Big Data and PySpark

Data Science Immersive

FLATIRON SCHOOL

Agenda Today

- Big Data overview
- Introduction to Distributed System with MapReduce and Hadoop Ecosystem
- Introduction to Spark and PySpark
  - Understanding Spark Context
  - Resilient Distributed Datasets(RDD) in Spark
    - What is RDD?
    - RDD transformation and Actions
  - Spark use cases

After class, you will be able to...

- Understand and explain distributed systems and their applications
- Understand use cases and motivation for MapReduce, as well as how it works
- Explain the advantages and disadvantages of MapReduce and PySpark
- Launch PySpark in Docker, open a sparkcontext, and work with basic RDD operations

But you WONT be able to...

- Write and produce MapReduce jobs in Java
- Become proficient in end-to-end ETL with data in the hadoop ecosystem
- Working with AWS instances and productionizing your ml models made with pyspark

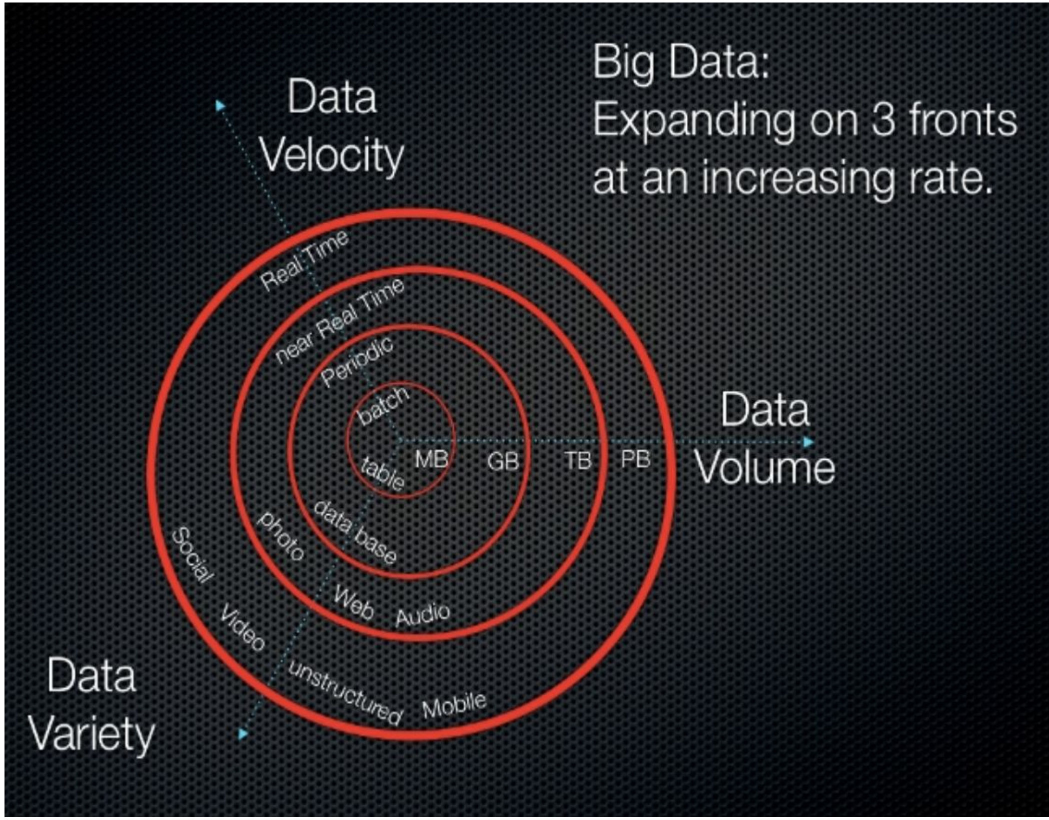
The framework evolving MapReduce, Docker, PySpark, and distributed systems in general are highly technical and specialized, so we will only learn a high level overview of it

What is Big Data?

- What comes to your mind when thinking of the word **big data**?
- How is it different from the projects you have worked on?

What is Big Data?

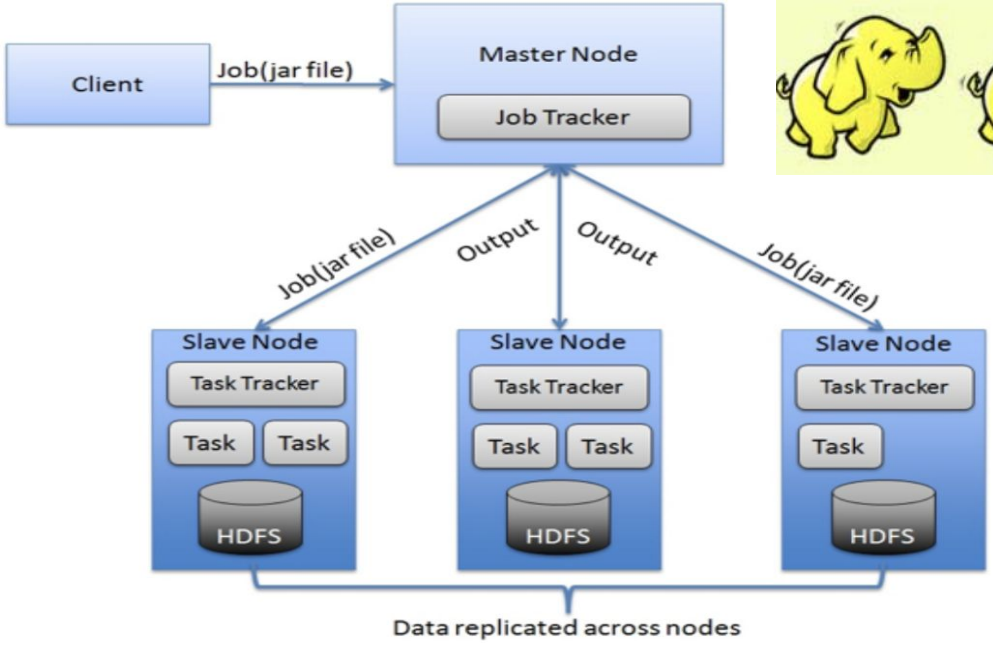
- Three V's of big data
  - Volume
  - Velocity
  - Variety

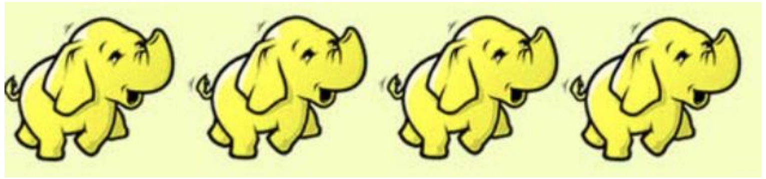


Why do we need distributed systems?...

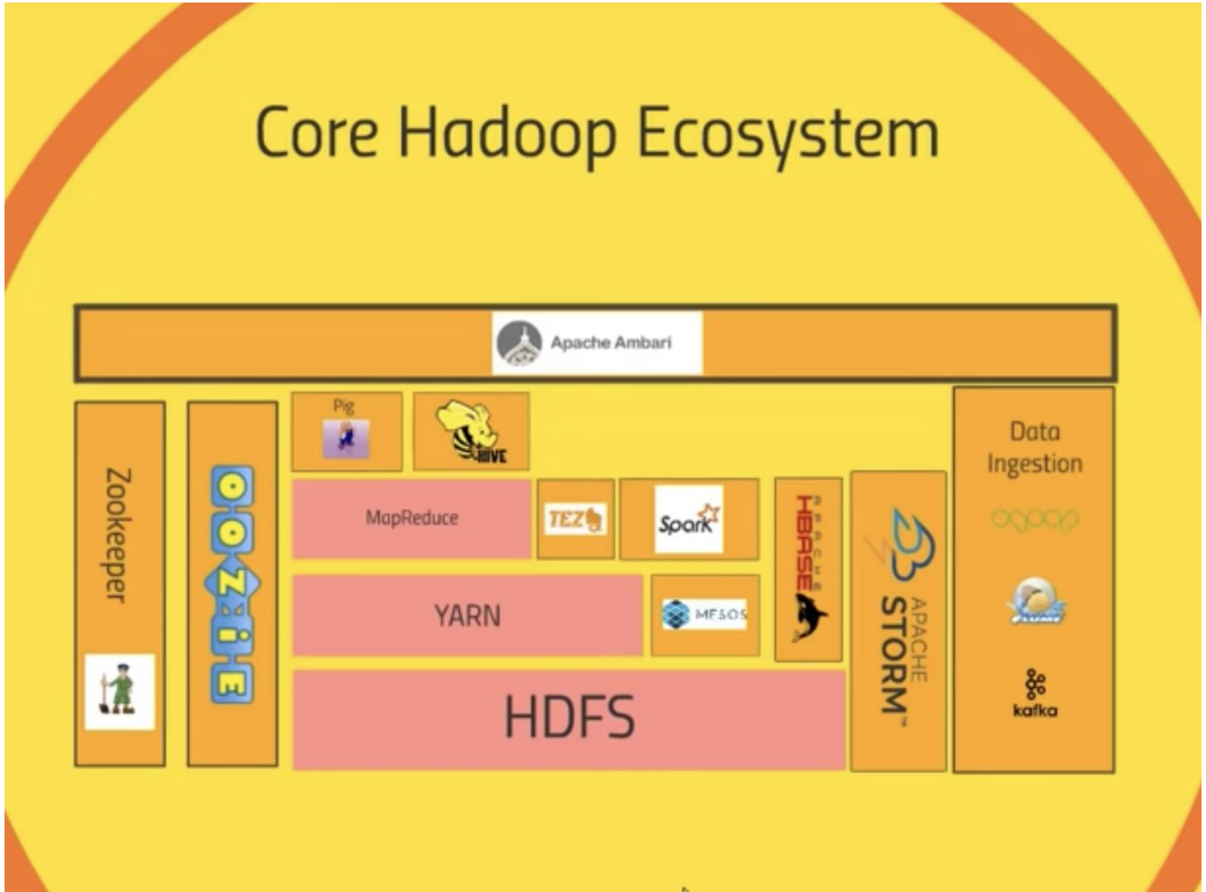
- How long would it take for 1TB of data to be read on a hard drive?
- 1 TB = 1024 GB = 1048576 MB
- 1048576 / 100 M/s = 2.91 hours
- But with 100 M/s across 10 systems, 1 TB/ 1000 MB/s = 17 min
- 1 TB/ 10,000 MB/s = 105 seconds

Distributed system - Hadoop

- A distributed computing system and environment for very large datasets on computer clusters.
- 



Hadoop Ecosystem



Divide & Conquer

- Instead of using just one system to perform such task, Hadoop and MapReduce allows tasks to be divided:
  - Split tasks into many subtasks
  - Solve these tasks individually
  - Recombine the subtasks into a final result

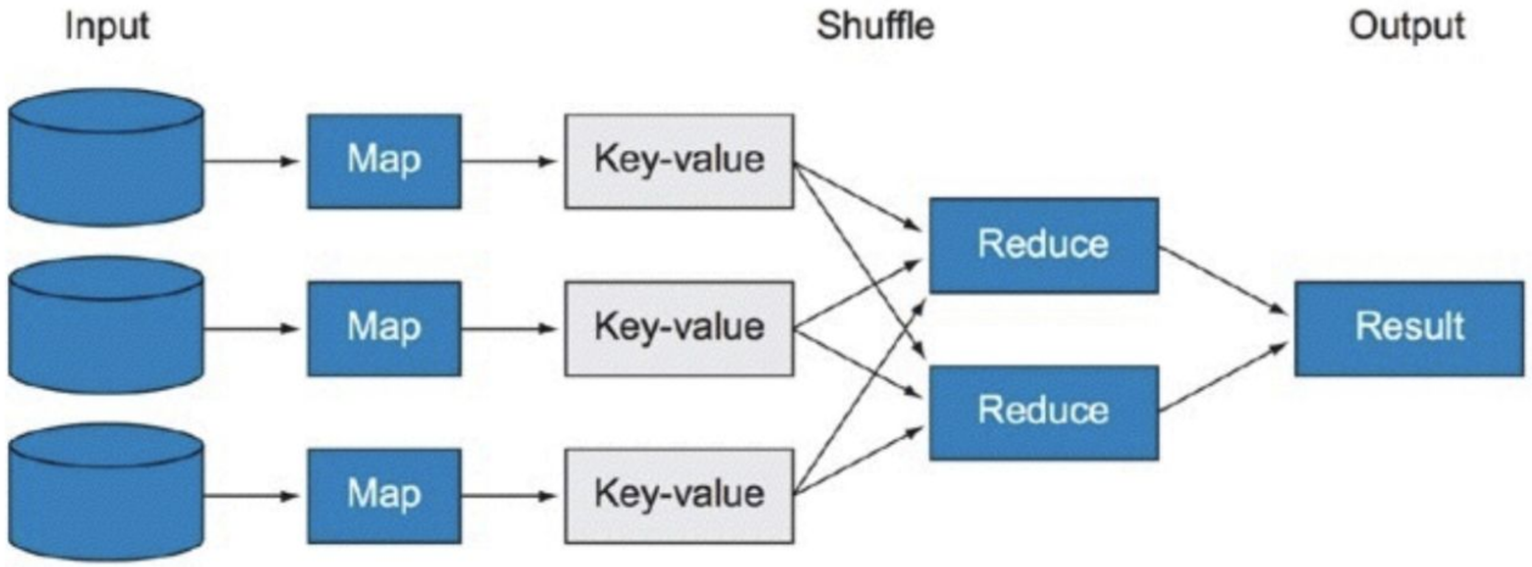
MapReduce

- MapReduce splits tasks into subtasks and allow them to be processed in parallel
- This happens in two phases:
    - The mapper phase - transforms your data in parallel across all nodes in your computing cluster in an efficient manner
    - The reducer phase - aggregate your entire data and return a final result

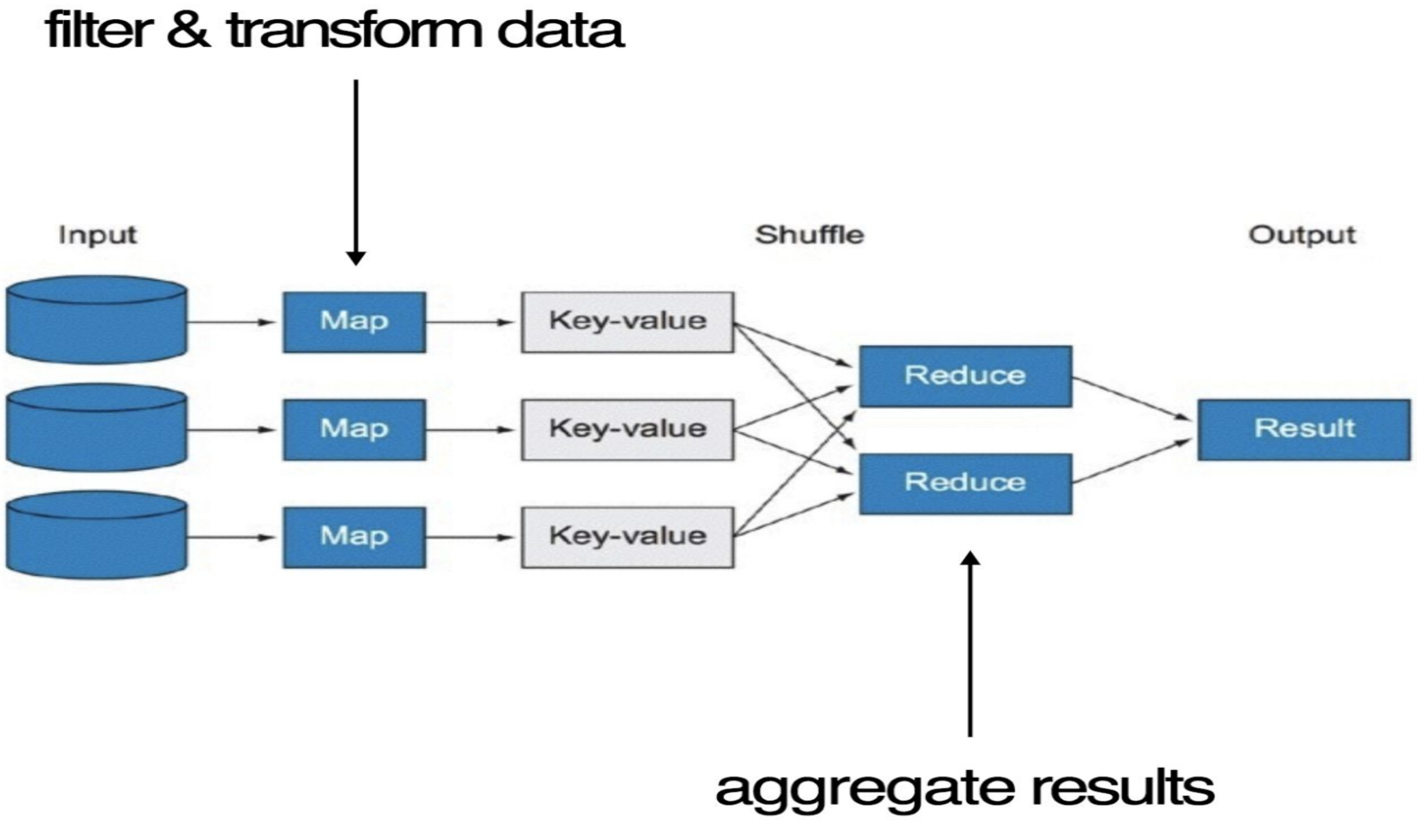
```
class Mapper
   method Map(docid id, doc d)
      for all term t in doc d do
         Emit(term t, count 1)

class Reducer
   method Reduce(term t, counts [c1, c2,...])
      sum = 0
      for all count c in [c1, c2,...] do
          sum = sum + c
      Emit(term t, count sum)
```

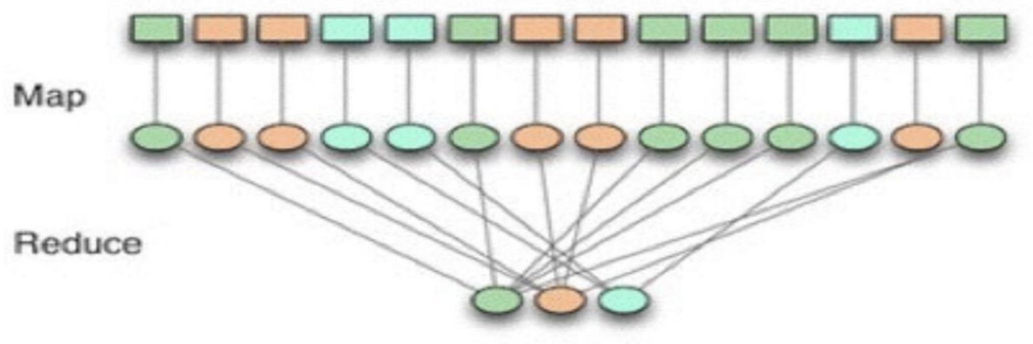
MapReduce

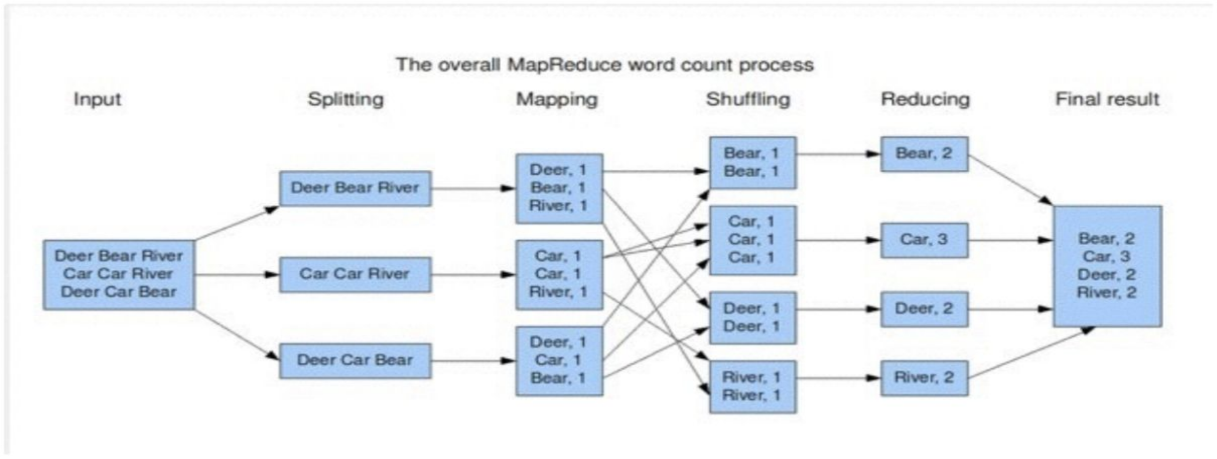

Input
Shuffle
Output
Map
Map
Map
Key-value
Key-value
Key-value
Reduce
Reduce
Result

MapReduce


filter & transform data
Input
Shuffle
Output
Map
Map
Map
Key-value
Key-value
Key-value
Reduce
Reduce
Result
aggregate results

Example: Word Count with MapReduce





Other MapReduce Examples?

What machine learning or data science related tasks can be efficiently solved using the MapReduce framework?

What tasks will be difficult for MapReduce framework?

Beyond Hadoop

- Even though Hadoop is quite wonderful and speeds up processing tremendously, there are a few problems...
    - Batch processing for iterative algorithm
    - Hadoop jobs are mostly written in Java, which can be painful and error-prone without expertise--therefore making it less accessible

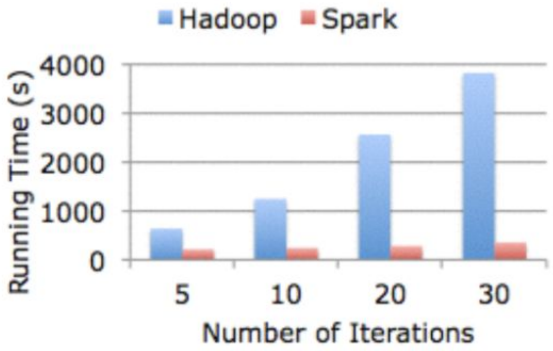
Beyond MapReduce

Spark

Lightning-Fast Cluster Computing

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

file.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
```



Spark

- Advantages of Spark
  - Keeps data for in-memory querying
  - More fault-tolerant
  - Allows for higher-level query language to implement sql-like semantics on top of MapReduce
  - Useful for iterative algorithm
  - Some Spark jobs can be up to 100x faster than traditional MapReduce

PySpark

Overview of PySpark

- Apache Spark is written in Scala. Spark is a general purpose data-processing software that offers high level API's written in Scala, Java, R, and Python.
- To support Python with Spark, Apache Spark Community released PySpark
- Similar computation speed and power as Scala
- PySpark APIs are similar to Pandas and Scikit-learn

Understanding SparkContext

Understanding SparkContext

- SparkContext is an entry point into the world of Spark
- An entry point is a way of connecting to Spark cluster
- An entry point is like a key to the house
- PySpark has a default SparkContext called `sc`

Spark RDDs

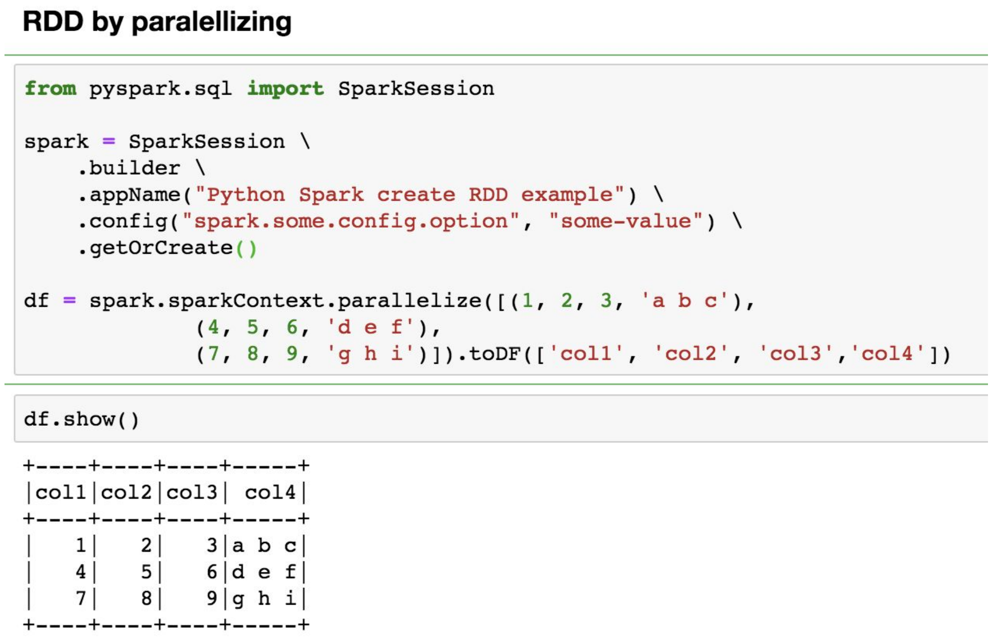
- Resilient Distributed Datasets
  - Resilient Distributed Datasets (RDD) are fundamental data structures of Spark. An RDD is, essentially, the Spark representation of a set of data, spread across multiple machines, with APIs to let you act on it. An RDD could come from any datasource, e.g. text files, a database, a JSON file etc.

Spark RDDs

- Features of RDDs:
  - Resilient: Ability to withstand failures
  - Distributed: Spanning across multiple machines
  - Datasets: Collection of partitioned data e.g, Arrays, Tables, Tuples etc

Spark RDDs

- How to create and work with an RDD?
  a. Parallelize an existing collection of objects



RDD by paralellizing

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.sparkContext.parallelize([(1, 2, 3, 'a b c'),
             (4, 5, 6, 'd e f'),
             (7, 8, 9, 'g h i')]).toDF(['col1', 'col2', 'col3','col4'])
```

```
df.show()
```

```
+----+----+----+-----+
|col1|col2|col3| col4|
+----+----+----+-----+
|   1|   2|   3|a b c|
|   4|   5|   6|d e f|
|   7|   8|   9|g h i|
+----+----+----+-----+
```

Spark RDDs

- How to create and work with an RDD?
  a. Parallelize
  b. Work with external data (which is hard to do with Docker)
     - Files in HDFS
     - Objects in Amazon S3 bucket
     - lines in a text file

```
fileRDD = sc.textFile("README.md")
```

```
type(fileRDD)

<class 'pyspark.rdd.PipelinedRDD'>
```

Spark RDDs

- How to create and work with an RDD?
  a. Parallelize
  b. Work with external data
     - Files in HDFS
     - Objects in Amazon S3 bucket
     - lines in a text file
  c. From existing RDD’s

RDD Operations

- Transformations - transform the given RDD and return a new one
  a. Basic RDD transformations:
     - map()
     - flatmap()

```
RDD = sc.parallelize(["hello world", "how are you"])
RDD_flatmap = RDD.flatMap(lambda x: x.split(" "))
```

     - filter()

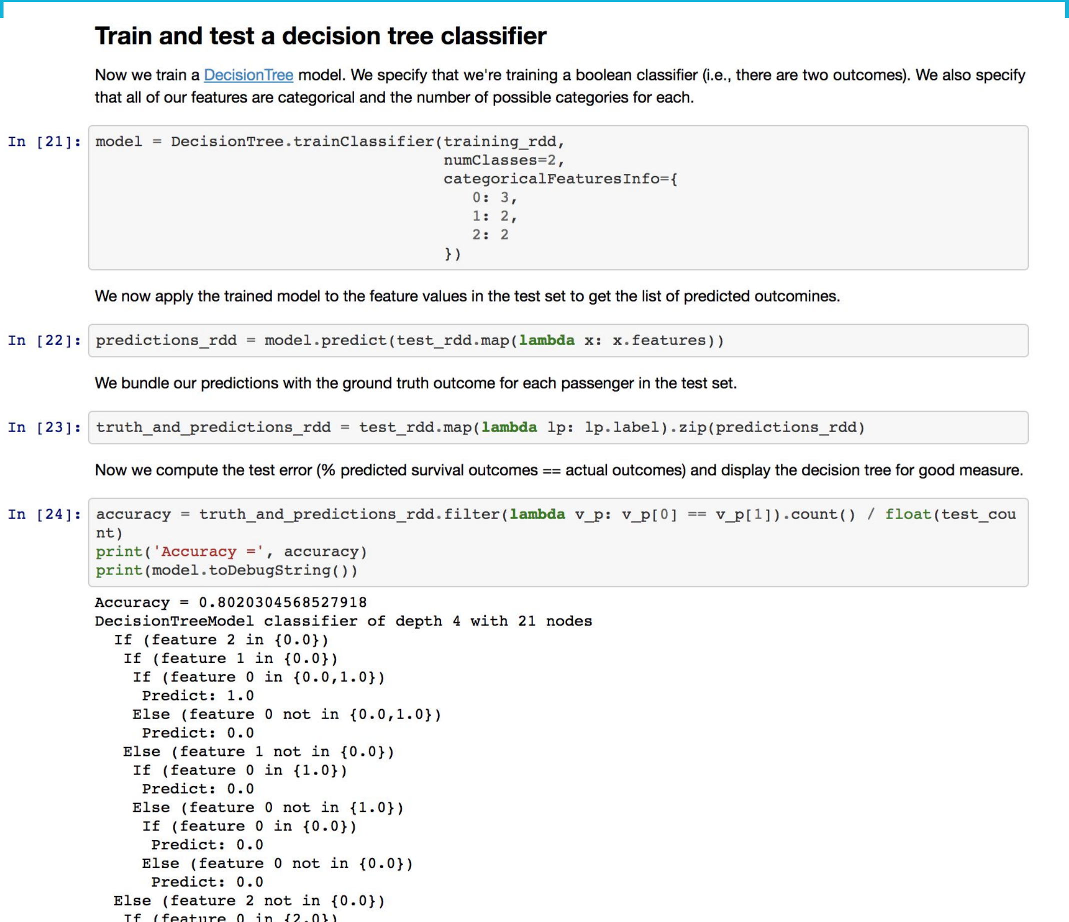
RDD Operations

- Actions - perform operations that return a value after running a computation on the RDD
  a. Basic RDD actions
     - collect() - return all the elements of the dataset as an array
     - take(N) returns an array with the first N elements of the dataset
     - first() - returns first element of RDD
     - count() - returns total number of elements in RDD

Spark Use Cases

- Streaming data and analyze data in real time
  - Data ETL
  - Data enrichment - combining streamed data with static data
- Machine Learning - mllib in pyspark.
  - Performs tasks such as market segmentation and sentiment analysis
  - Recommendation engines
- Interactive Analysis

Spark Use Cases



Train and test a decision tree classifier

Now we train a DecisionTree model. We specify that we're training a boolean classifier (i.e., there are two outcomes). We also specify that all of our features are categorical and the number of possible categories for each.

In [21]:
```
model = DecisionTree.trainClassifier(training_rdd,
                                     numClasses=2,
                                     categoricalFeaturesInfo={
                                        0: 3,
                                        1: 2,
                                        2: 2
                                     })
```

We now apply the trained model to the feature values in the test set to get the list of predicted outcomines.

In [22]:
```
predictions_rdd = model.predict(test_rdd.map(lambda x: x.features))
```

We bundle our predictions with the ground truth outcome for each passenger in the test set.

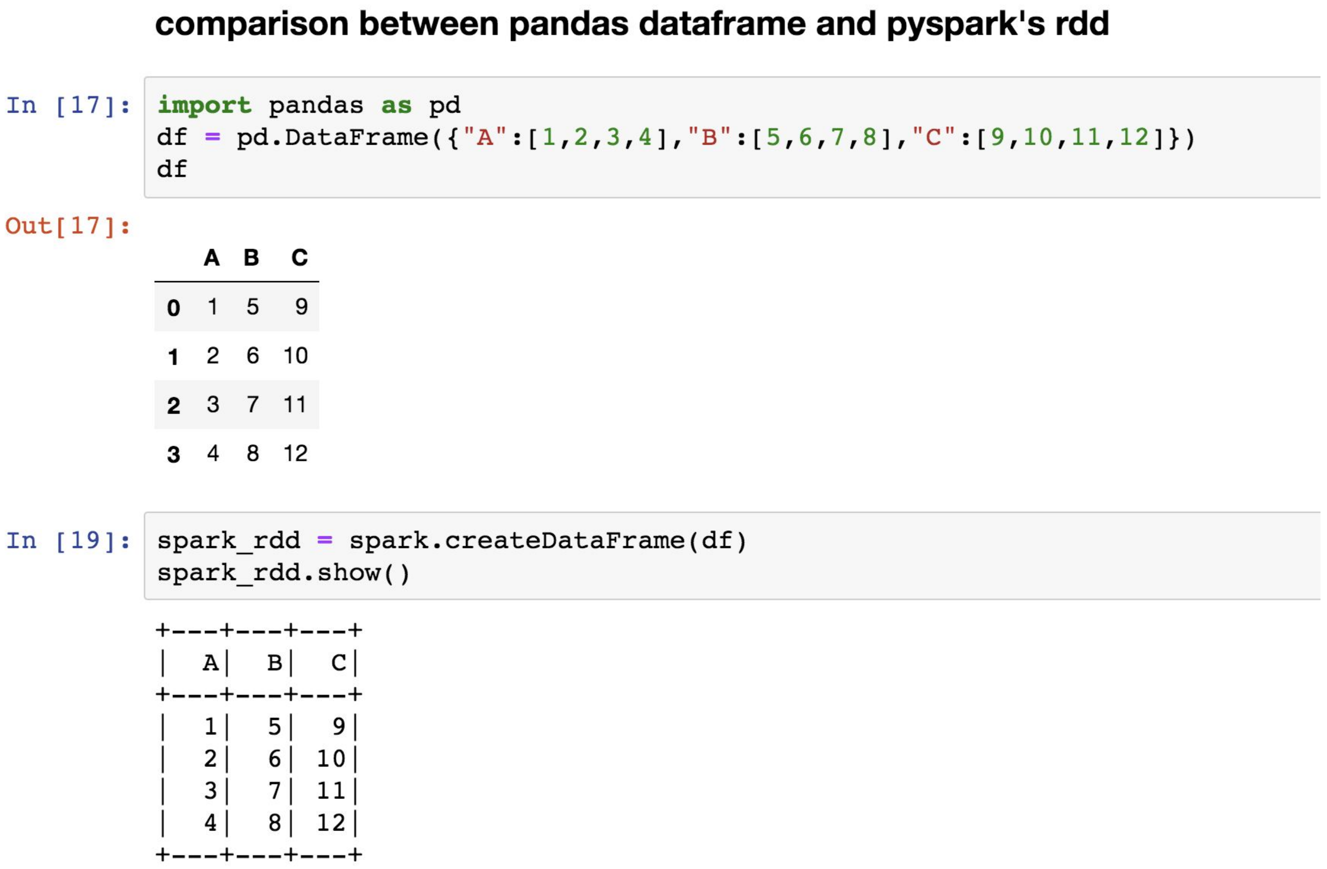
In [23]:
```
truth_and_predictions_rdd = test_rdd.map(lambda lp: lp.label).zip(predictions_rdd)
```

Now we compute the test error (% predicted survival outcomes == actual outcomes) and display the decision tree for good measure.

In [24]:
```
accuracy = truth_and_predictions_rdd.filter(lambda v_p: v_p[0] == v_p[1]).count() / float(test_cou
nt)
print('Accuracy =', accuracy)
print(model.toDebugString())
```

```
Accuracy = 0.8020304568527918
DecisionTreeModel classifier of depth 4 with 21 nodes
  If (feature 2 in {0.0})
   If (feature 1 in {0.0})
    If (feature 0 in {0.0,1.0})
     Predict: 1.0
    Else (feature 0 not in {0.0,1.0})
     Predict: 0.0
   Else (feature 1 not in {0.0})
    If (feature 0 in {1.0})
     Predict: 0.0
    Else (feature 0 not in {1.0})
     If (feature 0 in {0.0})
      Predict: 0.0
     Else (feature 0 not in {0.0})
      Predict: 0.0
  Else (feature 2 not in {0.0})
   If (feature 0 in {2.0})
```

Comparison of Pandas df and Spark RDD



comparison between pandas dataframe and pyspark's rdd

In [17]:
```
import pandas as pd
df = pd.DataFrame({"A":[1,2,3,4],"B":[5,6,7,8],"C":[9,10,11,12]})
df
```

Out[17]:

	A	B	C
0	1	5	9
1	2	6	10
2	3	7	11
3	4	8	12

In [19]:
```
spark_rdd = spark.createDataFrame(df)
spark_rdd.show()
```

```
+---+---+---+
|  A|  B|  C|
+---+---+---+
|  1|  5|  9|
|  2|  6| 10|
|  3|  7| 11|
|  4|  8| 12|
+---+---+---+
```

Resources

https://www.oreilly.com/library/view/data-analytics-with/9781491913734/ch04.html