**NY Parking Violations**

The NYC Department of Finance collects data on every parking ticket issued in NYC ( 10M per year!). This data is made publicly available to aid in ticket resolution and to guide policy- makers.

1. • When are tickets most likely to be issued?
2. • What are the most common years and types of cars to be ticketed?
3. • Where are tickets most commonly issued?
4. • Which color of the vehicle is most likely to get a ticket?

**1. When are tickets most likely to be issued?**

**Mapper :** Using regex we have filtered the data for the violation time which has A/P after the given time. Setting violation time as key and assigning a value of count 1 to corresponding each entry.

**import sys**

**import re**

**rule = re.compile('[^\s]{4}[A|P]**

**for line in sys.stdin:**

**line=line.strip(',')**

**vtime = line.split(",")[20]**

**match = rule.search(vtime)**

**if match:**

**vtime=vtime[:2]+vtime[-1:]# Only printing the Violation time**

**print('%s\t%s' % (vtime, '1'))**

**Reducer:** Passing the output from mapper which has only values for the violation time as key and counter as value. Sorting all the values with the same key and combining them by adding their counters gives us the final count for each key value.

**import sys**

**from operator import itemgetter**

**dict\_vtime\_count = {}**

**for line in sys.stdin:**

**vtime, count = line.split('\t',1)**

**try:**

**count = int(count)**

**dict\_vtime\_count[vtime] = dict\_vtime\_count.get(vtime, 0) + count**

**except ValueError:**

**pass**

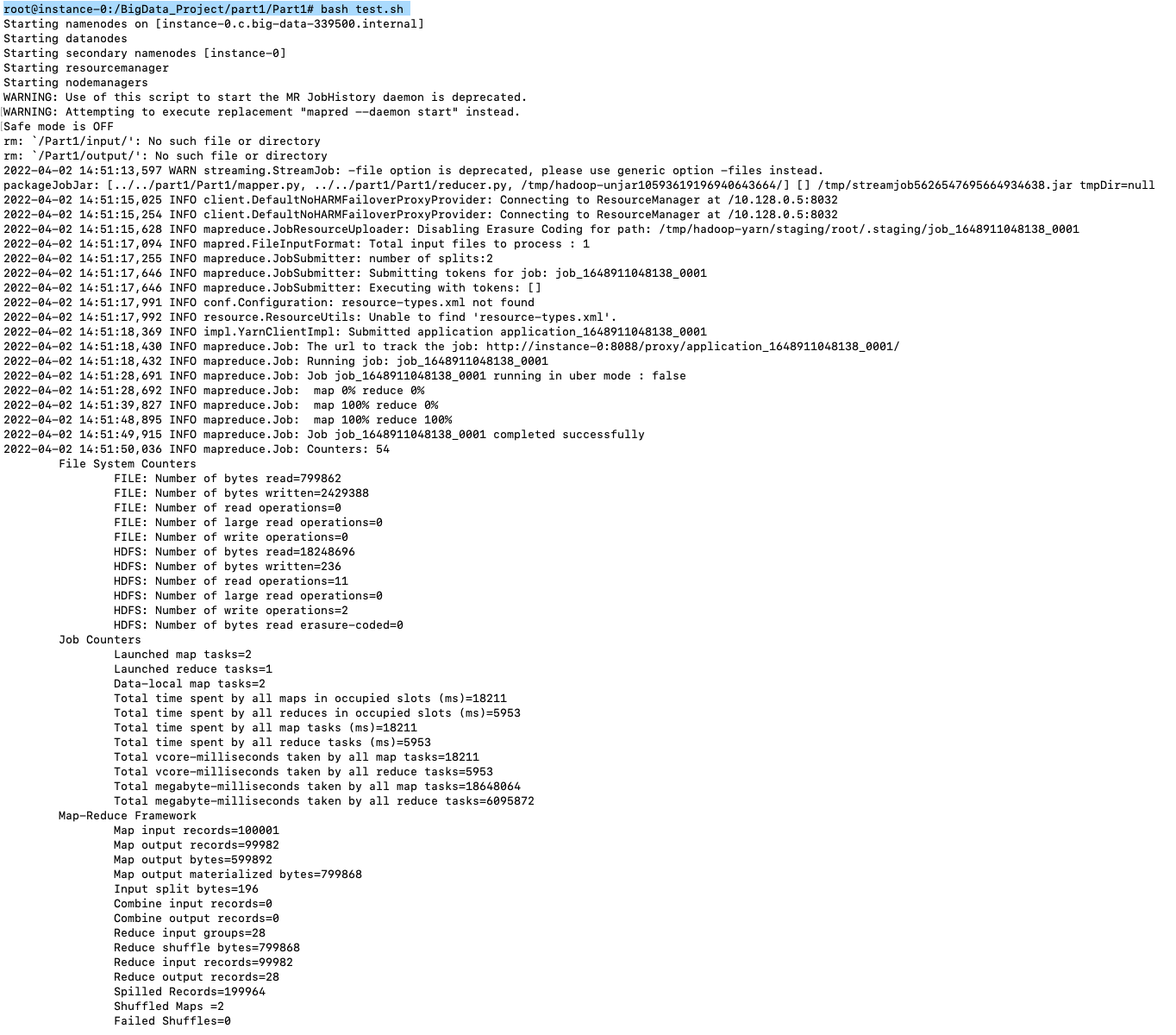
**sorted\_dict\_vtime\_count = sorted(dict\_vtime\_count.items(),**

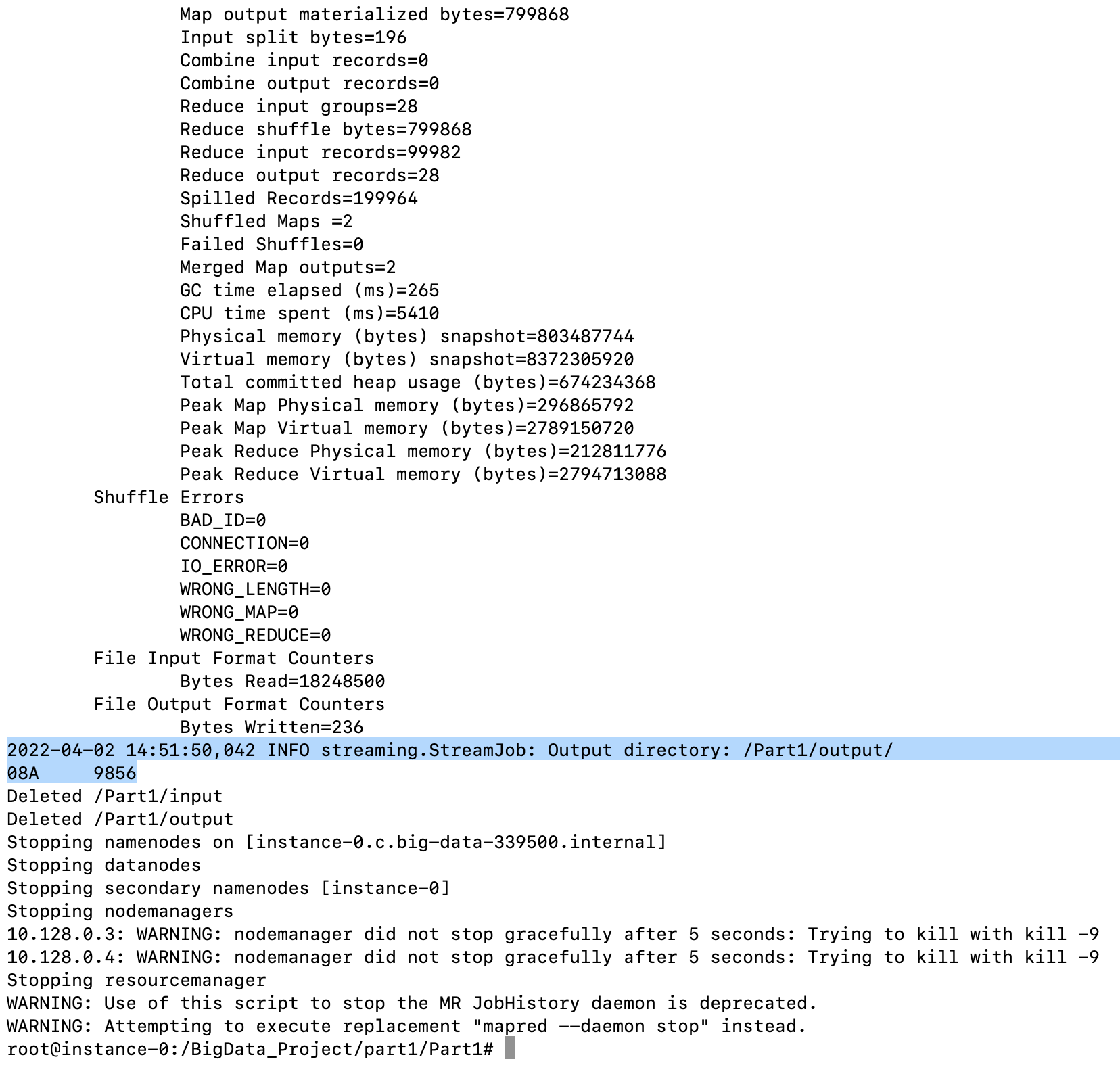
**key=itemgetter(1))[::-1]**

**for vtime, count in sorted\_dict\_vtime\_count:**

**print ('%s\t%s' % (vtime, count))**

**Test.sh:** Only output the key value with maximum count value using head -1 in tesh.sh file on hadoop.





**2. What are the most common years and types of cars to be ticketed?**

**Mapper** **:** Using regex we have filtered the data where car details have alphabetical name and year has numeric value. Setting car description (**car\_type+' '+year**) as key with value value 1 as counter.

**import sys**

**import re**

**rule = re.compile('[A-Z]\s[^\s]{3}')**

**for line in sys.stdin:**

**line=line.strip(',')**

**car\_type= line.split(",")[7]**

**year= line.split(",")[36]**

**car\_descrp=car\_type+' '+year**

**match = rule.search(car\_descrp)**

**if match:**

**print('%s\t%s' % (car\_descrp, '1'))**

**Reducer:**  Passing the output from mapper which has only values for the car description as key and counter as value. Sorting all the values with the same key and combining them by adding their counters gives us the final count for each key value.

**import sys**

**from operator import itemgetter**

**dic\_car\_count = {}**

**for line in sys.stdin:**

**car\_descp, count = line.split('\t',1)**

**try:**

**count = int(count)**

**dic\_car\_count [car\_descp] = dic\_car\_count .get(car\_descp, 0) + count**

**except ValueError:**

**pass**

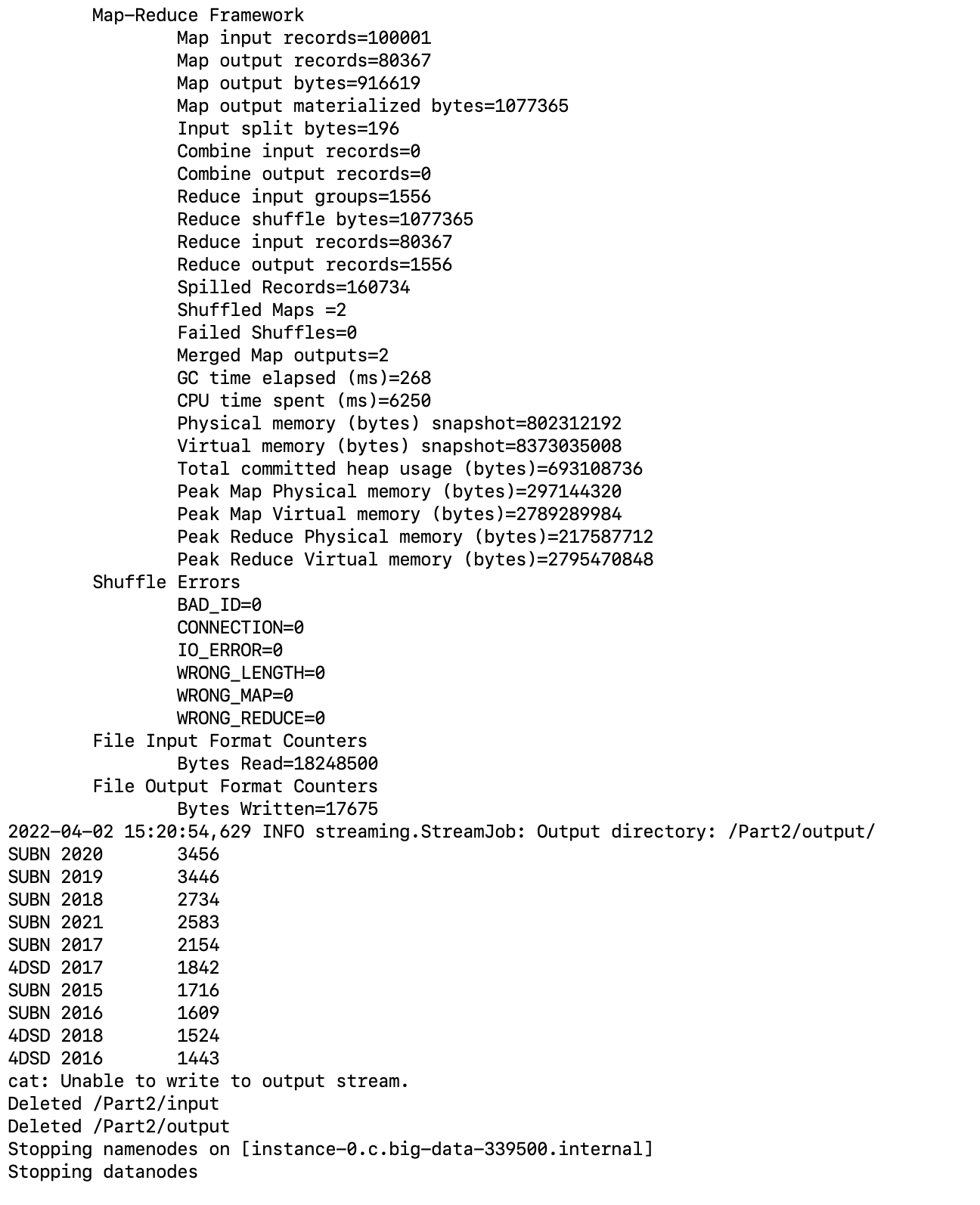
**sorted\_dict\_vtime\_count = sorted(dic\_car\_count .items(), key=itemgetter(1))[::-1]**

**for car\_descp, count in sorted\_dict\_vtime\_count:**

**print ('%s\t%s' % (car\_descp, count))**

**Test.sh:** Only output the key value with maximum count value using head -10 in tesh.sh file on hadoop. It shows **SUBN 2020** has the maximum number of tickets issued.





**3. Where are tickets most commonly issued?**

**Mapper** **:** Using regex we have filtering the data where streets have alpha numeric names and setting it as key of value value 1 as counter.

**import sys**

**import re**

**rule = re.compile('[A-Za-z0-9|/|-]')**

**for line in sys.stdin:**

**line=line.strip(',')**

**street= line.split(",")[25]**

**match = rule.search(street)**

**if match:**

**print('%s\t%s' % ( street, '1'))**

**Reducer:**  Passing the output from mapper which has only values for the street name as key and counter as value. Sorting all the values with the same key and combining them by adding their counters gives us the final count for each key value.

**import sys**

**from operator import itemgetter**

**dic\_street\_count = {}**

**for line in sys.stdin:**

**street, count = line.split('\t',1)**

**try:**

**count = int(count)**

**dic\_street\_count [street] = dic\_street\_count .get(street, 0) + count**

**except ValueError:**

**pass**

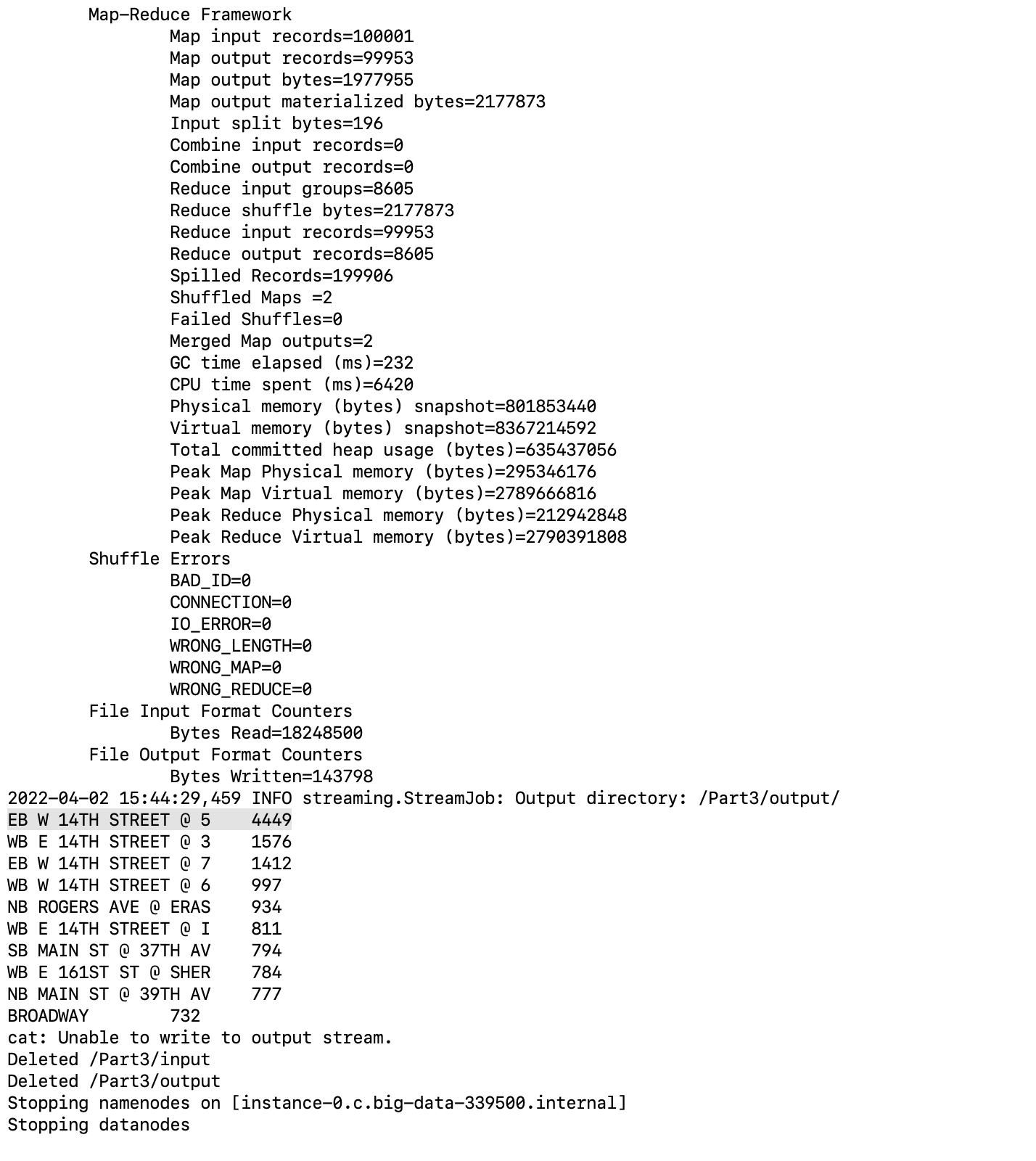
**sorted\_dict\_vtime\_count = sorted(dic\_street\_count .items(), key=itemgetter(1))[::-1]**

**for street, count in sorted\_dict\_vtime\_count:**

**print ('%s\t%s' % (street, count))**

**Test.sh:** Only output the key value with maximum count value using head -10 in tesh.sh file on hadoop. It shows **EB W 14TH STREET @ 5** has the maximum number of tickets issued that is **4449**.

****



**4 . Which color of the vehicle is most likely to get a ticket?**

**Mapper** **:** Using regex we have filtering the data where colors names and setting it as key of value value 1 as counter.

**import sys**

**import re**

**rule = re.compile('[A-Za-z]')**

**for line in sys.stdin:**

**line=line.strip(',')**

**color= line.split(",")[34]**

**match = rule.search(color)**

**if match:**

**print('%s\t%s' % ( color, '1'))**

**Reducer:**  Passing the output from mapper which has only values for the color as key and counter as value. Sorting all the values with the same key and combining them by adding their counters gives us the final count for each key value.

**import sys**

**from operator import itemgetter**

**dic\_color\_count = {}**

**total\_count=0**

**for line in sys.stdin:**

**color, count = line.split('\t',1)**

**try:**

**count = int(count)**

**dic\_color\_count [color] = dic\_color\_count .get(color, 0) + count**

**total\_count+=count**

**except ValueError:**

**pass**

**sorted\_dict\_color\_count = sorted(dic\_color\_count .items(), key=itemgetter(1))[::-1]**

**for color, count in sorted\_dict\_color\_count:**

**print ('%s\t%s' % (color, count/total\_count))**

**Test.sh:** Only output the key value with maximum count value using head -10 in tesh.sh file on hadoop. 

