Web: Inceptez.com Mail: info@inceptez.com Call: 7871299810, 7871299817

INCEPTEZ TECHNOLOGIES PYSPARK WORKOUTS

Shared Variables

Broadcast Variables:

```
broadcastVar = sc.broadcast(list(range(1,4)))

broadcastVar.value
```

Accumulator

```
accum = sc.accumulator(0)

rdd = sc.parallelize([1, 2, 3, 4,5,6])

rdd.first()

def f(x):

        global accum

        accum += x

rdd.foreach(f)

accum.value
```

Caching

```
hadoop fs -mkdir sparkdata

hadoop fs -put /home/hduser/pigdata/coursedetails.txt /user/hduser/sparkdata/

mydata = sc.textFile("/user/hduser/sparkdata/coursedetails.txt")

myrdd = mydata.map(lambda s:s.upper())

myrdd.cache()
```

```
myrdd2 = myrdd.filter(lambda s:s.startswith('H'))

myrdd2.count()
```

Persistence Levels

```
from pyspark import StorageLevel

myrdd.persist(StorageLevel.DISK_ONLY)
```

Transformations:

Map

```
x = sc.parallelize(["b", "a", "c"])

y = x.map(lambda z: (z, 1))

print(x.collect())

print(y.collect())
```

Output:

```
x: ['b', 'a', 'c']

y: [('b',1), ('a', 1), ('c', 1)]
```

Filter

```
x = sc.parallelize([1,2,3,4,5,6,7])

y = x.filter(lambda z: z%2 == 1) #keep odd values

print(x.collect())

print(y.collect())
```

FlatMap

```
x = sc.parallelize([1,2,3,4,5,10,20,30,1,2])

y = x.flatMap(lambda x: (x, x*100, 42))

print(x.collect())
print(y.collect())
```

Output

```
x: [1, 2, 3]

y: [1, 100, 42, 2, 200, 42, 3, 300, 42]
```

Sample (*withReplacement, fraction, seed*)

```
x = sc.parallelize([1, 2, 3, 4, 5])

y = x.sample(False, 0.4, 13)

print(x.collect())

print(y.collect())
```

Union

```
x = sc.parallelize([1,2,3], 2)

y = sc.parallelize([3,4], 1)

z = x.union(y)

print(z.glom().collect())
```

Output:

```
x: [1,2,3]

y: [3, 4]

z: [[1], [2, 3], [3, 4]]
```

GroupByKey

```
x = sc.parallelize([('B',5),('B',4),('A',3),('A',2),('A',1)],2)

y = x.groupByKey()

print(x.collect())

print(list((j[0], list(j[1])) for j in y.collect()))
```

Output

```
x: [('B', 5),('B', 4),('A', 3),('A', 2),('A', 1)]

y: [('A',[2, 3, 1]),('B',[5, 4])]
```

distinct

```
befdist = sc.parallelize([1, 2, 3, 4, 5,2,1,3,4,4,3,2])

applydist = befdist.distinct()

applydist.collect()
```

Join

```
x = sc.parallelize([("a", 2), ("b", 2)])

y = sc.parallelize([("a", 3), ("a", 4), ("b", 5)])

z = x.join(y)

print(z.collect())
```

Coalesce

```
x = sc.parallelize([1, 2, 3, 4, 5], 3)


y = x.filter(lambda z: z%2 == 1)

z = y.coalesce(2)

print(x.glom().collect())

print(y.glom().collect())
```

Repartition

```
z = x.repartition(3)

print(x.glom().collect())

print(z.glom().collect())
```

Actions:

Collect

```
x = sc.parallelize([1,2,3], 2)

y = x.collect()

print(x.glom().collect())

print(y)
```

Reduce:

```
x = sc.parallelize([1,2,3,4])

y = x.reduce(lambda a,b: a+b)

print(x.collect())

print(y)
```

Count

```
x = sc.parallelize([3,2,9,0,1, 4, 5,6,7,8,9,10], 2)

x.count()
```

Take

```
x.take(2)

x.takeOrdered(3)
```

CountByKey

```
x = sc.parallelize([('B',5),('B',4),('A',3),('A',2),('A',1)])

x.countByKey()
```

saveAsTextFile

```
x.saveAsTextFile('file:///home/hduser/sparkout')
```

INCEPTEZ TECHNOLOGIES

Identify the Trending Technologies in the market (Sentimental Analysis) - Word count Program

```
counts = sc.textFile("/user/hduser/sparkdata/coursedetails.txt").flatMap (lambda line: line.split('
')).map(lambda word: (word,1)).groupByKey(lambda v1,v2: v1+v2).collect()

print(counts)
```

UseCase: Analyze the auctions data to identify bids:

```
#To define indexes:
auctionid = 0
bid = 1
bidtime = 2
bidder = 3
bidderrate = 4
openbid = 5
price = 6
itemtype = 7
daystolive = 8

#To load the file
auctionRDD = sc.textFile("/user/hduser/sparkdata/auctiondata.csv").flatMap(lambda
line:line.encode("ascii","ignore").split(","))

#1. To see the first element of the RDD
auctionRDD.first()

# 2. To see the first 5 elements of the RDD
auctionRDD.take(5)

#3. What is the total number of bids?
totbids = auctionRDD.count()
print totbids

#4. What is the total number of distinct items that were auctioned?
totitems = auctionRDD.map(lambda line:line[0]).distinct().count()
print totitems

#5. What is the total number of item types that were auctioned?
totitemtypes=auctionRDD.map(lambda line:line[itemtype]).distinct().count()
print totitemtypes

#6. What is the total number of bids per item type?
bids_itemtype = auctionRDD.map(lambda x:(x[itemtype],1)).reduceByKey(lambda x,y:x+y).collect()
print bids_itemtype

#7. What is the total number of bids per auction?
bids_auctionRDD = auctionRDD.map(lambda x:(x[auctionid],1)).reduceByKey(lambda x,y:x+y)
bids_auctionRDD.take(5) #just to see the first 5 elements

#8. Across all auctioned items, what is the max number of bids?
maxbids = bids_auctionRDD.map(lambda x:x[bid]).reduce(max)
```

```
print maxbids

#9. Across all auctioned items, what is the minimum of bids?
minbids = bids_auctionRDD.map(lambda x:x[bid]).reduce(min)
print minbids

#10. What is the average bid?
avgbids = totbids/totitems
print avgbids
```

Work with Pair RDD

Load Data into Spark

1. We define the mapping for our input variables. While this isn’t a necessary step, it makes it easier to refer to the different fields by names.

```
IncidntNum = 0
Category = 1
Descript = 2
DayOfWeek = 3
Date = 4
Time = 5
PdDistrict = 6
Resolution = 7
Address = 8
X = 9
Y = 10
PdId = 11
```

2. To load data into Spark, at the Scala command prompt:

```
hadoop fs -put /home/hduser/sfpd.csv /user/hduser/sparkdata/

sfpdRDD = sc.textFile("/user/hduser/sparkdata/sfpd.csv").map(lambda
x:x.encode("ascii","ignore").split(","))
```

1. How do you see the first element of the inputRDD (sfpdRDD)?

```
print(sfpdRDD.first())
```

2. What do you use to see the first 5 elements of the RDD?

```
print(sfpdRDD.take(5))
```

3. What is the total number of incidents?

```
totincs = sfpdRDD.count()
```

4. What is the total number of distinct resolutions?

```
totres = sfpdRDD.map(lambda x: x[7]).distinct().count()
print totres
```

5. List the PdDistricts.

```
districts = sfpdRDD.map(lambda x: x[6]).distinct()
print(districts.collect())
```

Create & Explore PairRDD

Create pair RDD & apply pair RDD operations

1. Which five districts have the highest incidents?

```
higinc = sfpdRDD.map(lambda x: (x[0],1)).reduceByKey(lambda x,y:x+y).map(lambda
x:(x[1],x[0])).sortByKey(False).map(lambda x:(x[1],x[0])).take(5)
```

2. Which five addresses have the highest incidents?

```
higadd = sfpdRDD.map(lambda x: (x[8],1)).reduceByKey(lambda x,y:x+y).map(lambda
x:(x[1],x[0])).sortByKey(False).take(5)
```

3. What are the top three categories of incidents?

```
higcat = sfpdRDD.map(lambda x:(x[1],1)).reduceByKey(lambda x,y:x+y).map(lambda
x:(x[1],x[0])).sortByKey(False).take(3)
```

4. . What is the count of incidents by district?

```
num_inc_dist=sfpdRDD.map(lambda incident:(incident[PdDistrict],1)).reduceByKey(lambda x,y:x+y)
num_inc_dist=sfpdRDD.map(lambda incident:(incident[PdDistrict],1)).countByKey()
print(num_inc_dist)
```

Join Pair RDDs

This activity illustrates how joins work in Spark. There are two small datasets provided for this activity - J_AddCat.csv and J_AddDist.csv.

1. Given these two datasets, you want to find the type of incident and district for each address. What is one way of doing this? (HINT: An operation on pairs or pairRDDs)

```
catRDD = sc.textFile("/user/hduser/sparkdata/J_AddCat.csv").map(lambda x:x.split(",")).map(lambda
x:(x[1],x[0]))
distRDD = sc.textFile("/user/hduser/sparkdata/J_AddDist.csv").map(lambda x:x.split(",")).map(lambda
x:(x[1],x[0]))
```

```
joi = catRDD.join(distRDD)
```

2. What is the size of the resulting dataset from a join? Why?

```
joi.collect()
[(u'1400_Block_of_CLEMENT_ST', (u'BRIBERY', u'RICHMOND')), (u'1400_Block_of_VANDYKE_AV',
(u'BRIBERY', u'BAYVIEW')), (u'0_Block_of_SOUTHPARK_AV', (u'BRIBERY', u'SOUTHERN')),
(u'1900_Block_of_MISSION_ST', (u'BRIBERY', u'MISSION')), (u'100_Block_of_BLUXOME_ST', (u'BAD
CHECKS', u'SOUTHERN'))]
```

```
joi.count()
joi.take(10)
```

3. If you did a right outer join on the two datasets with Address/Category being the source RDD, what would be the size of the resulting dataset? Why?

```
Rightjoi = catRDD.rightOuterJoin(distRDD)
Rightjoi.collect()

[(u'1400_Block_of_CLEMENT_ST', (u'BRIBERY', u'RICHMOND')), (u'0_Block_of_CHUMASERO_DR', (None,
u'TARAVAL')), (u'1400_Block_of_VANDYKE_AV', (u'BRIBERY', u'BAYVIEW')),
(u'0_Block_of_SOUTHPARK_AV', (u'BRIBERY', u'SOUTHERN')), (u'1100_Block_of_MISSION_ST', (None,
u'SOUTHERN')), (u'1900_Block_of_MISSION_ST', (u'BRIBERY', u'MISSION')), (u'300_Block_of_BERRY_ST',
(None, u'SOUTHERN')), (u'100_Block_of_ROME_ST', (None, u'INGLESIDE')),
(u'100_Block_of_BLUXOME_ST', (u'BAD CHECKS', u'SOUTHERN'))]

Rightjoi.count()
```

4. If you did a left outer join on the two datasets with Address/Category being the source RDD, what would be the size of the resulting dataset? Why?

```
Leftjoi = catRDD.leftOuterJoin(distRDD)
Leftjoi.collect()
[(u'SUTTER_ST/MASON_ST', (u'EMBEZZLEMENT', None)), (u'1100_Block_of_SELBY_ST',
(u'EMBEZZLEMENT', None)), (u'1400_Block_of_CLEMENT_ST', (u'BRIBERY', u'RICHMOND')),
(u'1400_Block_of_VANDYKE_AV', (u'BRIBERY', u'BAYVIEW')), (u'1500_Block_of_15TH_ST',
(u'EMBEZZLEMENT', None)), (u'2000_Block_of_MARKET_ST', (u'EMBEZZLEMENT', None)),
(u'0_Block_of_SOUTHPARK_AV', (u'BRIBERY', u'SOUTHERN')), (u'100_Block_of_JEFFERSON_ST',
(u'EMBEZZLEMENT', None)), (u'1600_Block_of_FILLMORE_ST', (u'EMBEZZLEMENT', None)),
(u'200_Block_of_BUSH_ST', (u'EMBEZZLEMENT', None)), (u'1900_Block_of_MISSION_ST', (u'BRIBERY',
u'MISSION')), (u'800_Block_of_MARKET_ST', (u'EMBEZZLEMENT', None)),
(u'100_Block_of_BLUXOME_ST', (u'BAD CHECKS', u'SOUTHERN'))]

Leftjoi.count()
```