2. Cleaning the Data

To prepare the data for the steps that follow, you first need to remove or fix the issues you discovered above. You should also summarize the data into a less verbose format so that you can reduce the volume of data you'll have to process during the exercises.

Since you have thoroughly explored the data and looked into the suspicious results, creating the data cleansing utility should be pretty straightforward. The general idea is to generate a file with a single aggregate record per reducer. Again you will want to use MapReduce, and again you'll choose Python for its ease of working with JSON data.

You should also decide how to approach and process the play-related events. There are a number of events, such as "Play," "Pause," "Advance," etc. that all relate to playing content. While you could track them all independently, you need to consider whether there's any value in doing so. Thinking ahead to the problems you need to answer, you have three applications for the data. For the first problem, you could try to train a classifier based on the watching behavior, but that sounds like an uphill approach of dubious value. In the clustering problem, you could try to cluster user sessions based on watching behavior, but trying to cluster user behaviors according to how they pause, play and advance the content sounds like a challenging problem unto itself. In the recommendation problem, there's not much difference among the play events. They all convey pretty much the same implicit information. Given all that, you're better off if you collapse all of the play events into a single event type for now; you can revisit that decision later if needed. You should collapse all of the play events for a given item within a given session into a single event that gives the last position in the content reached. You lose some data by doing that, but it's data that you're not going to use unless you have to.

You should also consider how to process the "auth" and "refId" fields. They are only present in a portion of the events in the logs, which poses a problem. You don't know what they are, so you can't recreate them in the events where they're missing. For the classification problem, it's unlikely that those fields are going to carry signal for whether a user is an adult or not. The fields appear to be identifiers from our summary, and when they show up, they're present in every event, so they are also unlikely to help with the session clustering problem. Finally, they have nothing to do with content ratings, as far as you can tell. Since you're missing large chunks of the data for these fields and you have no apparent use for them, you can just leave them for out and stay as simple and clean as possible.

For the mapper, you want to translate the JSON into simple attributes so that you can recombine them into a single record per session in the reducer. For the attributes that appear in every log entry, session ID and user ID will go into the key, and you can just drop the user agent. The user agent does have some potentially useful information in it, but it is easily spoofed and hence unreliable, and getting that information out would require a large amount of manual effort. In any case, odds are good that whatever analytics solution the site uses already captures the valuable information from the user agents.

Step 1. Write a mapper to clean the JSON

Create a new file called `clean_map.py` with the following contents:

```
#!/usr/bin/python
import dateutil.parser
import json
import sys
from datetime import tzinfo, timedelta, datetime

def main():
   for line in sys.stdin:
      # Correct for double quotes
      data = json.loads(line.replace('""', '"'))

      # Correct for variance in field names
      item_id = 'item_id'
      session_id = 'session_id'
      created_at = 'created_at'

      if 'sessionID' in data:
         session_id = 'sessionID'
      if 'createdAt' in data:
         created_at = 'createdAt'
      elif 'craetedAt' in data:
         created_at = 'craetedAt'

      if 'payload' in data and 'itemId' in data['payload']:
         item_id = 'itemId'

      # Prepare the key
      userid = data['user']
      sessionid = data[session_id]
      timestamp = total_seconds(dateutil.parser.parse(data[created_at]) - EPOCH)

      key = '%s,%10d,%s' % (userid, timestamp, sessionid)

      # Write out the value
      if data['type'] == "Account" and data['payload']['subAction'] == "parentalControls":
         print "%s\tx:%s" % (key, data['payload']['new'])
      elif data['type'] == "Account":
         print "%s\tc:%s" % (key, data['payload']['subAction'])
      elif data['type'] == "AddToQueue":
         print "%s\ta:%s" % (key, data['payload'][item_id])
      elif data['type'] == "Home":
         print "%s\tP:%s" % (key, ",".join(data['payload']['popular']))
         print "%s\tR:%s" % (key, ",".join(data['payload']['recommended']))
         print "%s\tr:%s" % (key, ",".join(data['payload']['recent']))
      elif data['type'] == "Hover":
         print "%s\th:%s" % (key, data['payload'][item_id])
      elif data['type'] == "ItemPage":
         print "%s\ti:%s" % (key, data['payload'][item_id])
      elif data['type'] == "Login":
         print "%s\tL:" % key
      elif data['type'] == "Logout":
         print "%s\tl:" % key
      elif data['type'] == "Play" or \
           data['type'] == "Pause" or \
           data['type'] == "Position" or \
           data['type'] == "Stop" or \
           data['type'] == "Advance" or \
           data['type'] == "Resume":
         print "%s\tp:%s,%s" % (key, data['payload']['marker'], data['payload'][item_id])
      elif data['type'] == "Queue":
         print "%s\tq:" % key
      elif data['type'] == "Rate":
         print "%s\tt:%s,%s" % (key, data['payload'][item_id], data['payload']['rating'])
      elif data['type'] == "Recommendations":
         print "%s\tC:%s" % (key, ",".join(data['payload']['recs']))
      elif data['type'] == "Search":
         print "%s\tS:%s" % (key, ",".join(data['payload']['results']))
      elif data['type'] == "VerifyPassword":
         print "%s\tv:" % key
      elif data['type'] == "WriteReview":
         print "%s\tw:%s,%s,%s" % (key, data['payload'][item_id], data['payload']['rating'],
data['payload']['length'])

"""
# Return the number of seconds since the epoch, calculated the hard way.
"""
def total_seconds(td):
   return (td.microseconds + (td.seconds + td.days * 24 * 3600) * 10**6) / 10**6

"""
A constant for 0 time difference
"""
ZERO = timedelta(0)
```

```
"""
A Timezone class for UTC
"""
class UTC(tzinfo):
    def utcoffset(self, dt):
        return ZERO

    def tzname(self, dt):
        return "UTC"

    def dst(self, dt):
        return ZERO

"""
# A constant for the beginning of the epoch
"""
EPOCH = datetime(1970,1,1,tzinfo=UTC())
if __name__ == '__main__':
   main()
```

The script handles the “Recommendations” event (which happens when the user reviews a recommendation) separately from the recommendation field in the “Home” event (which just lists the items that were recommended for the user). The former represents user behavior, while the latter does not.

The one-letter codes in the mapper output are a compact way to let the reducer know the types of the records. The script converts the timestamps to seconds since the epoch so that they can be sorted consistently. Converting timestamps to seconds since the epoch is considerably more difficult in Python than it should be but it pays off in the end.

For the reducer, the script writes final output in JSON again because of the hierarchical nature of the session data. You could try unrolling that hierarchy in order to get down to a simpler format, like CSV, but for this stage you’re probably better off avoiding complication. Later when you’re extracting specific parts of the data, you’ll find it much easier to simplify the format.

The reducer script also needs to consider the treatment of parental control events. The script could simply set the kid/adult flag for the account, but that would mean that all the actions taken in that session before the event would be incorrectly associated with a kid account. You could address the problem by dropping all events in the session before the parental control event, but that will result in data loss. Even worse, it would result in the loss of labeled data, because you know that before the event the account must have been the opposite status. The best approach is to have your script create a new session when it encounters a parental control event so that it captures both sets of behavior data. This approach creates duplicate session with the same ID, which needs to be corrected later. Keep in mind, your script must also set a label for the session when it encounters an account action other than parental controls because those events indicate that account is being used by an adult.

Step 2. Write a reducer to clean the JSON

Create a new file called `clean_reduce.py` with the following contents:

```
!/usr/bin/python

import json
import sys

def main():
  currentSession = None
  lastTime = None
  data = {}

  for line in sys.stdin:
     key, value = line.strip().split('\t')
     userid, timestr, sessionid = key.split(',')
     flag, payload = value.split(':')

     if sessionid != currentSession:
        currentSession = sessionid;
        kid = None

        if data:
           data['end'] = lastTime

           print json.dumps(data)

        data = {"popular": [], "recommended": [], "searched": [], "hover": [], "queued": [],
                "browsed": [], "recommendations": [], "recent": [], "played": {}, "rated": {},
"reviewed": {},
                "actions": [], "kid": kid, "user": userid, "session": sessionid, "start": timestr}

     if flag == "C":
        data['recommendations'].extend(payload.split(","))
     elif flag == "L":
        data['actions'].append('login')
     elif flag == "P":
        data['popular'].extend(payload.split(","))
     elif flag == "R":
        data['recommended'].extend(payload.split(","))
     elif flag == "S":
        data['searched'].extend(payload.split(","))
     elif flag == "a":
        data['queued'].append(payload)
     elif flag == "c":
        data['actions'].append(payload)
        data['kid'] = False
     elif flag == "h":
        data['hover'].append(payload)
     elif flag == "i":
        data['browsed'].append(payload)
     elif flag == "l":
        data['actions'].append('logout')
     elif flag == "p":
        (marker, itemid) = payload.split(",")
        data['played'][itemid] = marker
     elif flag == "q":
        data['actions'].append('reviewedQueue')
     elif flag == "r":
        data['recent'].extend(payload.split(","))
     elif flag == "t":
        (itemid, rating) = payload.split(",")
        data['rated'][itemid] = rating
     elif flag == "v":
        data['actions'].append('verifiedPassword')
        data['kid'] = False
     elif flag == "w":
        (itemid, rating, length) = payload.split(",")
        data['reviewed'][itemid] = {}
        data['reviewed'][itemid]["rating"] = rating
        data['reviewed'][itemid]["length"] = length
     elif flag == "x":
        # If we see a parental controls event, assume this session was the opposite and start a new
session
        data['kid'] = payload != "kid"
        data['end'] = lastTime

        print json.dumps(data)

        data = {"popular": [], "recommended": [], "searched": [], "hover": [], "queued": [],
                "browsed": [], "recommendations": [], "recent": [], "played": {}, "rated": {},
"reviewed": {},
                "actions": [], "kid": payload == "kid", "user": userid, "session": sessionid, "start":
timestr}

     lastTime = timestr

    data['end'] = lastTime
    print json.dumps(data, sort_keys=True)

if __name__ == '__main__':
    main()
```

Step 3. Run the job

Run the job with the following commands:

```
$ hadoop jar $STREAMING -input data/heckle/ -input data/jeckle/ -output clean -mapper clean_map.py -
file clean_map.py -reducer clean_reduce.py -file clean_reduce.py
packageJobJar: [clean_map.py, clean_reduce.py, /tmp/hadoop-training/hadoop-unjar7699202339998285666/]
[] /var/folders/ll/xl67db9x2rs47q5fs4b255rr0000gp/T/streamjob1502790727777748866.jar tmpDir=null
13/11/18 11:42:31 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/11/18 11:42:31 INFO mapred.FileInputFormat: Total input paths to process : 20
13/11/18 11:42:31 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-training/mapred/local]
13/11/18 11:42:31 INFO streaming.StreamJob: Running job: job_201311081846_0009
13/11/18 11:42:31 INFO streaming.StreamJob: To kill this job, run:
13/11/18 11:42:31 INFO streaming.StreamJob: /Users/training/Downloads/hadoop-2.0.0-mr1-
cdh4.2.1//bin/hadoop job  -Dmapred.job.tracker=192.168.56.101:8021 -kill job_201311081846_0009
13/11/18 11:42:31 INFO streaming.StreamJob: Tracking URL: http://192.168.56.101:50030/jobdetails.jsp?
jobid=job_201311081846_0009
13/11/18 11:42:32 INFO streaming.StreamJob:  map 0%  reduce 0%
13/11/18 11:42:42 INFO streaming.StreamJob:  map 3%  reduce 0%
...
13/11/18 11:44:48 INFO streaming.StreamJob:  map 70%  reduce 23%
13/11/18 11:45:12 INFO streaming.StreamJob:  map 100%  reduce 100%
13/11/18 11:45:12 INFO streaming.StreamJob: To kill this job, run:
13/11/18 11:45:12 INFO streaming.StreamJob: /usr/bin/hadoop job  -
Dmapred.job.tracker=192.168.56.101:8021 -kill job_201311081846_0009
13/11/18 11:45:12 INFO streaming.StreamJob: Tracking URL: http://192.168.56.101:50030/jobdetails.jsp?
jobid=job_201311081846_0009
13/11/18 11:45:12 ERROR streaming.StreamJob: Job not successful. Error: NA
13/11/18 11:45:12 INFO streaming.StreamJob: killJob...
Streaming Command Failed!
```

Step 4. Debug the job

Again, the job fails. To find out why the job failed, open a web browser and go to the job detail page URL given in the command output. In the example above, that URL is: `http://localhost:50030/jobdetails.jsp?jobid=job_201311081846_0009` . Yours will differ. From the job detail page, you can see that 6 map tasks were killed after 8 failed attempts. (The last failed attempt may also be killed, so it may be counted as 7 failed and 1 killed.) Click the link for the failed attempts to view the job failures page where you find the stack trace in the second column that the problem is a broken pipe again. As before, a broken pipe in Hadoop streaming means there was a problem with the script. Click on one of the links for the task logs in the last column, and you'll see (in the `stderr` section) that a print statement in the mapper is causing a `KeyError` to be thrown. Interestingly, 14 map tasks managed to complete successfully, so odds are that there is something in the data that first appears in the input split that causes the error. Since the error message says that the "marker" key cannot be found, use that to look for the data records that are causing the problem.

1. Dig through the data

Run another streaming job to dig through the entire data set. Streaming has trouble with escaped or deeply nested quotes, so leave out the quotes around the fields for which you're searching.

Run the job with the following command and then run `hadoop fs -getmerge`:

```
$ hadoop jar $STREAMING -D stream.non.zero.exit.is.failure=false -D mapred.reduce.tasks=0 -input
data/heckle/ -input data/jeckle/ -output marker -mapper "bash -c 'grep -e Play -e Pause -e Position -e
Stop -e Advance -e Resume | grep -v marker'"
…
$ hadoop fs -getmerge marker bad.json
```

The `hadoop fs -getmerge` command concatenates the contents of a directory into a single file on the local disk. The files are concatenated in lexical sort order of their names.

You now have a file called `bad.json` that contains all of the bad records. In this example, we recommend using `-getmerge` on the directory instead `-cat` on the individual files because the command did not use a reduce phase, so there's an output file for every map task. the options set the numbers of reducers to 0 because there's no need to sort or aggregate the data.

You can now see what's happening with the data:

```
$ head bad.json
{"auth": "24cfd88:2d157097", "craetedAt": "2013-05-09T08:00:03Z", "payload": {}, "refId": "61b37fb5",
"sessionID": "f63213df-54ee-4e94-a667-f22fdf3ccc09", "type": "Play", "user": 38600072, "userAgent":
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:12.0) Gecko/20100101 Firefox/12.0"}
{"auth": "122e5f8:76bf9f21", "craetedAt": "2013-05-09T08:00:15Z", "payload": {}, "refId": "1c46d559",
"sessionID": "3d728cce-e9a0-4a63-868d-23bf90161bab", "type": "Play", "user": 19064312, "userAgent":
"Mozilla/5.0 (X11; Linux i686) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.79
Safari/535.11"}
{"auth": "24d9690:75eb97e4", "craetedAt": "2013-05-09T08:00:17Z", "payload": {}, "refId": "44334724",
"sessionID": "471cf55f-9ea3-47f8-bc2b-a777a376c841", "type": "Play", "user": 38639248, "userAgent":
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/536.11 (KHTML, like Gecko)
Chrome/20.0.1132.43 Safari/536.11"}
{"auth": "5496592:16990573", "craetedAt": "2013-05-09T08:00:18Z", "payload": {}, "refId": "187ed658",
"sessionID": "fcf4b69f-f824-42ce-bcf6-ce0e0e0d12d5", "type": "Play", "user": 88696210, "userAgent":
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.2 (KHTML, like Gecko) Chrome/15.0.874.102
Safari/535.2"}
{"auth": "937df9:7de2b69b", "craetedAt": "2013-05-09T08:00:19Z", "payload": {}, "refId": "df45d969",
"sessionID": "14ee7957-1eee-454b-ad31-188e4183f77f", "type": "Play", "user": 9666041, "userAgent":
"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; managedpc)"}
{"auth": "18b8a82:6d07a4c8", "craetedAt": "2013-05-09T08:00:19Z", "payload": {}, "refId": "5f2f4220",
"sessionID": "46369e3b-fb1f-4941-9181-68409ebcedf6", "type": "Play", "user": 25922178, "userAgent":
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14 (KHTML, like Gecko)"}
{"auth": "519d3a6:6f015056", "craetedAt": "2013-05-09T08:00:20Z", "payload": {}, "refId": "39c200b9",
"sessionID": "f1d50b04-bec2-45e3-9097-30c6e8c696b6", "type": "Play", "user": 85578662, "userAgent":
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2) Gecko/20100115 Firefox/3.6"}
{"auth": "1ddb3d6:773f0262", "craetedAt": "2013-05-09T08:00:20Z", "payload": {}, "refId": "2b722d44",
"sessionID": "edf4e5b9-175b-4345-a4b6-64bb6296654e", "type": "Play", "user": 31306710, "userAgent":
"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; MATM)"}
{"auth": "466aeae:241324c5", "craetedAt": "2013-05-09T08:00:22Z", "payload": {}, "refId": "2896f34f",
"sessionID": "acb683d2-e398-41da-9f11-5a35f44331d3", "type": "Play", "user": 73838254, "userAgent":
"Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.24) Gecko/20111103 Firefox/3.6.24"}
{"auth": "1495427:3828a4fc", "craetedAt": "2013-05-09T08:00:25Z", "payload": {}, "refId": "7730ee61",
"sessionID": "99f43be9-00ce-434d-801d-fe6a5a737d31", "type": "Play", "user": 21582887, "userAgent":
"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727;
.NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; MS-RTC LM 8;
InfoPath.3)"}

$ grep -v '"payload": {}' bad.json
```

These records all lack content in the payload field, probably because of some logging issue in the system that generated these logs. Without the payload field, these records don't tell you anything useful, so you may as well discard them.

2. Modify your mapper

Open up the `clean_map.py` file and modify it as follows (changes in bold red):

```
#!/usr/bin/python
import dateutil.parser
import json
import sys
from datetime import tzinfo, timedelta, datetime
def main():
   for line in sys.stdin:
      # Correct for double quotes
      data = json.loads(line.replace('""', '"'))
      # Correct for variance in field names
      item_id = 'item_id'
      session_id = 'session_id'
      created_at = 'created_at'
      if 'sessionID' in data:
         session_id = 'sessionID'
      if 'createdAt' in data:
         created_at = 'createdAt'
      elif 'craetedAt' in data:
         created_at = 'craetedAt'
      if 'payload' in data and 'itemId' in data['payload']:
         item_id = 'itemId'
      # Prepare the key
      userid = data['user']
      sessionid = data[session_id]
      timestamp = total_seconds(dateutil.parser.parse(data[created_at]) - EPOCH)
      key = '%s,%10d,%s' % (userid, timestamp, sessionid)
      # Write out the value
      if data['type'] == "Account" and data['payload']['subAction'] == "parentalControls":
         print "%s\tx:%s" % (key, data['payload']['new'])
      elif data['type'] == "Account":
         print "%s\tc:%s" % (key, data['payload']['subAction'])
      elif data['type'] == "AddToQueue":
         print "%s\ta:%s" % (key, data['payload'][item_id])
      elif data['type'] == "Home":
         print "%s\tP:%s" % (key, ",".join(data['payload']['popular']))
         print "%s\tR:%s" % (key, ",".join(data['payload']['recommended']))
         print "%s\tr:%s" % (key, ",".join(data['payload']['recent']))
      elif data['type'] == "Hover":
         print "%s\th:%s" % (key, data['payload'][item_id])
      elif data['type'] == "ItemPage":
         print "%s\ti:%s" % (key, data['payload'][item_id])
      elif data['type'] == "Login":
         print "%s\tL:" % key
      elif data['type'] == "Logout":
         print "%s\tl:" % key
      elif data['type'] == "Play" or \
           data['type'] == "Pause" or \
           data['type'] == "Position" or \
           data['type'] == "Stop" or \
           data['type'] == "Advance" or \
           data['type'] == "Resume":
         if len(data['payload']) > 0:
            print "%s\tp:%s,%s" % (key, data['payload']['marker'], data['payload'][item_id])
      elif data['type'] == "Queue":
         print "%s\tq:" % key
      elif data['type'] == "Rate":
         print "%s\tt:%s,%s" % (key, data['payload'][item_id], data['payload']['rating'])
      elif data['type'] == "Recommendations":
         print "%s\tC:%s" % (key, ",".join(data['payload']['recs']))
      elif data['type'] == "Search":
         print "%s\tS:%s" % (key, ",".join(data['payload']['results']))
      elif data['type'] == "VerifyPassword":
         print "%s\tv:" % key
      elif data['type'] == "WriteReview":
         print "%s\tw:%s,%s,%s" % (key, data['payload'][item_id], data['payload']['rating'],
data['payload']['length'])

"""
Return the number of seconds since the epoch, calculated the hard way.
"""
def total_seconds(td):
   return (td.microseconds + (td.seconds + td.days * 24 * 3600) * 10**6) / 10**6

"""
A constant for 0 time difference
"""
ZERO = timedelta(0)
```

```
"""
A Timezone class for UTC
"""
class UTC(tzinfo):
    def utcoffset(self, dt):
        return ZERO
    def tzname(self, dt):
        return "UTC"
    def dst(self, dt):
        return ZERO

"""
A constant for the beginning of the epoch
"""
EPOCH = datetime(1970,1,1,tzinfo=UTC())
if __name__ == '__main__':
   main()
```

3. Rerun the job

Save `clean_map.py` and try again. Remove the clean directory from the previous attempt and rerun the job:

```
$ hadoop fs -rm -R clean
Deleted clean
$ hadoop jar $STREAMING -mapper clean_map.py -file clean_map.py -reducer clean_reduce.py -file
clean_reduce.py -input data/heckle -input data/jeckle -output clean
packageJobJar: [clean_map.py, clean_reduce.py, /tmp/hadoop-training/hadoop-unjar917675366485832632/]
[] /var/folders/ll/xl67db9x2rs47q5fs4b255rr0000gp/T/streamjob6209378816773863738.jar tmpDir=null
13/11/18 13:30:33 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/11/18 13:30:33 INFO mapred.FileInputFormat: Total input paths to process : 20
13/11/18 13:30:33 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-training/mapred/local]
13/11/18 13:30:33 INFO streaming.StreamJob: Running job: job_201311081846_0012
13/11/18 13:30:33 INFO streaming.StreamJob: To kill this job, run:
13/11/18 13:30:33 INFO streaming.StreamJob: /usr/bin/hadoop job  -
Dmapred.job.tracker=192.168.56.101:8021 -kill job_201311081846_0012
13/11/18 13:30:33 INFO streaming.StreamJob: Tracking URL: http://192.168.56.101:50030/jobdetails.jsp?
jobid=job_201311081846_0012
13/11/18 13:30:34 INFO streaming.StreamJob:  map 0%  reduce 0%
13/11/18 13:30:44 INFO streaming.StreamJob:  map 3%  reduce 0%
…
13/11/18 13:32:59 INFO streaming.StreamJob:  map 100%  reduce 30%
13/11/18 13:33:02 INFO streaming.StreamJob:  map 100%  reduce 74%
13/11/18 13:33:05 INFO streaming.StreamJob:  map 100%  reduce 100%
13/11/18 13:33:09 INFO streaming.StreamJob: Job complete: job_201311081846_0012
13/11/18 13:33:09 INFO streaming.StreamJob: Output: clean
```

This time the entire job succeeds, leaving you with a single, clean, aggregate data file with each record containing a full session.

4. View the data file

To view the data file, execute the following commands:

```
$ hadoop fs -ls -h clean
Found 3 items
-rw-r--r--   1 daniel supergroup          0 2013-11-10 01:49 clean/_SUCCESS
drwxrwxrwx   - daniel supergroup          0 2013-11-10 01:46 clean/_logs
-rw-r--r--   1 daniel supergroup       3.1m 2013-11-10 01:49 clean/part-00000
$ hadoop fs -cat clean/part-00000 | head -1
{"session": "2b5846cb-9cbf-4f92-a1e7-b5349ff08662", "hover": ["16177", "10286", "8565", "10596",
"29609", "13338"], "end": "1368189995", "played": {"16316": "4990"}, "browsed": [], "recommendations":
["13338", "10759", "39122", "26996", "10002", "25224", "6891", "16361", "7489", "16316", "12023",
"25803", "4286e89", "1565", "20435", "10596", "29609", "14528", "6723", "35792e23", "25450",
"10143e155", "10286", "25668", "37307"], "actions": ["login"], "reviewed": {}, "start": "1368189205",
"recommended": ["8565", "10759", "10002", "25803", "10286"], "rated": {}, "user": "10108881",
"searched": [], "popular": ["16177", "26365", "14969", "38420", "7097"], "kid": null, "queued":
["10286", "13338"], "recent": ["18392e39"]}
cat: Unable to write to output stream.
```

The warning you see — `cat: Unable to write to output stream.` — is completely normal when piping through the head command as Hadoop has more data write but is restricted by the head command.

You've now reduced the logs down to under 4MB, which will make the data much easier to manage. Second, you can see from the file contents that your cleaning scripts worked correctly. However, you can't see that the reducer added a tab character to the end of every line, but since it's JSON data, it won't have any effect. We are now ready to move on to solving the challenge problems.