BIG DATA & DISTRIBUTED PROCESSING

- Big Data Overview
- Scalable Systems
- Hadoop
- Spark
  - History
  - RDDs
  - DataFrames
  - Spark Architecture
  - Spark API
- PySpark Examples
- PySpark Exercise
- Assignment

START SPARK SESSION

```
import pyspark
from pyspark.sql import SparkSession

conf = pyspark.SparkConf().setAll([
         ('spark.master’,   'local[*]'),
         ('spark.app.name', 'PySpark Demo')])

spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

Use * to use all available cores, or integer value to specify number of cores to use

Configuration parameters for Spark session

Get existing Spark session or create new one

LOAD DATA

- Loading data from local file system

```
df = spark.read.text("file:///<path>/<file>.txt")

df = spark.read.csv("file:///<path>/<file>.csv",
                    header=True).cache()
```

- Loading data from HDFS

```
df = spark.read.text("hdfs:///<path>/<file>.txt") \
                     .cache()

df = spark.read.csv("hdfs:///<path>/<file>.csv",
                     header=True,
                     interSchema=True).cache()
```

Indicates whether column headers exist

Automatically infer data types of columns

Cache data in memory

CHAINING

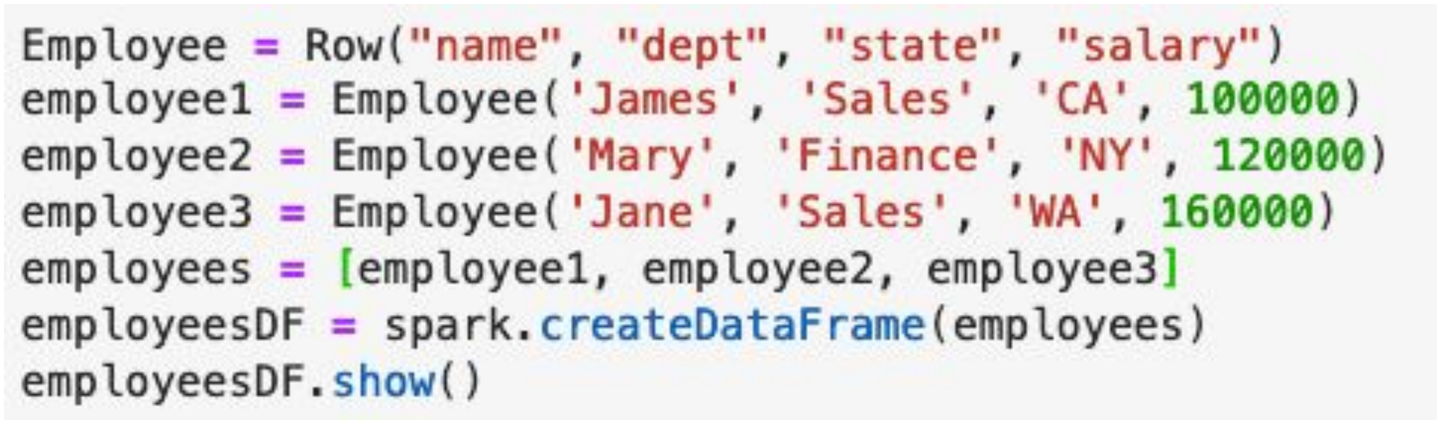
```
text_file = sc.textFile("hdfs://...")

counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)

counts.saveAsTextFile("hdfs://...")
```

Line continuation indicator

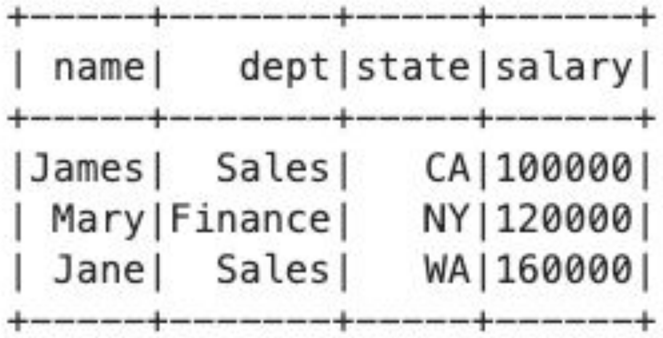
Chaining: Making multiple method calls on same object

RDD Wordcount

CREATE DATAFRAME



```
Employee = Row("name", "dept", "state", "salary")
employee1 = Employee('James', 'Sales', 'CA', 100000)
employee2 = Employee('Mary', 'Finance', 'NY', 120000)
employee3 = Employee('Jane', 'Sales', 'WA', 160000)
employees = [employee1, employee2, employee3]
employeesDF = spark.createDataFrame(employees)
employeesDF.show()
```



```
+-----+-------+-----+------+
| name|   dept|state|salary|
+-----+-------+-----+------+
|James|  Sales|   CA|100000|
| Mary|Finance|   NY|120000|
| Jane|  Sales|   WA|160000|
+-----+-------+-----+------+
```

SENTENCE DATAFRAME

```python
sent_0 = Row(value='This is a sentence')
sent_1 = Row(value='This is another sentence')
sentences = [sent_0, sent_1]
sentenceDF = spark.createDataFrame(sentences)
sentenceDF.show()
```

```
+--------------------+
|               value|
+--------------------+
|  This is a sentence|
|This is another s...|
+--------------------+
```

DATAFRAME OPERATIONS

- Check type - `type(variable)`
- Display schema - [`printSchema`]
- Show content of the DataFrame - `show`
- Number of rows - `count`
- Number of columns - `len(dataFrame)`
- Select columns - `select`
- Summary - `describe`
- Group by columns - `groupBy`

DATAFRAME OPERATIONS

- Filter based on condition on columns - `filter`
- Sort by column name - `sort`
- Split string based on delimiter - `split`
- Explode(Map rows to columns) - `explode`
- Alias set - `alias`

SPLIT

```
from pyspark.sql.functions import *
```

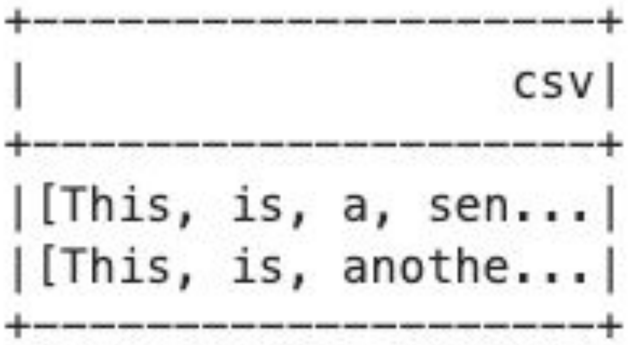
```
wordsDF1 = sentenceDF.select(split("value"," ").alias("csv"))
wordsDF1.show()
```

```
+--------------------+
|                 csv|
+--------------------+
|[This, is, a, sen...|
|[This, is, anothe...|
+--------------------+
```

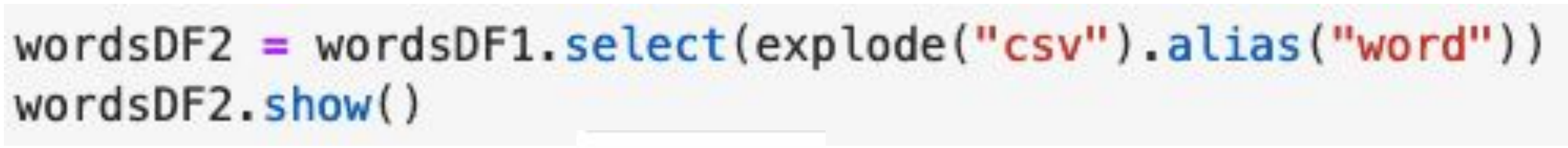
Split each line based on specified delimiter

EXPLODE

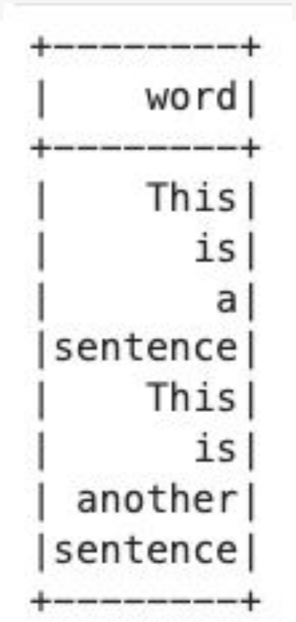
wordsDF1



```
+--------------------+
|                 csv|
+--------------------+
|[This, is, a, sen...|
|[This, is, anothe...|
+--------------------+
```



```
wordsDF2 = wordsDF1.select(explode("csv").alias("word"))
wordsDF2.show()
```



```
+--------+
|    word|
+--------+
|    This|
|      is|
|       a|
|sentence|
|    This|
|      is|
| another|
|sentence|
+--------+
```

Map columns to rows

Combining split() and explode()

```
wordsDF = sentenceDF.select(explode(split("value"," ")).alias("word"))
wordsDF.show()
```

```
+--------+
|    word|
+--------+
|    This|
|      is|
|       a|
|sentence|
|    This|
|      is|
| another|
|sentence|
+--------+
```

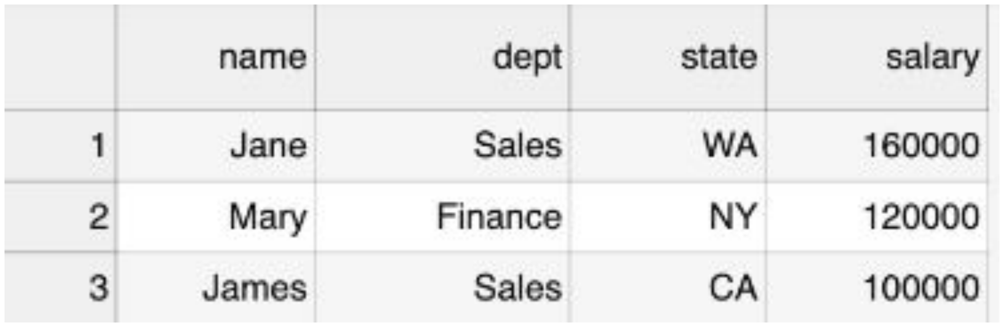
SAVING DATAFRAME TO FILE

DataFrame contents are coalesced into 1 partition and written to employees_sorted.csv/part-00000-*.csv

Save column headers

Overwrite existing file

```
employees_sortedDF.coalesce(1).\
   write.csv("file:///<path>/employees.csv", \
                header=True, mode="overwrite")
```

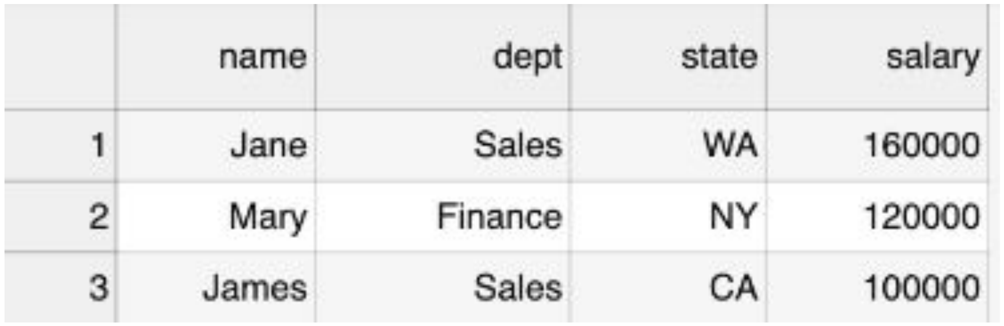


	name	dept	state	salary
1	Jane	Sales	WA	160000
2	Mary	Finance	NY	120000
3	James	Sales	CA	100000

SAVING DATAFRAME TO HDFS

DataFrame contents are coalesced into 1 partition and written to HDFS

```
employees_sortedDF.coalesce(1).\
   write.csv("hdfs:///<path>/employees.csv", \
               header=True, mode="overwrite")
```



	name	dept	state	salary
1	Jane	Sales	WA	160000
2	Mary	Finance	NY	120000
3	James	Sales	CA	100000