In [1]:
```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import *

spark = SparkSession\
    .builder\
    .appName("chapter-13-RDD-advanced")\
    .getOrCreate()

import os
SPARK_BOOK_DATA_PATH = os.environ['SPARK_BOOK_DATA_PATH']
```

In [15]:
```
myCollection = "Spark The Definitive Guide : Big Data Processing Made Si
  .split(" ")
words = spark.sparkContext.parallelize(myCollection, 2)
```

In [18]:
```
# COMMAND ----------

rdd = words.map(lambda word: (word.lower(), 1))
```

In [17]:
```
words.collect()
```

Out[17]:
```
['Spark',
 'The',
 'Definitive',
 'Guide',
 ':',
 'Big',
 'Data',
 'Processing',
 'Made',
 'Simple']
```

In [19]:
```
rdd.collect()
```

Out[19]:
```
[('spark', 1),
 ('the', 1),
 ('definitive', 1),
 ('guide', 1),
 (':', 1),
 ('big', 1),
 ('data', 1),
 ('processing', 1),
 ('made', 1),
 ('simple', 1)]
```

In [20]:
```
# COMMAND ----------

keyword = words.keyBy(lambda word: word.lower()[0])
```

In [21]:
```
keyword.collect()
```

Out[21]:
```
[('s', 'Spark'),
 ('t', 'The'),
 ('d', 'Definitive'),
 ('g', 'Guide'),
 (':', ':'),
 ('b', 'Big'),
 ('d', 'Data'),
 ('p', 'Processing'),
 ('m', 'Made'),
 ('s', 'Simple')]
```

In [22]:
```
# COMMAND ----------

keyword.mapValues(lambda word: word.upper()).collect()
```

Out[22]:
```
[('s', 'SPARK'),
 ('t', 'THE'),
 ('d', 'DEFINITIVE'),
 ('g', 'GUIDE'),
 (':', ':'),
 ('b', 'BIG'),
 ('d', 'DATA'),
 ('p', 'PROCESSING'),
 ('m', 'MADE'),
 ('s', 'SIMPLE')]
```

In [23]:
```
# COMMAND ----------

keyword.flatMapValues(lambda word: word.upper()).collect()
```

Out[23]:
```
[('s', 'S'),
 ('s', 'P'),
 ('s', 'A'),
 ('s', 'R'),
 ('s', 'K'),
 ('t', 'T'),
 ('t', 'H'),
 ('t', 'E'),
 ('d', 'D'),
 ('d', 'E'),
 ('d', 'F'),
 ('d', 'I'),
 ('d', 'N'),
 ('d', 'I'),
 ('d', 'T'),
 ('d', 'I'),
 ('d', 'V'),
 ('d', 'E'),
 ('g', 'G'),
 ('g', 'U'),
 ('g', 'I'),
 ('g', 'D'),
 ('g', 'E'),
 (':', ':'),
 ('b', 'B'),
 ('b', 'I'),
 ('b', 'G'),
 ('d', 'D'),
 ('d', 'A'),
 ('d', 'T'),
 ('d', 'A'),
 ('p', 'P'),
 ('p', 'R'),
 ('p', 'O'),
 ('p', 'C'),
 ('p', 'E'),
 ('p', 'S'),
 ('p', 'S'),
 ('p', 'I'),
 ('p', 'N'),
 ('p', 'G'),
 ('m', 'M'),
 ('m', 'A'),
 ('m', 'D'),
 ('m', 'E'),
 ('s', 'S'),
 ('s', 'I'),
 ('s', 'M'),
 ('s', 'P'),
 ('s', 'L'),
 ('s', 'E')]
```

In [24]:
```
# COMMAND ----------

keyword.keys().collect()
```

Out[24]:
```
['s', 't', 'd', 'g', ':', 'b', 'd', 'p', 'm', 's']
```

In [25]:
```
keyword.values().collect()
```

Out[25]:
```
['Spark',
 'The',
 'Definitive',
 'Guide',
 ':',
 'Big',
 'Data',
 'Processing',
 'Made',
 'Simple']
```

In [26]:
```
# COMMAND ----------

import random
```

In [27]:
```
distinctChars = words.flatMap(lambda word: list(word.lower())).distinct
  .collect()
```

In [28]:
```
distinctChars
```

Out[28]:
```
['s',
 'p',
 'r',
 'h',
 'd',
 'i',
 'g',
 'b',
 'c',
 'l',
 'a',
 'k',
 't',
 'e',
 'f',
 'n',
 'v',
 'u',
 ':',
 'o',
 'm']
```

In [29]:
```
sampleMap = dict(map(lambda c: (c, random.random()), distinctChars))
```

In [30]:
```
sampleMap
```

Out[30]:
```
{'s': 0.39780991844777414,
 'p': 0.3139593386048264,
 'r': 0.6595234516288071,
 'h': 0.4890327399856319,
 'd': 0.4170982802131986,
 'i': 0.7953451452463078,
 'g': 0.47466884441909674,
 'b': 0.5484171285346666,
 'c': 0.3976839608663212,
 'l': 0.47771876601935026,
 'a': 0.24785976542108645,
 'k': 0.34967788548329104,
 't': 0.775982478837066,
 'e': 0.820186314693686,
 'f': 0.42137931555502905,
 'n': 0.14442271869544232,
 'v': 0.153853295248098,
 'u': 0.13959839857665546,
 ':': 0.06251663438939137,
 'o': 0.17374719114484305,
 'm': 0.4255914281617944}
```

In [31]:
```
words.map(lambda word: (word.lower()[0], word))\
  .sampleByKey(True, sampleMap, 6).collect()
```

Out[31]:
```
[('t', 'The'), ('t', 'The'), ('g', 'Guide')]
```

In [ ]:
```
# COMMAND ----------

chars = words.flatMap(lambda word: word.lower())
KVcharacters = chars.map(lambda letter: (letter, 1))
```

In [ ]:
```
def maxFunc(left, right):
  return max(left, right)
def addFunc(left, right):
  return left + right
```

In [ ]:
```
nums = sc.parallelize(range(1,31), 5)
```

In [ ]:
```
# COMMAND ----------

KVcharacters.countByKey()
```

In [ ]:
```
# COMMAND ----------

KVcharacters.groupByKey().map(lambda row: (row[0], reduce(addFunc, row[
  .collect()
# note this is Python 2, reduce must be imported from functools in Pyth
```

In [ ]:
```
# COMMAND ----------

nums.aggregate(0, maxFunc, addFunc)
```

In [ ]:
```
# COMMAND ----------

depth = 3
nums.treeAggregate(0, maxFunc, addFunc, depth)
```

In [ ]:
```
# COMMAND ----------

KVcharacters.aggregateByKey(0, addFunc, maxFunc).collect()
```

In [ ]:
```
# COMMAND ----------

def valToCombiner(value):
  return [value]
def mergeValuesFunc(vals, valToAppend):
  vals.append(valToAppend)
  return vals
def mergeCombinerFunc(vals1, vals2):
  return vals1 + vals2
```

In [ ]:
```
outputPartitions = 6
KVcharacters\
  .combineByKey(
    valToCombiner,
    mergeValuesFunc,
    mergeCombinerFunc,
    outputPartitions)\
  .collect()
```

In [ ]:
```
# COMMAND ----------

KVcharacters.foldByKey(0, addFunc).collect()
```

In [ ]:
```
# COMMAND ----------

import random
distinctChars = words.flatMap(lambda word: word.lower()).distinct()
charRDD = distinctChars.map(lambda c: (c, random.random()))
charRDD2 = distinctChars.map(lambda c: (c, random.random()))
```

In [ ]:
```
charRDD.cogroup(charRDD2).take(5)
```

In [ ]:
```
# COMMAND ----------

keyedChars = distinctChars.map(lambda c: (c, random.random()))
outputPartitions = 10
KVcharacters.join(keyedChars).count()
```

In [ ]:
```
KVcharacters.join(keyedChars, outputPartitions).count()
```

In [ ]:
```
# COMMAND ----------

numRange = sc.parallelize(range(10), 2)
words.zip(numRange).collect()
```

In [ ]:
```
# COMMAND ----------

words.coalesce(1).getNumPartitions() # 1
```

In [3]:
```
# COMMAND ----------
file_path = SPARK_BOOK_DATA_PATH + "/data/retail-data/all/"

df = spark.read.option("header", "true").option("inferSchema", "true")\
  .csv(file_path)
```

In [5]:
```
df.take(5)
```

Out[5]:
```
[Row(InvoiceNo='536365', StockCode='85123A', Description='WHITE HANGIN
G HEART T-LIGHT HOLDER', Quantity=6, InvoiceDate='12/1/2010 8:26', Uni
tPrice=2.55, CustomerID=17850, Country='United Kingdom'),
 Row(InvoiceNo='536365', StockCode='71053', Description='WHITE METAL L
ANTERN', Quantity=6, InvoiceDate='12/1/2010 8:26', UnitPrice=3.39, Cus
tomerID=17850, Country='United Kingdom'),
 Row(InvoiceNo='536365', StockCode='84406B', Description='CREAM CUPID
HEARTS COAT HANGER', Quantity=8, InvoiceDate='12/1/2010 8:26', UnitPri
ce=2.75, CustomerID=17850, Country='United Kingdom'),
 Row(InvoiceNo='536365', StockCode='84029G', Description='KNITTED UNIO
N FLAG HOT WATER BOTTLE', Quantity=6, InvoiceDate='12/1/2010 8:26', Un
itPrice=3.39, CustomerID=17850, Country='United Kingdom'),
 Row(InvoiceNo='536365', StockCode='84029E', Description='RED WOOLLY H
OTTIE WHITE HEART.', Quantity=6, InvoiceDate='12/1/2010 8:26', UnitPri
ce=3.39, CustomerID=17850, Country='United Kingdom')]
```

In [6]:
```
df.count()
```

Out[6]:
```
541909
```

In [10]:
```
type(df), type(df.rdd)
```

Out[10]:
```
(pyspark.sql.dataframe.DataFrame, pyspark.rdd.RDD)
```

In [9]:
```
df.rdd.getNumPartitions()
```

Out[9]:
```
4
```

In [11]:
```
df2 = df.coalesce(10)
```

In [12]:
```
type(df2)
```

Out[12]:
```
pyspark.sql.dataframe.DataFrame
```

In [13]:
```
rdd = df2.rdd
```

In [14]:
```
rdd.getNumPartitions()
```

Out[14]:
```
4
```

In [ ]:
```
# COMMAND ----------

def partitionFunc(key):
  import random
  if key == 17850 or key == 12583:
    return 0
  else:
    return random.randint(1,2)

keyedRDD = rdd.keyBy(lambda row: row[6])
```

In [ ]:
```
keyedRDD\
  .partitionBy(3, partitionFunc)\
  .map(lambda x: x[0])\
  .glom()\
  .map(lambda x: len(set(x)))\
  .take(5)


# COMMAND ----------
```