16.2.2

mrjob Library

To make sure you can answer your client's questions, you and Jennifer decide to get a little practice using MapReduce via Python. For this you'll be using the mrjob library.

Python has a library called mrjob, which stands for "MapReduce job" and it can help us practice MapReduce outside of the Hadoop ecosystem. There is very little setup, and we can run the library from any computer with Python installed.

Let's start by installing mrjob in our Python environment. Open the terminal and activate your PythonData environment. Once activated, run the following command in your terminal:

```
$ pip install MRJob
```

NOTE

If `pip install MRJob` does not work, try `pip3 install MRJob`.

Next, open your text editor and create a new Python file named `bacon_counter.py`. In the file, enter the following code to import mrjob:

```
from mrjob.job import MRJob
```

Create a class called `Bacon_count`, which inherits, or takes properties, from the `MRJob` class. We create this class to be called to run the full MapReduce job with `MRJob`:

```
class Bacon_count(MRJob):
```

Next, create a `mapper()` function that will take `(self, _, line)` as parameters. The `mapper()` function will assign the input to key-value pairs:

```
def mapper(self, _, line):
```

The second parameter (here using an underscore (_), explained next) allows methods to be mapped together. Since we are not chaining anything together, we use the Python convention of an underscore to indicate that we won't use this parameter. The `line` parameter will be the line of text taken from the raw input file.

The function will loop through each word in the line of text, as described below:

1. Call the `split()` method on each line to break the text into a list of words.
2. Each word will convert to lowercase.
3. If the words match the search word "bacon," a key-value pair will show as `yield "bacon", 1`.
4. When you call a function with yield it returns what is called a generator object. A generator is an iterator like a list, however unlike a list the contents are not stored in memory, useful for large files. When yield is called the function is suspended and returns a value. A generator won't return another value until `next()` is called, which is something that mrJobs calls a number of times till it is done. So, for a yield, each time the word "bacon" appears, mrJobs returns `"bacon", 1`. If "bacon" appears three times, then an output of `"bacon", 1` would be produced three times.

NOTE

For more information on generators and yield checkout this **realpython.com (https://realpython.com/introduction-to-python-generators/#understanding-the-python-yield-statement)** article on how to use generators and yield in python.

```
for word in line.split():
    if word.lower() == "bacon":
        yield "bacon", 1
```

There's a shuffle step that occurs after the mapper. There is no code written for this step, and it occurs because the class inherits from the mrjob library. This shuffle step organizes the key-value pairs so that there's only one key for

each unique key, and combines the values into a list.

The reducer function might not look like it's doing as much as the mapper function, but it's just as important. The reducer function takes three parameters: `self, key, and values`:

1. The `self` parameter is used in Python to represent the instance of the class.
2. The `key` parameter represents the key of the key-value pair created in the mapper function. In this example, the key is "bacon."
3. The `values` parameter is a list of values created in the mapper function. We want to sum all of these values. Recall that from the mapper function the yield was used to produce multiple outputs. With the reducer we'll produce the key and sum of all the values assigned with it:

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

The final bit of code, shown below, is conventional Python code for running the program:

```
if __name__ == "__main__":
   Bacon_count.run()
```

All together your code should look like the following:

```
from mrjob.job import MRJob
class Bacon_count(MRJob):
   def mapper(self, _, line):
       for word in line.split():
           if word.lower() == "bacon":
               yield "bacon", 1

   def reducer(self, key, values):
       yield key, sum(values)
if __name__ == "__main__":
   Bacon_count.run()
```

You might have noticed that nowhere in the code is a file imported or opened. The mrjob library works by reading in a file passed to it in the terminal.

Let's use a file full of random words, **input.txt (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_16/input.txt)** we just created.

Run the following code in the terminal:

```
$ python bacon_counter.py input.txt
```

The output might seem a little confusing at first, but all that the extra output is stating, is that it created a temporary folder, outputted the results to that folder, and then removed the folder. In the middle of all that, you'll find the result of the `MRJob`.

As mentioned, the mrjob library won't be used in production anywhere, as we'll employ better libraries for that purpose. It's good to practice with smaller files and to try predicting the result, and then confirm with mrjob.

SKILL DRILL

Give it a shot! Create your own sample text file and see if you can find the number of times different words appear.