Lecture 17: Hadoop and the mrjob package
Some slides adapted from C. Budak
Recap

Previous lecture: Hadoop/MapReduce framework in general

This lecture: actually doing things

In particular: mrjob Python package

https://pythonhosted.org/mrjob/

Installation: pip install mrjob (or conda, or install from source...)

Recap: Basic concepts

**Mapper**: takes a (key, value) pair as input
   - Outputs zero or more (key, value) pairs
   - Outputs grouped by key

**Combiner**: takes a key and a subset of values for that key as input
   - Outputs zero or more (key, value) pairs
   - Runs after the mapper, only on a slice of the data
   - Must be **idempotent**

**Reducer**: takes a key and all values for that key as input
   - Outputs zero or more (key, value) pairs
Recap: a prototypical MapReduce program

Input: \(<k_1, v_1>\)  
\(\rightarrow\) map  
\(\rightarrow\) combine  
\(\rightarrow\) reduce  
\(\rightarrow\) Output \(<k_3, v_3>\)

Note: this output could be made the input to another MR program.
Recap: Basic concepts

**Step:** One sequence of map, combine, reduce
   All three are optional, but must have at least one!

**Node:** a computing unit (e.g., a server in a rack)

**Job tracker:** a single node in charge of coordinating a Hadoop job
   Assigns tasks to worker nodes

**Worker node:** a node that performs actual computations in Hadoop
   e.g., computes the Map and Reduce functions
Python mrjob package

Developed at Yelp for simplifying/prototyping MapReduce jobs

mrjob acts like a wrapper around Hadoop Streaming
Hadoop Streaming makes Hadoop computing model available to languages other than Java

But mrjob can also be run without a Hadoop instance at all!
  e.g., locally on your machine
Why use `mrjob`?

Fast prototyping
   Can run locally without a Hadoop instance...
   ...but can also run atop Hadoop or Spark

Much simpler interface than Java Hadoop

Sensible error messages
   i.e., usually there’s a Python traceback error if something goes wrong
Because everything runs “in Python”
Basic `mrjob` script

```python
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

```
keith@Steinhaus:~$ cat my_file.txt
Here is a first line.
And here is a second one.
Another line.
The quick brown fox jumps over the lazy dog.
keith@Steinhaus:~$
keith@Steinhaus:~$ python mr_word_count.py my_file.txt
No configs found; falling back on auto-configuration
No configs specified for inline runner
Running step 1 of 1...
Creating temp directory
/tmp/mr_word_count.keith.20171105.022629.949354
Streaming final output from
/tmp/mr_word_count.keith.20171105.022629.949354/output[...
"chars"    103
"lines"    4
"words"    22
Removing temp directory
/tmp/mr_word_count.keith.20171105.022629.949354...
keith@Steinhaus:~$
```
Basic mrjob script

This is a MapReduce job that counts the number of characters, words, and lines in a file.

```
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

Each mrjob program you write requires defining a class, which extends the MRJob class.

These mapper and reducer methods are precisely the Map and Reduce operations in our job. Recall the difference between the `yield` keyword and the `return` keyword.

This if-statement will run precisely when we call this script from the command line.
This is a MapReduce job that counts the number of characters, words, and lines in a file.

from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
Basic `mrjob` script

In `mrjob`, an MRJob object implements one or more steps of a MapReduce program. Recall that a step is a single Map->Reduce->Combine chain. All three are optional, but must have at least one in each step.

```python
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    
    Methods defining the steps go here.

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

If we have more than one step, then we have to do a bit more work... (we'll come back to this)
**Basic mrjob script**

```python
from mrjob.job import MRJob
class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1
    def reducer(self, key, values):
        yield key, sum(values)
if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

This is a MapReduce job that counts the number of characters, words, and lines in a file.

**Warning:** do not forget these two lines, or else your script will not run!
Basic `mrjob` script: recap

```python
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

```bash
keith@Steinhaus:~$ cat my_file.txt
Here is a first line.
And here is a second one.
Another line.
The quick brown fox jumps over the lazy dog.
keith@Steinhaus:~$ python mr_word_count.py my_file.txt
No configs found; falling back on auto-configuration
No configs specified for inline runner
Running step 1 of 1...
Creating temp directory
/tmp/mr_word_count.keith.20171105.022629.949354
Streaming final output from
/tmp/mr_word_count.keith.20171105.022629.949354/output.
..
"chars"    103
"lines"    4
"words"    22
Removing temp directory
/tmp/mr_word_count.keith.20171105.022629.949354...
keith@Steinhaus:~$
```
More complicated jobs: multiple steps

```python
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w']+")

class MRMostUsedWord(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper.get_words,
                   combiner=self.combiner.count_words,
                   reducer=self.reducer.count_words),
            MRStep(reducer=self.reducer.find_max_word)]

    def mapper_get_words(self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner_count_words(self, word, counts):
        # optimization: sum the words we've seen so far
        yield (word, sum(counts))

    def reducer_count_words(self, word, counts):
        # send all (num occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), word)

    # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=count, value=word
        yield max(word_count_pairs)

if __name__ == '__main__':
    MRMostUsedWord.run()
```

keith@Steinhau:~$ python mr_most_common_word.py moby_dick.txt
No configs found; falling back on auto-configuration
No configs specified for inline runner
Running step 1 of 2...
Creating temp directory
/tmp/mr_most_common_word.keith.20171105.032400.702113
Running step 2 of 2...
Streaming final output from
/tmp/mr_most_common_word.keith.20171105.032400.702113/output...
14711    "the"
Removing temp directory
/tmp/mr_most_common_word.keith.20171105.032400.702113...
keith@Steinhau:~$
To have more than one step, we need to override the existing definition of the method steps() in MRJob. The new steps() method must return a list of MRStep objects.

An MRStep object specifies a mapper, combiner and reducer. All three are optional, but must specify at least one.
First step: count words

This pattern should look familiar. It implements word counting.

One key difference, because this reducer output is going to be the input to another step.
Second step: find the largest count.

Note: word_count_pairs is like a list of pairs. Refer to how Python max works on a list of tuples.
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"\w+"
)

class MRMostUsedWord(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_words,
                   combiner=self.combiner_count_words,
                   reducer=self.reducer_count_words),
            MRStep(reducer=self.reducer_find_max_word)]

    def mapper_get_words(self, self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner_count_words(self, self, word, counts):
        # optimization: sum the words we've seen so far
        yield (word, sum(counts))

    def reducer_count_words(self, self, word, counts):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), word)

        # discard the key; it is just None
    def reducer_find_max_word(self, self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)

if __name__ == '_main_':
    MRMostUsedWord.run()
MRJob.{mapper, combiner, reducer}

**MRJob.mapper(key, value)**

- **key** – parsed from input; **value** – parsed from input.
  - Yields zero or more tuples of (out_key, out_value).

**MRJob.combiner(key, values)**

- **key** – yielded by mapper; **value** – generator yielding all values from node corresponding to key.
  - Yields one or more tuples of (out_key, out_value)

**MRJob.reducer(key, values)**

- **key** – key yielded by mapper; **value** – generator yielding all values from corresponding to key.
  - Yields one or more tuples of (out_key, out_value)

Details: [https://pythonhosted.org/mrjob/job.html](https://pythonhosted.org/mrjob/job.html)
More complicated reducers: Python’s \texttt{reduce}

So far our reducers have used Python built-in functions \texttt{sum} and \texttt{max}

```python
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

```python
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w]+")

class MRMostUsedWord(MRJob):
    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield word, 1

    def reducer_count_words(self, word, counts):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), word)

    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=count, value=word
        yield max(word_count_pairs)

if __name__ == '__main__':
    MRMostUsedWord.run()
```
More complicated reducers: Python’s `reduce`

So far our reducers have used Python built-in functions `sum` and `max`.

What if I want to multiply the values instead of `sum`?
Python does not have `product()` function analogous to `sum()`...

What if my values aren’t numbers, but I have a sum defined on them?
e.g., tuples representing vectors
Want \((a, b) + (x, y) = (a+x, b+y)\), but tuples don’t support addition.

**Solution:** use `functools.reduce`
More complicated reducers: Python’s \texttt{reduce}

Using \texttt{reduce} and \texttt{lambda}, we can get just about any reducer we want.

Note: this example was run in Python 2. You'll need to import \texttt{functools} to do this.
Running `mrjob` on a Hadoop cluster

We’ve already seen how to run `mrjob` from the command line. Previous examples emulated Hadoop But no actual Hadoop instance was running!

That’s fine for prototyping and testing…

…but how do I actually run it on my Hadoop cluster? E.g., on Fladoop

Open a terminal if you’d like to follow along.
Step 1: Moving your mrjob script to the grid

keith@Steinhaus:~/mrjob_demo$ ls
moby_dick.txt   mr_most_common_word.py my_file.txt
mr_bigproduct.py mr_word_count.py   numlist.txt

Here I have downloaded the mrjob demo zip archive from the website, unzipped it, and cd (changed directory) into the resulting directory.
Step 1: Moving your mrjob script to the grid

Here I have downloaded the mrjob demo zip archive from the website, unzipped it, and cd (changed directory) into the resulting directory.

We can tell from the prompt what my username is, what machine I’m on, and where I am in the directory structure.
Step 1: Moving your mrjob script to the grid

I need to get this file from my laptop (the “local” machine) to the flux hadoop cluster (the “remote” machine).

```python
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```
Step 1: Moving your mrjob script to the grid

keith@Steinhaus:~/mrjob_demo$ ls
moby_dick.txt   mr_most_common_word.py  my_file.txt
mr_bigproduct.py   mr_word_count.py   numlist.txt
keith@Steinhaus:~/mrjob_demo$ scp mr_word_count.py klevin@flux-hadoop-login.arc-ts.umich.edu:/mr_word_count.py

Copy the local file mr_word_count.py...

```python
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```
Step 1: Moving your `mrjob` script to the grid

keith@Steinhaus:~/mrjob_demo$ ls
moby_dick.txt  mr_most_common_word.py my_file.txt
mr_bigproduct.py  mr_word_count.py  numlist.txt

keith@Steinhaus:~/mrjob_demo$ scp mr_word_count.py
klevin@flux-hadoop-login.arc-ts.umich.edu:~/mr_word_count.py

Copy the local file `mr_word_count.py`...

...to the remote machine, and save it with the same name, in the home directory.

```python
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```
Step 1: Moving your `mrjob` script to the grid

```bash
keith@Steinhaus:~/mrjob_demo$ ls
moby_dick.txt  mr_most_common_word.py my_file.txt
mr_bigproduct.py  mr_word_count.py  numlist.txt
keith@Steinhaus:~/mrjob_demo$ scp mr_word_count.py
klevin@flux-hadoop-login.arc-ts.umich.edu:~/mr_word_count.py
[...prompted for authentication...]
```

I hit enter and I am asked to give my password and 2-factor authentication. Once I authenticate successfully, the file is copied, and `scp` shows its progress (percentage, file size, rate of copying, total time).

```python
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```
Step 1: Moving your `mrjob` script to the grid

keith@Steinhaus:~/mrjob_demo$ ssh klevin@flux-hadoop-login.arc-ts.umich.edu
[...authentication and greeting from the flux-hadoop cluster...]
[klevin@flux-hadoop-login2 ~]$  

Now I'll `ssh` to the flux-hadoop cluster. Once I authenticate successfully I get a command line prompt. Notice that from the prompt I can see that I am now signed on to a different machine (`flux-hadoop-login2`), and I am currently in the home (`~`) directory on that machine.
Step 1: Moving your `mrjob` script to the grid

keith@Steinhaus:~/mrjob_demo$ ssh klevin@flux-hadoop-login.arc-ts.umich.edu
[...authentication and greeting from the flux-hadoop cluster...]
[klevin@flux-hadoop-login2 ~]$ ls
ASEOOS  hotelling_tsquared.m  mr_word_count.py  scripts  cmdfiles
matlab  multinet  stats507w19  data  matlabdata

`ls` lists the contents of the current directory, and we see that `mr_word_count.py` is there, as it should be.
Step 1: Moving your mrjob script to the grid

Just to be sure, let's look at the first few lines using head. Comparing with our original file, it looks like it worked!
Running **mrjob** on Fladoop

```
[klevin@flux-hadoop-login2]$ python mr_word_count.py -r hadoop
hdfs:///var/stats507w19/moby_dick.txt
[...output redacted...]
Copying local files into
hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680/files/

[...Hadoop information redacted...]
Counters from step 1:
  (no counters found)
Streaming final output from
hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680/output
"chars"    1230866
"lines"    22614
"words"    215717
removing tmp directory /tmp/mr_word_count.klevin.20171113.145355.093680
deleting hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680 from HDFS
[klevin@flux-hadoop-login2]$
```
Running mrjob on Fladoop

[klevin@flux-hadoop-login2]$ python mr_word_count.py -r hadoop hdfs:///var/stats507w19/moby_dick.txt
[...output redacted...]
Copying local files into
hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680/files/
[...Hadoop information redacted...]
Counters from step 1:
(no counters found)
Streaming final output from
hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680/output
"chars"    1230866
"lines"    22614
"words"    215717
removing tmp directory /tmp/mr_word_count.klevin.20171113.145355.093680
deleting hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680 from HDFS
[klevin@flux-hadoop-login2]$
Running mrjob on Fladoop

```
$ python mr_word_count.py -r hadoop hdfs:///var/stats507w19/moby_dick.txt

[...output redacted...]

Copying local files into hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680/files/

[...Hadoop information redacted...]

Counters from step 1:
(no counters found)

Streaming final output from hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680/output

"chars"    1230866
"lines"    22614
"words"    215717

removing tmp directory /tmp/mr_word_count.klevin.20171113.145355.093680

deleting hdfs:///user/klevin/tmp/mrjob/mr_word_count.klevin.20171113.145355.093680 from HDFS
```

This is a path to a file on HDFS, not on the local file system!

hdfs:///var/stats507w19 is a directory created specifically for our class. Some problems in the homework will ask you to use files that I’ve put here.
Running `mrjob` on Fladoop: redirecting output

Here I'm running the same command, but I'm redirecting the output to the file `melville.txt`, instead of letting the output get written to the terminal.
Running **mrjob** on Fladoop: redirecting output

Notice that the messages on the screen look basically the same as before, except we never see the “chars”, “words” or “lines” counts get written out. That’s because we’ve redirected **stdout** of this process to the file *mellville.txt*. The result is that only **stderr** (i.e., errors, warnings and information for the user) is written to the terminal.
Running `mrjob` on Fladoop: redirecting output

```
[kleven@flux-hadoop-login2 ~]$ python mr_word_count.py -r hadoop
hdfs:///var/stats507w19/moby_dick.txt > melville.txt
[...output redacted...]
job output is in
hdfs:///user/kleven/tmp/mrjob/mr_word_count.kleven.20190320.145525.603643/output
Streaming final output from
hdfs:///user/kleven/tmp/mrjob/mr_word_count.kleven.20190320.145525.603643/output...
Removing HDFS temp directory
hdfs:///user/kleven/tmp/mrjob/mr_word_count.kleven.20190320.145525.603643...
Removing temp directory /tmp/mr_word_count.kleven.20190320.145525.603643...
[kleven@flux-hadoop-login2 ~]$ cat melville.txt
"chars"    1215296
"lines"    22614
"words"    215717
[kleven@flux-hadoop-login2 ~]$ cat melville.txt
...and catting `melville.txt` shows that it does indeed contain the counts as expected.
```
Instead of copying from my machine to the cluster, now I'm doing the opposite. I'm copying the file `melville.txt` from my home directory on the flux hadoop cluster to the current directory.

Recall that the dot (.) refers to the current directory, so this command basically says copy the file `melville.txt` from the cluster and save it (with the same name) right here in the current directory (i.e., `mrjob_demo`).
Running `mrjob` on Fladoop: retrieving files

Once I hit enter I have to authenticate and wait for the file transfer to complete...
Running `mrjob` on Fladoop: retrieving files

And notice that `melville.txt` is now here on my local machine.
Running mrjob on Fladoop: retrieving files

keith@Steinhaus:~/mrjob_demo$ scp klevin@flux-hadoop-login.arc-ts.umich.edu:/melville.txt .
[...authentication...]
melville.txt

keith@Steinhaus:~/mrjob_demo$ ls
melville.txt    mr_most_common_word.py    numlist.txt
moby_dick.txt   mr_word_count.py
mr_bigproduct.py my_file.txt

keith@Steinhaus:~/mrjob_demo$ cat melville.txt

"chars"     1215296
"lines"     22614
"words"     215717

...and if we cat it, it looks like we expected.
HDFS is a separate file system

Local file system
Accessible via ls, mv, cp, cat...

/home/klevin
/home/klevin/stats507
/home/klevin/myfile.txt
(and lots of other files…)

Hadoop distributed file system
Accessible via hdfs...

/var/stats507w19
/var/stats507w19/fof
/var/stats507w19/populations_small.txt
(and lots of other files…)

Shell provides commands for moving files around, listing files, creating new files, etc. But if you try to use these commands to do things on HDFS... no dice!

Hadoop has a special command line tool for dealing with HDFS, called hdfs
Basics of hdfs

Usage: hdfs dfs [options] COMMAND [arguments]

Where COMMAND is, for example:
  -ls, -mv, -cat, -cp, -put, -tail
All of these should be pretty self-explanatory except -put
For your homework, you should only need -cat and perhaps -cp/-put

Getting help:

[klevin@flux-hadoop-login1 mrjob_demo]$ hdfs dfs -help
[...tons of help prints to shell...]
[klevin@flux-hadoop-login1 mrjob_demo]$ hdfs dfs -help | less
**hdfs** essentially replicates shell command line

```
[klever@flux-hadoop-login2 mrjob_demo]$ cat demo_file.txt
This is just a demo file.
Normally, a file this small would have no reason to be on HDFS.
[klever@flux-hadoop-login2 mrjob_demo]$ hdfs dfs -put demo_file.txt
hdfs:/var/stats507w19/demo_file.txt
[klever@flux-hadoop-login2 mrjob_demo]$ hdfs dfs -cat
hdfs:/var/stats507w19/demo_file.txt
This is just a demo file.
Normally, a file this small would have no reason to be on HDFS.
[klever@flux-hadoop-login2 mrjob_demo]$
```

**Important points:**

- hdfs:/var and /var are different directories on different file systems.
- hdfs **dfs** -CMD because hdfs supports lots of other stuff, too.
- Don’t forget a hyphen before your command! **-cat**, not **cat**
To see all our HDFS files

```
[klevin@flux-hadoop-login2 mrjob_demo]$ hdfs dfs -ls hdfs:/var/stats507w19
Found 8 items
-rw-r--r--   3 klevin stats507w19admins 1291775 2019-03-12 12:47 hdfs:///var/stats507w19/darwin.txt
-rw-r--r--   3 klevin stats507w19admins 90 2019-03-12 10:30 hdfs:///var/stats507w19/demo_file.txt
drwxr-xr-x   - klevin stats507w19admins  0 2018-04-04 20:56 hdfs:///var/stats507w19/fof
-rw-r--r--   3 klevin stats507w19admins 1276097 2019-03-12 10:35 hdfs:///var/stats507w19/moby_dick.txt
-rw-r--r--   3 klevin stats507w19admins 48 2019-03-12 10:59 hdfs:///var/stats507w19/numbers.txt
-rw-r--r--   3 klevin stats507w19admins 48 2019-03-12 11:30 hdfs:///var/stats507w19/numbers_weird.txt
-rw-r--r--   3 klevin stats507w19admins  0 2019-03-13 12:35 hdfs:///var/stats507w19/populations_large.txt
-rw-r--r--   3 klevin stats507w19admins 251 2019-03-12 11:22 hdfs:///var/stats507w19/scientists.txt
```

You’ll use some of these files in your homework.
mrjob hides complexity of MapReduce

We need only define mapper, reducer, combiner

Package handles everything else
   Most importantly, interacting with Hadoop

But mrjob does provide powerful tools for specifying Hadoop configuration
   [https://pythonhosted.org/mrjob/guides/configs-basics.html](https://pythonhosted.org/mrjob/guides/configs-basics.html)

You don’t have to worry about any of this in this course, but you should be aware of it in case you need it in the future.
**mrjob: protocols**

mrjob assumes that all data is “newline-delimited bytes”

That is, newlines separate lines of input

Each line is a single unit to be processed in isolation

(e.g., a line of words to count, an entry in a database, etc)

mrjob handles inputs and outputs via **protocols**

**Protocol** is an object that has `read()` and `write()` methods

- `read()`: convert bytes to (key,value) pairs
- `write()`: convert (key,value) pairs to bytes
mrjob: protocols

Controlled by setting three variables in config file `mrjob.conf`:

- `INPUT_PROTOCOL`
- `INTERNAL_PROTOCOL`
- `OUTPUT_PROTOCOL`

Defaults:

```python
INPUT_PROTOCOL = mrjob.protocol.RawValueProtocol
INTERNAL_PROTOCOL = mrjob.protocol.JSONProtocol
OUTPUT_PROTOCOL = mrjob.protocol.JSONProtocol
```

Again, you don’t have to worry about this in this course, but you should be aware of it.

Data passed around internally via JSON. This is precisely the kind of thing that JSON is good for.