Problem Set 5: MapReduce on Subway Data

1 - Ridership per Station:

Riders_per_station_mapper.py

```
import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

def mapper():
"""
    The input to this mapper will be the final Subway-MTA dataset, the same as
    in the previous exercise.  You can check out the csv and its structure below:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

    For each line of input, the mapper output should PRINT (not return) the UNIT as
    the key, the number of ENTRIESn_hourly as the value, and separate the key and
    the value by a tab. For example: 'R002\t105105.0'

    Since you are printing the output of your program, printing a debug
    statement will interfere with the operation of the grader. Instead,
    use the logging module, which we've configured to log to a file printed
    when you click "Test Run". For example:
    logging.info("My debugging message")
    Note that, unlike print, logging.info will take only a single argument.
    So logging.info("my message") will work, but logging.info("my","message") will not.

    The logging module can be used to give you more control over your debugging
    or other messages than you can get by printing them. In this exercise, print
    statements from your mapper will go to your reducer, and print statements
    from your reducer will be considered your final output. By contrast, messages
    logged via the loggers we configured will be saved to two files, one
    for the mapper and one for the reducer. If you click "Test Run", then we
    will show the contents of those files once your program has finished running.
    The logging module also has other capabilities; see
    https://docs.python.org/2/library/logging.html for more information.
    """
```

```
    for line in sys.stdin:

    # your code here
    data = line.strip().split(',');

        if len(data) !=22 or data[6] == "ENTRIESn_hourly":
            continue
        else:
            print '{0}\t{1}'.format(data[1], data[6])
            logging.info("{0}\t{1}".format(data[1], data[6]))
mapper()
```

Rider_per_station_reducer.py

```
import sys
import logging

from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
        level=logging.INFO, filemode='w')

def reducer():
  '''
  Given the output of the mapper for this exercise, the reducer should PRINT
  (not return) one line per UNIT along with the total number of ENTRIESn_hourly
  over the course of May (which is the duration of our data), separated by a tab.
  An example output row from the reducer might look like this: 'R001\t500625.0'

  You can assume that the input to the reducer is sorted such that all rows
  corresponding to a particular UNIT are grouped together.

  Since you are printing the output of your program, printing a debug
  statement will interfere with the operation of the grader. Instead,
  use the logging module, which we've configured to log to a file printed
  when you click "Test Run". For example:
  logging.info("My debugging message")
  Note that, unlike print, logging.info will take only a single argument.
  So logging.info("my message") will work, but logging.info("my","message") will not.
  '''
    entries = 0
    old_key = None
    for line in sys.stdin:
```

```
                # your code here
                data = line.strip().split("\t")
                 if len(data) !=2:
                        continue

                key, count = data
                if old_key and old_key != key:
                        print "{0}\t{1}".format(old_key,entries)
                        entries = 0
                old_key = key
                entries += float(count)
        if old_key != None:
                print "{0}\t{1}".format(old_key, entries)
reducer()
```

```
Your program worked correctly! Make sure to click 'Submit' before moving on.
```

2 - Ridership by Weather Type:

Ridership_by_weather_mapper.py

```
import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                level=logging.INFO, filemode='w')

def mapper():
    '''
    For this exercise, compute the average value of the ENTRIESn_hourly column
    for different weather types. Weather type will be defined based on the
    combination of the columns fog and rain (which are boolean values).
    For example, one output of our reducer would be the average hourly entries
    across all hours when it was raining but not foggy.

    Each line of input will be a row from our final Subway-MTA dataset in csv format.
    You can check out the input csv file and its structure below:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

    Note that this is a comma-separated file.
```

```
This mapper should PRINT (not return) the weather type as the key (use the
given helper function to format the weather type correctly) and the number in
the ENTRIESn_hourly column as the value. They should be separated by a tab.
For example: 'fog-norain\t12345'

Since you are printing the output of your program, printing a debug
statement will interfere with the operation of the grader. Instead,
use the logging module, which we've configured to log to a file printed
when you click "Test Run". For example:
logging.info("My debugging message")
Note that, unlike print, logging.info will take only a single argument.
So logging.info("my message") will work, but logging.info("my","message") will not.
'''
# Takes in variables indicating whether it is foggy and/or rainy and
# returns a formatted key that you should output.  The variables passed in
# can be booleans, ints (0 for false and 1 for true) or floats (0.0 for
# false and 1.0 for true), but the strings '0.0' and '1.0' will not work,
# so make sure you convert these values to an appropriate type before
# calling the function.
    def format_key(fog, rain):
        return '{}fog-{}rain'.format('' if fog else 'no','' if rain else 'no')


    for line in sys.stdin:
        data = line.strip().split(',');

        if len(data) !=22 or data[6] == "ENTRIESn_hourly":
            continue
        else:
            print "{0}\t{1}".format(format_key(float(data[14]),float(data[15])), data[6])
            logging.info("{0}\t{1}".format(format_key(float(data[14]),float(data[15])),
data[6]))
mapper()
```

ridership_by_weather_reducer.py

```
import sys
import logging

from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
        level=logging.INFO, filemode='w')
```

```
def reducer():
    '''
    Given the output of the mapper for this assignment, the reducer should
    print one row per weather type, along with the average value of
    ENTRIESn_hourly for that weather type, separated by a tab. You can assume
    that the input to the reducer will be sorted by weather type, such that all
    entries corresponding to a given weather type will be grouped together.

    In order to compute the average value of ENTRIESn_hourly, you'll need to
    keep track of both the total riders per weather type and the number of
    hours with that weather type. That's why we've initialized the variable
    riders and num_hours below. Feel free to use a different data structure in
    your solution, though.

    An example output row might look like this:
    'fog-norain\t1105.32467557'

    Since you are printing the output of your program, printing a debug
    statement will interfere with the operation of the grader. Instead,
    use the logging module, which we've configured to log to a file printed
    when you click "Test Run". For example:
    logging.info("My debugging message")
    Note that, unlike print, logging.info will take only a single argument.
    So logging.info("my message") will work, but logging.info("my","message") will not.
    '''
        entries = 0.0
        avg = 0.0
        num  = 0
        old_key = None

        for line in sys.stdin:
            data = line.strip().split("\t")
             if len(data) !=2:
                continue
            this_key, count = data

            if old_key and old_key != this_key:
                print "{0}\t{1}".format(old_key,avg)
                entries = 0
                num = 0
            old_key = this_key
            entries += float(count)
```

```
            num += 1
            avg =  entries / num

    if old_key != None:
            print "{0}\t{1}".format(old_key, avg)
            logging.info("{0}\t{1}".format(old_key, avg))
```

reducer()

```
This is the logging output from your reducer:
nofog-rain      1098.95330076


Your program worked correctly! Make sure to click 'Submit' before moving on.


Your program produced the following output:
fog-norain      1315.57980681
fog-rain1115.13151799
nofog-norain    1078.54679697
nofog-rain      1098.95330076
```

3 - Busiest Hour:

Busiest_hour_mapper.py

```
import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
            level=logging.INFO, filemode='w')

def mapper():
    """
    In this exercise, for each turnstile unit, you will determine the date and time
    (in the span of this data set) at which the most people entered through the unit.

    The input to the mapper will be the final Subway-MTA dataset, the same as
```

in the previous exercise. You can check out the csv and its structure below:
https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

For each line, the mapper should return the UNIT, ENTRIESn_hourly, DATEn, and
TIMEn columns, separated by tabs. For example:
'R001\t100000.0\t2011-05-01\t01:00:00'

Since you are printing the output of your program, printing a debug
statement will interfere with the operation of the grader. Instead,
use the logging module, which we've configured to log to a file printed
when you click "Test Run". For example:
logging.info("My debugging message")
Note that, unlike print, logging.info will take only a single argument.
So logging.info("my message") will work, but logging.info("my","message") will not.
"""

```
for line in sys.stdin:
        data = line.strip().split(",")
         if len(data) == 22 and data[6] == 'ENTRIESn_hourly':
                continue

        print "{0}\t{1}\t{2}\t{3}".format(data[1],data[6],data[2],data[3])
```

mapper()

busiest_hour_reducer.py

```
import sys
import logging

from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
            level=logging.INFO, filemode='w')

def reducer():
```

'''
Write a reducer that will compute the busiest date and time (that is, the
date and time with the most entries) for each turnstile unit. Ties should
be broken in favor of datetimes that are later on in the month of May. You
may assume that the contents of the reducer will be sorted so that all entries
corresponding to a given UNIT will be grouped together.

The reducer should print its output with the UNIT name, the datetime (which

is the DATEn followed by the TIMEn column, separated by a single space), and
the number of entries at this datetime, separated by tabs.

For example, the output of the reducer should look like this:

```
R001    2011-05-11 17:00:00     31213.0
R002  2011-05-12 21:00:00       4295.0
R003  2011-05-05 12:00:00       995.0
R004  2011-05-12 12:00:00       2318.0
R005  2011-05-10 12:00:00       2705.0
R006  2011-05-25 12:00:00       2784.0
R007  2011-05-10 12:00:00       1763.0
R008  2011-05-12 12:00:00       1724.0
R009  2011-05-05 12:00:00       1230.0
R010  2011-05-09 18:00:00       30916.0
...
...
```

Since you are printing the output of your program, printing a debug
statement will interfere with the operation of the grader. Instead,
use the logging module, which we've configured to log to a file printed
when you click "Test Run". For example:
logging.info("My debugging message")
Note that, unlike print, logging.info will take only a single argument.
So logging.info("my message") will work, but logging.info("my","message") will not.
'''

```
max_entries = 0.0
old_key = None
datetime = ''

for line in sys.stdin:

        data = line.strip().split('\t')

        if len(data) != 4:
                continue

        this_key, count, date, time  = data
        count = float(count)

        if old_key and old_key != this_key:
                print "{0}\t{1}\t{2}".format(old_key, datetime, max_entries)
                max_entries = 0
```

```
                datetime = ''

            old_key = this_key
             if count >= max_entries:
                max_entries = count
                datetime = str(date) + ' ' + str(time)

    if old_key != None:
            print "{0}\t{1}\t{2}".format(old_key, datetime, max_entries)
            logging.info("{0}\t{1}\t{2}".format(old_key, datetime, max_entries))
reducer()
```

```
This is the logging output from your mapper:


This is the logging output from your reducer:
R552    2011-05-20 21:54:55      2917.0


Your program worked correctly! Make sure to click 'Submit' before moving on.
```