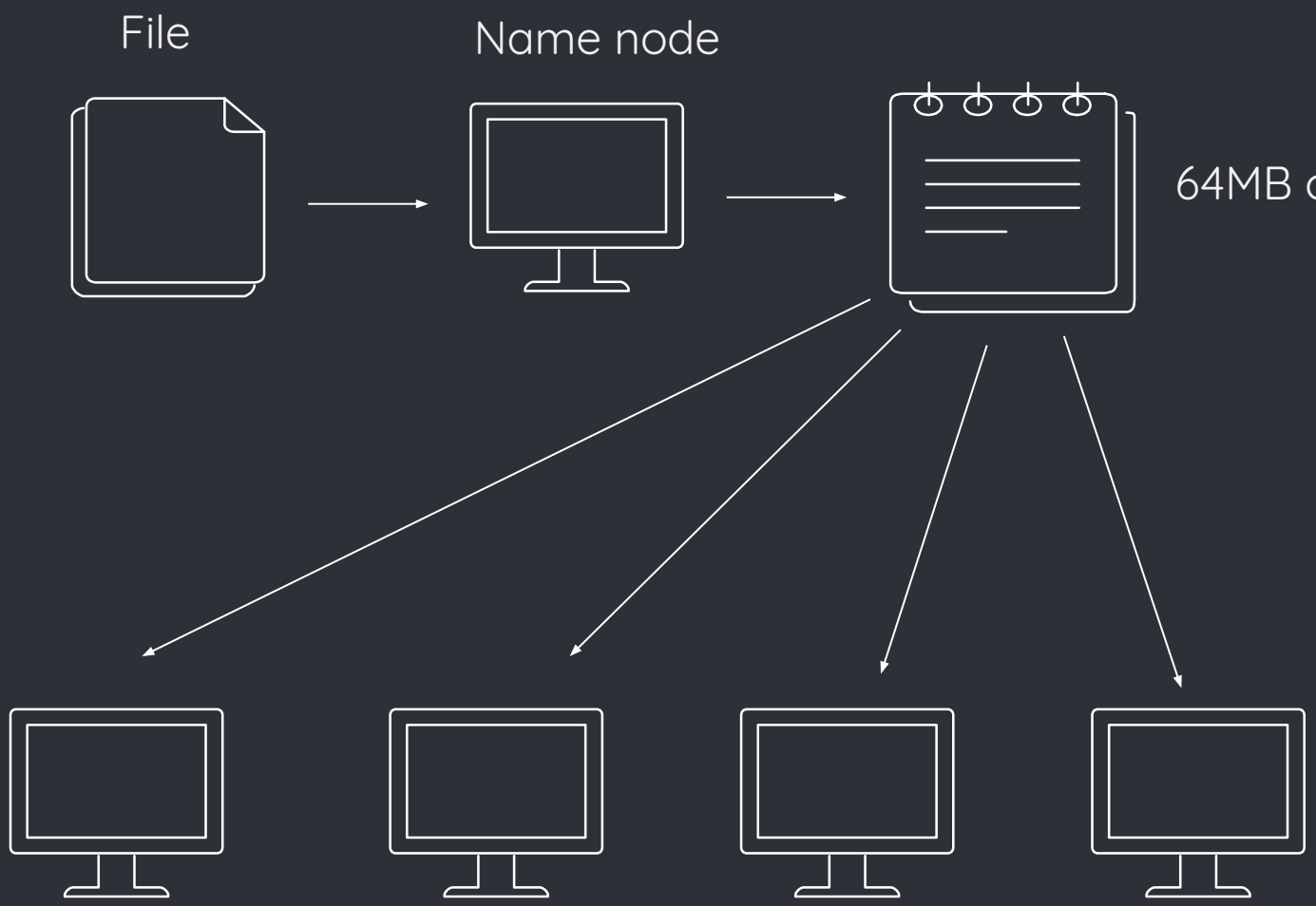
Map-Reduce

Lacks abstraction for leveraging distributed memory

Inefficient for reuse of intermediate results

https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

Hadoop Distributed File System



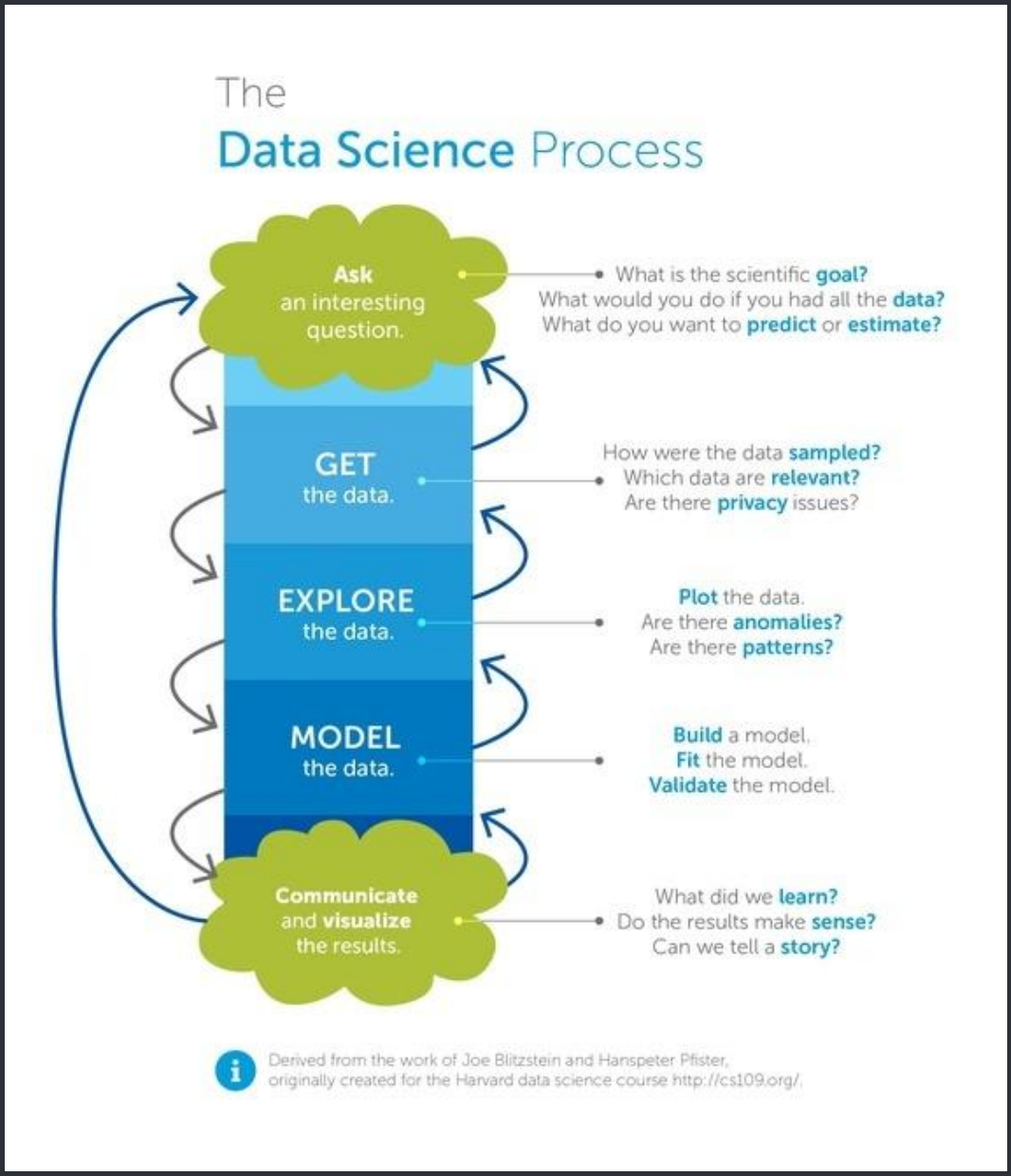
Chunks triplicated and scattered to worker nodes

- Data Science

Statistical models

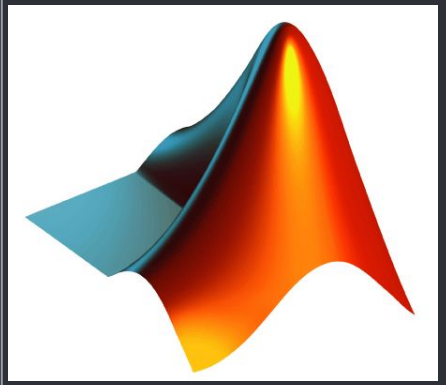
Machine Learning

Visualization



Data Science Backgrounds

Natural Sciences



Computer Science

Statistics

And its ~ 8,000 packages related to Statistics and others

Big data processing and Data Science on this domain is a messy team effort of Data Scientists and DevOps/Data Engineers

“

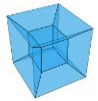
“Big Data needs Data Science but Data Science doesn’t need Big Data”

Carla Gentry aka
@data_nerd

HONOR THY DATA SCIENTIST AND THY ENGINEER

http://partiallyderivative.com/resources/2016/2/17/5-commandments-improving-data-scientist-and-engineer-relations

PyData's Blaze Ecosystem



The Blaze Ecosystem

The Blaze ecosystem is a set of libraries that help users store, describe, query and process data. It is composed of the following core projects:

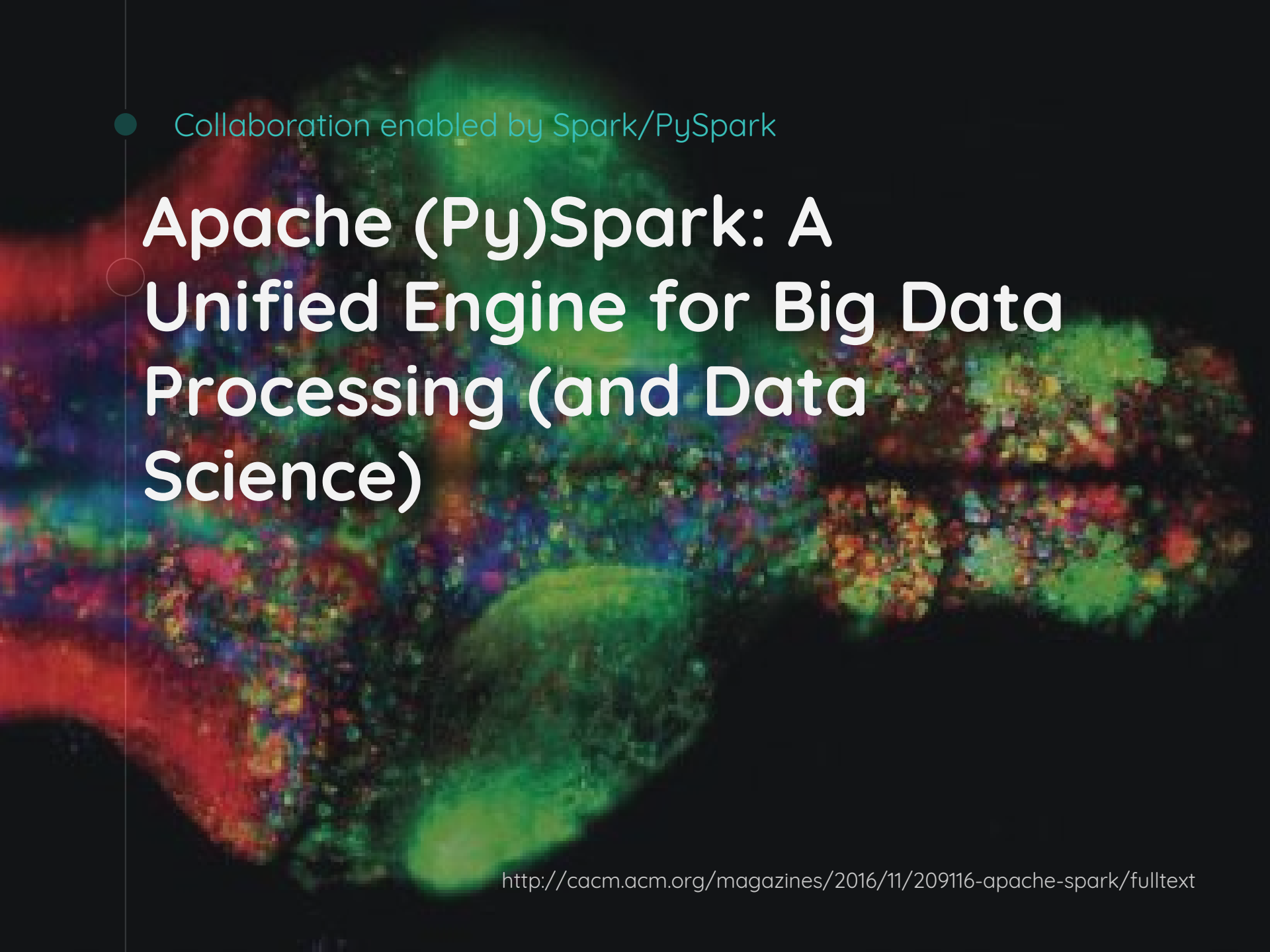
- Blaze: An interface to query data on different storage systems
- Dask: Parallel computing through task scheduling and blocked algorithms
- Datashape: A data description language
- DyND: A C++ library for dynamic, multidimensional arrays
- Libndtypes: A C/C++ library for a low-level version of Datashape
- Ndtypes-python: Python bindings for libndtypes
- Odo: Data migration between different storage systems

PyData's Blaze Ecosystem

Blaze is a query system that looks like NumPy/Pandas. You write Blaze queries, Blaze translates those queries to something else (like SQL), and ships those queries to various database to run on other people's fast code. It smoothes out this process to make interacting with foreign data as accessible as using Pandas. This is actually quite difficult.

Blaze increases human accessibility, not computational performance.

http://blaze.readthedocs.io/en/latest/overview.html



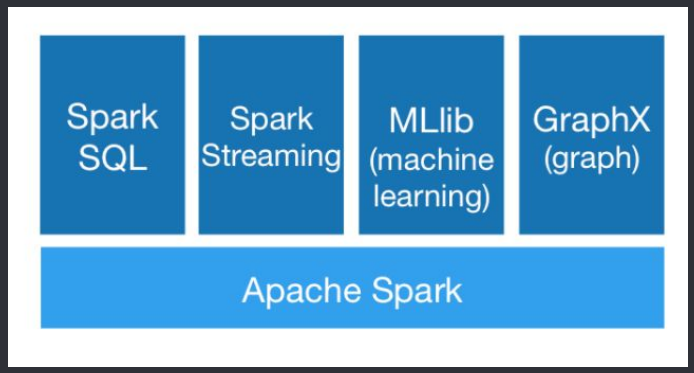
Collaboration enabled by Spark/PySpark

Apache (Py)Spark: A Unified Engine for Big Data Processing (and Data Science)

http://cacm.acm.org/magazines/2016/11/209116-apache-spark/fulltext

Collaboration enabled by Spark/PySpark

“unified engine for distributed processing”

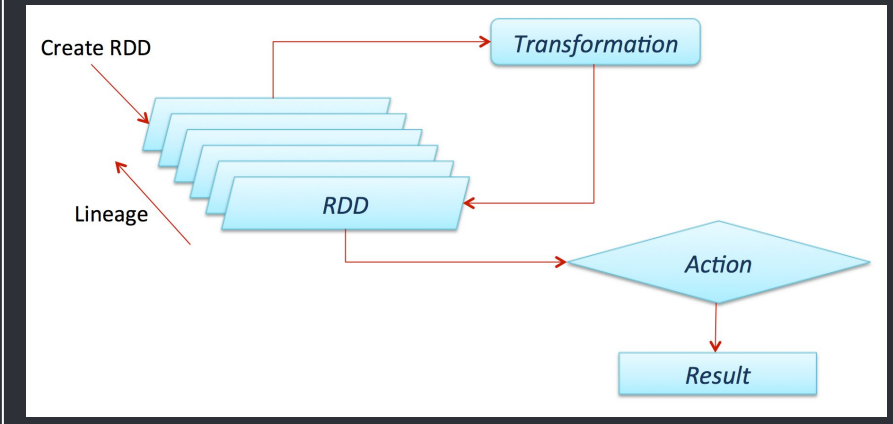


Collaboration enabled by Spark/PySpark

RDD:

Resilient distributed datasets (key programming abstraction)

fault-tolerant collections of objects partitioned across a cluster that can be manipulated in parallel



https://dzone.com/refcardz/apache-spark

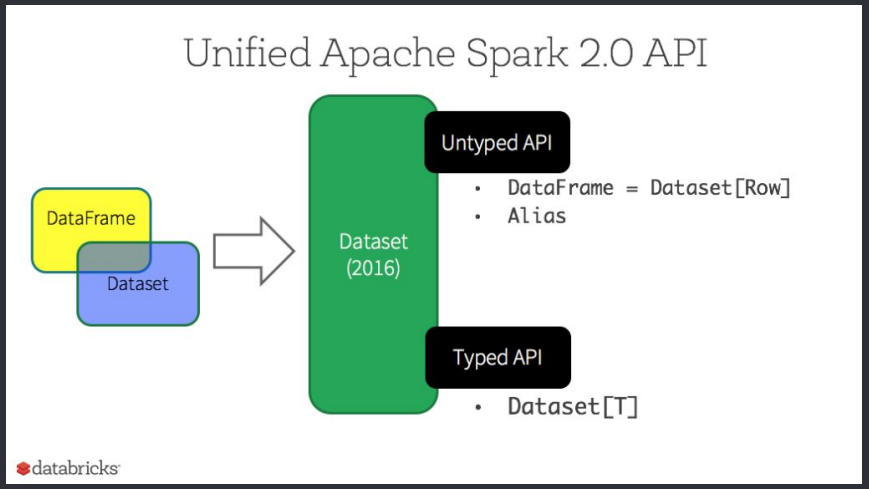
Collaboration enabled by Spark/PySpark

Dataframe:

Immutable distributed collection of data

organized into columns, like a table in RDBMs

HAPPIER DATA SCIENTISTS



https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html

Big Data Processing & Data Science Environment in Python

rdd

py4j

dataframe

PySpark

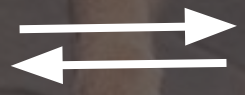
Python API for Apache Spark

Behind Spark/PySpark

Spark Context

File/Collection

Transformations



RDD

Actions

Lazy execution

Behind Spark/PySpark

Transformations

map

filter

flatMap

union

distinct

Actions

count

show

collect

RDD & Python File Read

```
pokemon_rdd = sc.textFile(path+"pokemon.csv")
line_lenghts_from_rdd = pokemon_rdd.map(lambda Row: len(Row)).collect()
```

```
with open(path+"pokemon.csv") as f:
    lines = f.readlines()

line_lengths = list()
for line in lines:
    line_lengths.append(len(line.strip("\n")))
```

Finding Pandas in PySpark

PySpark for the impatient

Finding Pandas in PySpark

PySpark provides data analysts/scientists/researchers a familiar python environment for Big Data processing

Pandas-like behaviour

Comparison of PySpark and Pandas Methods

Finding Pandas in PySpark

Pandas DataFrames

```
In [9]: pandas_df = pd.read_csv(path+"pokemon.csv",sep=",")
        pandas_df.head(5)
```

Out[9]:

	id	identifier	species_id	height	weight	base_experience	order	is_default
0	1	bulbasaur	1	7	69	64	1	1
1	2	ivysaur	2	10	130	142	2	1
2	3	venusaur	3	20	1000	236	3	1
3	4	charmander	4	6	85	62	5	1
4	5	charmeleon	5	11	190	142	6	1

Spark DataFrames

```
In [6]: sparkDF = sqlContext.createDataFrame(pandas_df)
```

```
In [10]: sparkDF.show(2)
```

```
+---+----------+----------+------+------+---------------+-----+----------+
| id|identifier|species_id|height|weight|base_experience|order|is_default|
+---+----------+----------+------+------+---------------+-----+----------+
|  1| bulbasaur|         1|     7|    69|             64|    1|         1|
|  2|   ivysaur|         2|    10|   130|            142|    2|         1|
+---+----------+----------+------+------+---------------+-----+----------+
only showing top 2 rows
```

Finding Pandas in PySpark

Pandas Count

```
In [40]: pandas_df.count()

Out[40]: id                 811
         identifier         811
         species_id         811
         height             811
         weight             811
         base_experience    811
         order              811
         is_default         811
         dtype: int64
```

Spark Count

```
In [39]: sparkDF.count()

Out[39]: 811
```

Finding Pandas in PySpark

Pandas Columns

```
In [42]: pandas_df.columns

Out[42]: Index(['id', 'identifier', 'species_id', 'height', 'weight', 'base_experience',
                'order', 'is_default'],
               dtype='object')
```

PySpark Columns

```
In [41]: sparkDF.columns

Out[41]: ['id',
          'identifier',
          'species_id',
          'height',
          'weight',
          'base_experience',
          'order',
          'is_default']
```

Finding Pandas in PySpark

Pandas dtypes

```
In [43]: pandas_df.dtypes

Out[43]: id                 int64
         identifier        object
         species_id         int64
         height             int64
         weight             int64
         base_experience    int64
         order              int64
         is_default         int64
         dtype: object
```

PySpark dtypes

```
In [44]: sparkDF.dtypes

Out[44]: [('id', 'bigint'),
          ('identifier', 'string'),
          ('species_id', 'bigint'),
          ('height', 'bigint'),
          ('weight', 'bigint'),
          ('base_experience', 'bigint'),
          ('order', 'bigint'),
          ('is_default', 'bigint')]
```

Finding Pandas in PySpark

PySpark describe()

```
sparkDF.select("height","weight").describe().show()
```

```
+-------+------------------+------------------+
|summary|            height|            weight|
+-------+------------------+------------------+
|  count|               811|               811|
|   mean|12.516646115906289| 636.0961775585697|
| stddev|12.266612584292778|1077.6587096303547|
|    min|                 1|                 1|
|    max|               145|              9997|
+-------+------------------+------------------+
```

Pandas describe()

```
pandas_df[['height','weight']].describe()
```

	height	weight
count	811.000000	811.000000
mean	12.516646	636.096178
std	12.266613	1077.658710
min	1.000000	1.000000
25%	6.000000	94.500000
50%	10.000000	295.000000
75%	15.000000	650.000000
max	145.000000	9997.000000

Finding Pandas in PySpark

Pandas selection

```
pandas_df[pandas_df.height> 100]
```

	id	identifier	species_id	height	weight	base_experience	order	is_default
320	321	wailord	321	145	3980	175	387	1
792	10072	steelix-mega	208	105	7400	214	120	0
799	10079	rayquaza-mega	384	108	3920	351	467	0

PySpark selection

```
sparkDF.filter("height>100").show()
```

```
+-----+------------+---------+------+------+---------------+-----+----------+
|   id|  identifier|species_id|height|weight|base_experience|order|is_default|
+-----+------------+---------+------+------+---------------+-----+----------+
|  321|     wailord|      321|   145|  3980|            175|  387|         1|
|10072|steelix-mega|      208|   105|  7400|            214|  120|         0|
|10079|rayquaza-mega|      384|   108|  3920|            351|  467|         0|
+-----+------------+---------+------+------+---------------+-----+----------+
```

Finding Pandas in PySpark

Pandas finding uniques, distincts

```
len(pd.unique(pandas_df['species_id']))
```

```
721
```

PySpark finding uniques, distincts

```
sparkDF.select("species_id").distinct().count()
```

```
721
```

Finding Pandas in PySpark

Pandas group by, sort

```
pandas_df[['species_id','id']].groupby(['species_id']).agg({"id":"count"}).\
    sort_values(by='id',ascending=False).head(2)
```

	id
species_id	
25	7
479	6

PySpark group by, sort

```
sparkDF.groupBy("species_id").agg(count("id").alias("counts")).\
    sort("counts",ascending=False).show(2)
```

```
+----------+------+
|species_id|counts|
+----------+------+
|        25|     7|
|       479|     6|
+----------+------+
only showing top 2 rows
```

Finding Pandas in PySpark

Pandas join

```
pandas_df.merge(joiner_df,left_index=['species_id'],right_index=['id'],how='left').head(2)
```

	id_x	identifier_x	species_id	height	weight	base_experience	order_x	is_default	id_y	identifier_y	...	gender_rate	capture_rate	base_happine
0	1	bulbasaur	1	7	69	64	1	1	1.0	bulbasaur	...	1.0	45.0	70.0
1	2	ivysaur	2	10	130	142	2	1	2.0	ivysaur	...	1.0	45.0	70.0

2 rows × 26 columns

PySpark join

```
sparkDF.join(joiner_sparkDF,sparkDF.species_id==joiner_sparkDF.id,'left').show(2)
```

```
+---+----------+----------+------+------+---------------+-----+----------+---+----------+-------------+--------------
---------+------------------+--------+--------+----------+-----------+------------+--------------+-------+-----------
--+----------------------+--------------+----------------+-----+--------------+
| id|identifier|species_id|height|weight|base_experience|order|is_default| id|identifier|generation_id|evolves_from_s
pecies_id|evolution_chain_id|color_id|shape_id|habitat_id|gender_rate|capture_rate|base_happiness|is_baby|hatch_count
er|has_gender_differences|growth_rate_id|forms_switchable|order|conquest_order|
+---+----------+----------+------+------+---------------+-----+----------+---+----------+-------------+--------------
---------+------------------+--------+--------+----------+-----------+------------+--------------+-------+-----------
--+----------------------+--------------+----------------+-----+--------------+
|406|     budew|       406|     2|    12|             56|  378|         1|406|     budew|            4|
NaN|               158|       5|      12|       NaN|          4|         255|            70|      1|           20|
0|             4|               0|  341|           NaN|
|203| girafarig|       203|    15|   415|            159|  258|         1|203| girafarig|            2|
NaN|               101|      10|       8|       3.0|          4|          60|            70|      0|           20|
1|             2|               0|  234|           NaN|
+---+----------+----------+------+------+---------------+-----+----------+---+----------+-------------+--------------
---------+------------------+--------+--------+----------+-----------+------------+--------------+-------+-----------
--+----------------------+--------------+----------------+-----+--------------+
only showing top 2 rows
```

SCIKIT-LEARN & RDDs

A distributed learning using scikit-learn on PySpark

```
site_names_pdf = pd.read_csv(path+"location_super.csv",sep="|")
```

```
site_names_pdf.dtypes
```

```
district        int64
latitude      float64
longitude     float64
dtype: object
```

Problem: Cluster nearby locations (lat,long) with 700m apart

sklearn.cluster.DBSCAN

class sklearn.cluster. **DBSCAN** (*eps=0.5*, *min_samples=5*, *metric='euclidean'*, *algorithm='auto'*, *leaf_size=30*, *p=None*, *n_jobs=1*) [source]

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

Parameters:

eps : float, optional

The maximum distance between two samples for them to be considered as in the same neighborhood.

min_samples : int, optional

The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.

metric : string, or callable

The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by metrics.pairwise.calculate_distance for its metric parameter. If metric is “precomputed”, X is assumed to be a distance matrix and must be square. X may be a sparse matrix, in which case only “nonzero” elements may be considered neighbors for DBSCAN.

Load locations as RDD of lines

```
sites_names_rdd = sc.textFile("./location_super.csv")
```

Split lines to RDD of lists

```
sites_names_rdd.map(split_row).take(5)
```

```
[[u'district', u'latitude', u'longitude'],
 [u'3', u'38.55042047', u'-121.3914158'],
 [u'5', u'38.47350069', u'-121.4901858'],
 [u'2', u'38.65784584', u'-121.4621009'],
 [u'6', u'38.50677377', u'-121.4269508']]
```

Filter Column name

```
sites_names_rdd.map(split_row).filter(lambda Row: Row[0]!='district').take(5)
```

```
[[u'3', u'38.55042047', u'-121.3914158'],
 [u'5', u'38.47350069', u'-121.4901858'],
 [u'2', u'38.65784584', u'-121.4621009'],
 [u'6', u'38.50677377', u'-121.4269508'],
 [u'2', u'38.6374478', u'-121.3846125']]
```

Create (key,value) pair

```
sites_names_rdd.map(split_row).filter(lambda Row: Row[0]!='district').\
    map(lambda Row: (Row[0],Row[1:])).take(5)
```

```
[(u'3', [u'38.55042047', u'-121.3914158']),
 (u'5', [u'38.47350069', u'-121.4901858']),
 (u'2', [u'38.65784584', u'-121.4621009']),
 (u'6', [u'38.50677377', u'-121.4269508']),
 (u'2', [u'38.6374478', u'-121.3846125'])]
```

Group similar keys

```
sites_names_rdd.map(split_row).filter(lambda Row: Row[0]!='district').\
    map(lambda Row: (Row[0],Row[1:])).groupByKey().take(5)
```

```
[(u'1', <pyspark.resultiterable.ResultIterable at 0x119d00a50>),
 (u'3', <pyspark.resultiterable.ResultIterable at 0x119cf8950>),
 (u'5', <pyspark.resultiterable.ResultIterable at 0x119cf8b10>),
 (u'2', <pyspark.resultiterable.ResultIterable at 0x1199d0850>),
 (u'4', <pyspark.resultiterable.ResultIterable at 0x11a3e6890>)]
```

Prepare for DB Scan (create dictionaries from list of list)

```
def prepare_for_dbscan(x):
    xx = list(x[1])
    list_coord_dict = list()
    for xxx in xx:
        coord_dict = dict()
        coord_dict['latitude'] = np.float(xxx[0].encode("utf-8"))
        coord_dict['longitude'] = np.float(xxx[1].encode("utf-8"))
        list_coord_dict.append(coord_dict)
    return x[0], list_coord_dict

sites_names_rdd.map(split_row).filter(lambda Row: Row[0]!='district').\
    map(lambda Row: (Row[0],Row[1:])).groupByKey().map(prepare_for_dbscan).take(1)
```

```
[(u'1',
  [{'latitude': 38.60960217, 'longitude': -121.4918375},
   {'latitude': 38.65994218, 'longitude': -121.5259008},
   {'latitude': 38.60893745, 'longitude': -121.5187927},
   {'latitude': 38.64378844, 'longitude': -121.5341593},
   {'latitude': 38.62798948, 'longitude': -121.4857344},
   {'latitude': 38.61153542, 'longitude': -121.5370613},
   {'latitude': 38.63036289, 'longitude': -121.4755269},
   {'latitude': 38.62453557, 'longitude': -121.5264198},
   {'latitude': 38.62504428, 'longitude': -121.4978986},
   {'latitude': 38.61488957, 'longitude': -121.4934272},
   {'latitude': 38.63207451, 'longitude': -121.5287747},
```

Prepare for DB Scan (create pandas dataframe and distance matrix)

```
def create_pandas_df(x):
    df = pd.DataFrame(x[1])
    return x[0],df

def pandas_df_to_matrix(x):
    return x[0], x[1].as_matrix(columns=['latitude', 'longitude'])

sites_names_rdd.map(split_row).filter(lambda Row: Row[0]!='district').\
    map(lambda Row: (Row[0],Row[1:])).groupByKey().map(prepare_for_dbscan).map(create_pandas_df).\
    map(pandas_df_to_matrix).take(2)
```

```
[(u'1', array([[  38.60960217, -121.4918375 ],
         [  38.65994218, -121.5259008 ],
         [  38.60893745, -121.5187927 ],
         ...,
         [  38.62201506, -121.4700579 ],
         [  38.60799414, -121.4700221 ],
         [  38.61300454, -121.4918727 ]])),
 (u'3', array([[  38.55042047, -121.3914158 ],
         [  38.56433456, -121.4618826 ],
         [  38.55611545, -121.4142729 ],
         ...,
         [  38.55790107, -121.4106352 ],
         [  38.57783198, -121.4704595 ],
         [  38.57203045, -121.4670118 ]]))]
```

Get Municipal PSGC, Number of Clusters, Actual Geo-coordinates

```
def create_clusters_via_dbscan(x):
    kms_per_radian = 6371.0088
    epsilon = .7/ kms_per_radian
    coords = x[1]
    db = DBSCAN(eps=epsilon, min_samples=1, algorithm='ball_tree', metric='haversine').fit(np.radians(coords))
    cluster_labels = db.labels_
    num_clusters = len(set(cluster_labels))
    clusters = pd.Series([coords[cluster_labels == n] for n in range(num_clusters)])
    return x[0], num_clusters, len(coords)

sites_names_rdd.map(split_row).filter(lambda Row: Row[0]!='district').\
    map(lambda Row: (Row[0],Row[1:])).groupByKey().map(prepare_for_dbscan).map(create_pandas_df).\
    map(pandas_df_to_matrix).map(create_clusters_via_dbscan).take(10)
```

```
[(u'1', 4, 868),
 (u'3', 1, 1575),
 (u'5', 1, 1159),
 (u'2', 5, 1462),
 (u'4', 1, 1161),
 (u'6', 1, 1359)]
```

DBScan approach from: http://geoffboeing.com/2014/08/clustering-to-reduce-spatial-data-set-size/, retrieved 25 Feb 2016

Thanks!

ANY QUESTIONS?

You can find me at

@rbahaguejr

https://medium.com/@rbahaguejr