RDD Transformations and Actions

AWS Instance: https://ec2-3-18-111-123.us-east-2.compute.amazonaws.com:8888/ (https://ec2-3-18-111-123.us-east-2.compute.amazonaws.com:8888/)

In [4]:
```
%%writefile example2.txt
the first line
the second
then the third
the fourth
fifth
```

```
Overwriting example2.txt
```

In [5]:
```
from pyspark import SparkContext
```

In [6]:
```
sc = SparkContext()
```

In [8]:
```
sc.textFile('example2.txt')
```

Out[8]:
```
example2.txt MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.jav
a:-2
```

In [9]:
```
text_rdd = sc.textFile('example2.txt')
```

Map

In [10]:
```
words = text_rdd.map(lambda s: s.split())
```

In [11]:
```
words.collect()
```

Out[11]:
```
[['the', 'first', 'line'],
 ['the', 'second'],
 ['then', 'the', 'third'],
 ['the', 'fourth'],
 ['fifth']]
```

In [12]:
```
words.count()
```

Out[12]:
```
5
```

In [13]:
```
text_rdd.collect()
```

Out[13]:
```
['the first line', 'the second', 'then the third', 'the fourth', 'fifth']
```

FlatMap

In [14]:
```
words_flatmap = text_rdd.flatMap(lambda s: s.split())
```

In [15]:
```
words_flatmap.collect()
```

Out[15]:
```
['the',
 'first',
 'line',
 'the',
 'second',
 'then',
 'the',
 'third',
 'the',
 'fourth',
 'fifth']
```

RDD and key value pairs

In [16]:
```
%%writefile services.txt
#EventId    Timestamp    Customer   State    ServiceID    Amount
201       10/13/2017      100       NY       131          100.00
204       10/18/2017      700       TX       129          450.00
202       10/15/2017      203       CA       121          200.00
206       10/19/2017      202       CA       131          500.00
203       10/17/2017      101       NY       173          750.00
205       10/19/2017      202       TX       121          200.00
```

```
Writing services.txt
```

In [17]:
```
services = sc.textFile('services.txt')
```

In [18]:
```
services.collect()
```

Out[18]:
```
['#EventId    Timestamp    Customer   State    ServiceID    Amount',
 '201       10/13/2017      100       NY       131          100.00',
 '204       10/18/2017      700       TX       129          450.00',
 '202       10/15/2017      203       CA       121          200.00',
 '206       10/19/2017      202       CA       131          500.00',
 '203       10/17/2017      101       NY       173          750.00',
 '205       10/19/2017      202       TX       121          200.00']
```

In [19]:
```
services.collect
```

Out[19]:
```
<bound method RDD.collect of services.txt MapPartitionsRDD[8] at textFile at
NativeMethodAccessorImpl.java:-2>
```

In [20]:
```
services.take(2)
```

Out[20]:
```
['#EventId    Timestamp    Customer   State    ServiceID    Amount',
 '201       10/13/2017      100       NY       131          100.00']
```

```
In [21]: services.map(lambda line: line.split())
```

```
Out[21]: PythonRDD[10] at RDD at PythonRDD.scala:48
```

```
In [22]: services.map(lambda line: line.split()).take(3)
```

```
Out[22]: [['#EventId', 'Timestamp', 'Customer', 'State', 'ServiceID', 'Amount'],
          ['201', '10/13/2017', '100', 'NY', '131', '100.00'],
          ['204', '10/18/2017', '700', 'TX', '129', '450.00']]
```

Remove #

```
In [23]: services.map(lambda line: line[1:] if line[0]=='#' else line ).collect()
```

```
Out[23]: ['EventId    Timestamp    Customer   State    ServiceID    Amount',
          '201       10/13/2017      100       NY       131          100.00',
          '204       10/18/2017      700       TX       129          450.00',
          '202       10/15/2017      203       CA       121          200.00',
          '206       10/19/2017      202       CA       131          500.00',
          '203       10/17/2017      101       NY       173          750.00',
          '205       10/19/2017      202       TX       121          200.00']
```

```
In [24]: type(services)
```

```
Out[24]: pyspark.rdd.RDD
```

```
In [27]: services.count()
```

```
Out[27]: 7
```

```
In [28]: clean = services.map(lambda line: line[1:] if line[0]=='#' else line )
```

```
In [29]: clean.take(3)
```

```
Out[29]: ['EventId    Timestamp    Customer   State    ServiceID    Amount',
          '201       10/13/2017      100       NY       131          100.00',
          '204       10/18/2017      700       TX       129          450.00']
```

```
In [30]: clean = clean.map(lambda line: line.split())
```

```
In [31]: clean.collect()
```

```
Out[31]: [['EventId', 'Timestamp', 'Customer', 'State', 'ServiceID', 'Amount'],
          ['201', '10/13/2017', '100', 'NY', '131', '100.00'],
          ['204', '10/18/2017', '700', 'TX', '129', '450.00'],
          ['202', '10/15/2017', '203', 'CA', '121', '200.00'],
          ['206', '10/19/2017', '202', 'CA', '131', '500.00'],
          ['203', '10/17/2017', '101', 'NY', '173', '750.00'],
          ['205', '10/19/2017', '202', 'TX', '121', '200.00']]
```

```
In [33]: pairs = clean.map(lambda new_list: (new_list[3], new_list[-1]))
```

In [34]:
```
pairs.collect()
```

Out[34]:
```
[('State', 'Amount'),
 ('NY', '100.00'),
 ('TX', '450.00'),
 ('CA', '200.00'),
 ('CA', '500.00'),
 ('NY', '750.00'),
 ('TX', '200.00')]
```

In [35]:
```
type(pairs)
```

Out[35]:
```
pyspark.rdd.PipelinedRDD
```

To use reduceByKey, convert data to the form of Key value pairs with tuples

In [37]:
```
values = pairs.reduceByKey(lambda amount1, amount2: amount1+amount2)
```

In [38]:
```
values.collect()
```

Out[38]:
```
[('State', 'Amount'),
 ('NY', '100.00750.00'),
 ('TX', '450.00200.00'),
 ('CA', '200.00500.00')]
```

In [39]:
```
values = pairs.reduceByKey(lambda amount1, amount2: float(amount1) + float(amo
unt2))
```

In [40]:
```
values.collect()
```

Out[40]:
```
[('State', 'Amount'), ('NY', 850.0), ('TX', 650.0), ('CA', 700.0)]
```

In [41]:
```
clean.collect()
```

Out[41]:
```
[['EventId', 'Timestamp', 'Customer', 'State', 'ServiceID', 'Amount'],
 ['201', '10/13/2017', '100', 'NY', '131', '100.00'],
 ['204', '10/18/2017', '700', 'TX', '129', '450.00'],
 ['202', '10/15/2017', '203', 'CA', '121', '200.00'],
 ['206', '10/19/2017', '202', 'CA', '131', '500.00'],
 ['203', '10/17/2017', '101', 'NY', '173', '750.00'],
 ['205', '10/19/2017', '202', 'TX', '121', '200.00']]
```

Grab State & Amount

In [56]:
```
step1 = clean.map(lambda lst: (lst[3], lst[-1]))
```

In [57]:
```
step1.collect()
```

Out[57]:
```
[('State', 'Amount'),
 ('NY', '100.00'),
 ('TX', '450.00'),
 ('CA', '200.00'),
 ('CA', '500.00'),
 ('NY', '750.00'),
 ('TX', '200.00')]
```

Reduce by Key

In [58]:
```
step2 = step1.reduceByKey(lambda amt1, amt2: float(amt1) + float(amt2))
```

In [59]:
```
step2.collect()
```

Out[59]:
```
[('State', 'Amount'), ('NY', 850.0), ('TX', 650.0), ('CA', 700.0)]
```

Remove State, Amounts titles

In [60]:
```
step3 = step2.filter(lambda x: not x[0]=='State')
```

In [61]:
```
step3.collect()
```

Out[61]:
```
[('NY', 850.0), ('TX', 650.0), ('CA', 700.0)]
```

Sort results by Amounts

In [62]:
```
step4 = step3.sortBy(lambda stamt: stamt[1], ascending=False)
```

In [63]:
```
step4.collect()
```

Out[63]:
```
[('NY', 850.0), ('CA', 700.0), ('TX', 650.0)]
```

Tuple Unpacking

In [64]:
```
x = ['Id', 'State', 'Amount']
```

In [65]:
```
def amount(x):
    #unpack values
    (Id, St, Amt) = x
    return Amt
```

In [66]:
```
amount(x)
```

Out[66]:
```
'Amount'
```

In [ ]: