**NBA Shot Logs**

For each pair of the players (A, B), we define the fear score of A when facing B as the hit rate, such that B is the closest defender when A is shooting. Based on the fear sore, for each player, please find out who is his ”most unwanted defender”.

**Mapper 1:** Filtering the data with respect to defender,player and shot and assigning a value 1 if shot is made or 0 if missed.

import sys

mapper1\_output=[]

for line in sys.stdin:

line=line.strip(',')

name= line.split(",")[-2]

shot= '1' if line.split(",")[14] == 'made' else '0'

defen\_las=line.split(",")[15].strip('"')

defen\_fir=line.split(",")[16].strip('"')

defen\_ful=defen\_fir+" "+defen\_las

defen\_ful.strip ("")

data=name+','+defen\_ful+'\t'+ shot

mapper1\_output.append(data)

for l in mapper1\_output:

print(l)

**Reducer 1:** Splitting the input and saving it in a dictionary where the key value is player and defender. Counter is set to calculate the total hits and save it as a string alone separated from the count of shots that have been made by “\t”.

import sys

from operator import itemgetter

dict\_score\_count = {}

reducer\_1\_ouput = []

for line in sys.stdin:

record = line.split('\t')

data,count = record[0], record[1] # count is the value 0 or 1 depending on the shot

missed /made

try:

count = int(count)

dict\_score\_count[data] = [dict\_score\_count.get(data, [0,0])[0] + count,

dict\_score\_count.get(data, [0,0])[1]+1] #total counter is set

except ValueError:

pass

for key, value in dict\_score\_count.items():

reducer\_1\_ouput.append(key+'\t'+str(value))

for l in reducer\_1\_ouput:

print(l)

**Mapper 2:** Calculating fear score for each player

import sys

mapper1\_output=[]

mapper\_2\_output = []

for line in sys.stdin:

players\_pairs, result\_shots = line.split('\t')

attacker, defender = players\_pairs.strip().split(',')

attacker = attacker.strip(); defender = defender.strip()

try:

result\_shots = eval(result\_shots)

if result\_shots[1] == 1 and result\_shots[0] == 0:

continue

else:

result\_shots = result\_shots[0]/result\_shots[1]

except:

pass

mapper\_2\_output.append(attacker + '\t' + defender + ',' + str(result\_shots))

for l in mapper\_2\_output:

print(l)

**Reducer 2**: Sorting values using attacker/player as key

import sys

reducer\_2\_output = {}

for entry in sys.stdin:

attacker, defender\_score = entry.split('\t')

defender, score = defender\_score.split(',')

score = float(score)

if attacker not in reducer\_2\_output:

reducer\_2\_output[attacker] = [(defender, score)]

else:

reducer\_2\_output[attacker].append((defender, score))

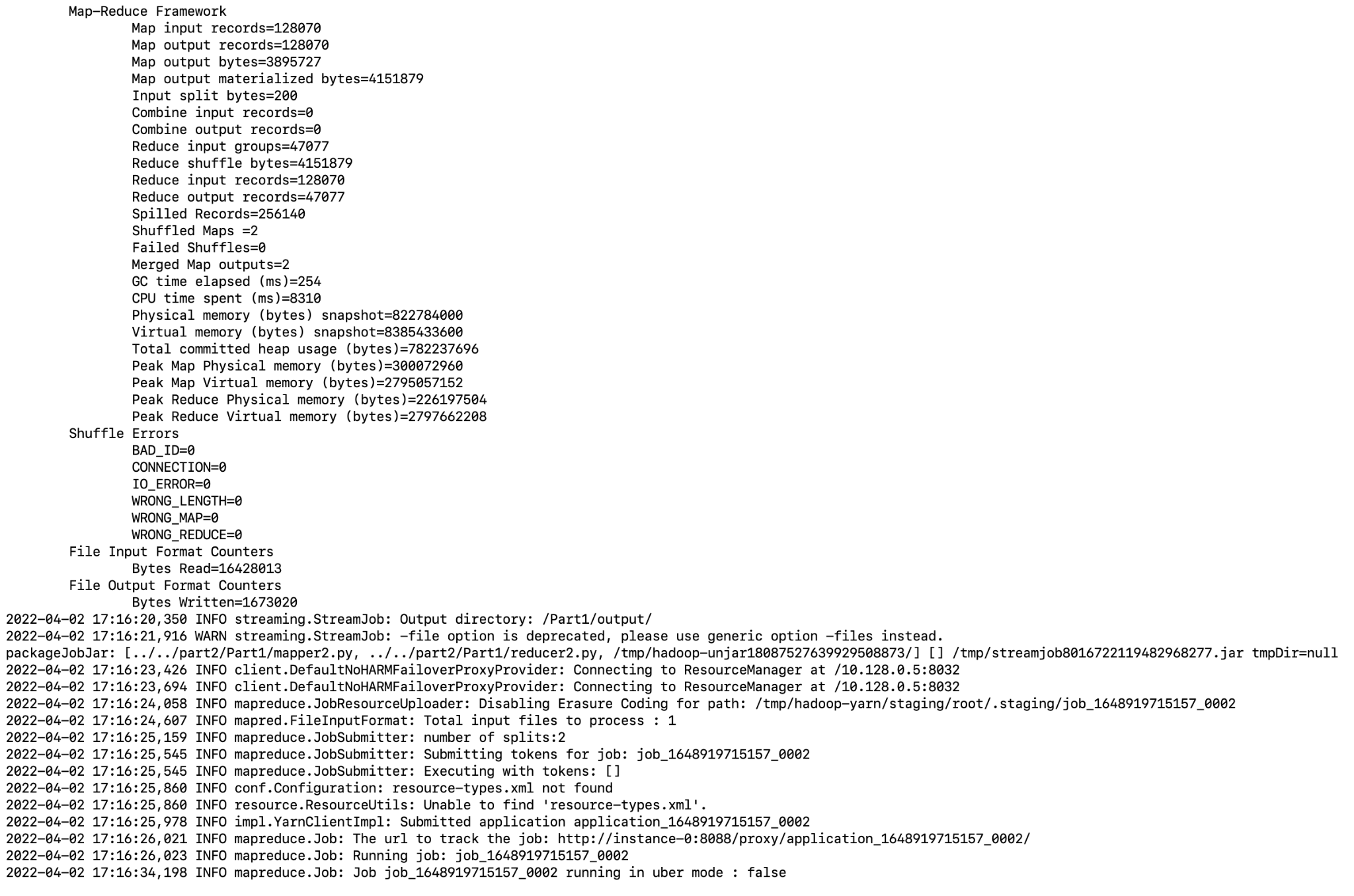
for player, defender\_score in reducer\_2\_output.items():

reducer\_2\_output[player] = sorted(reducer\_2\_output[player], key=lambda x: x[1])[0]

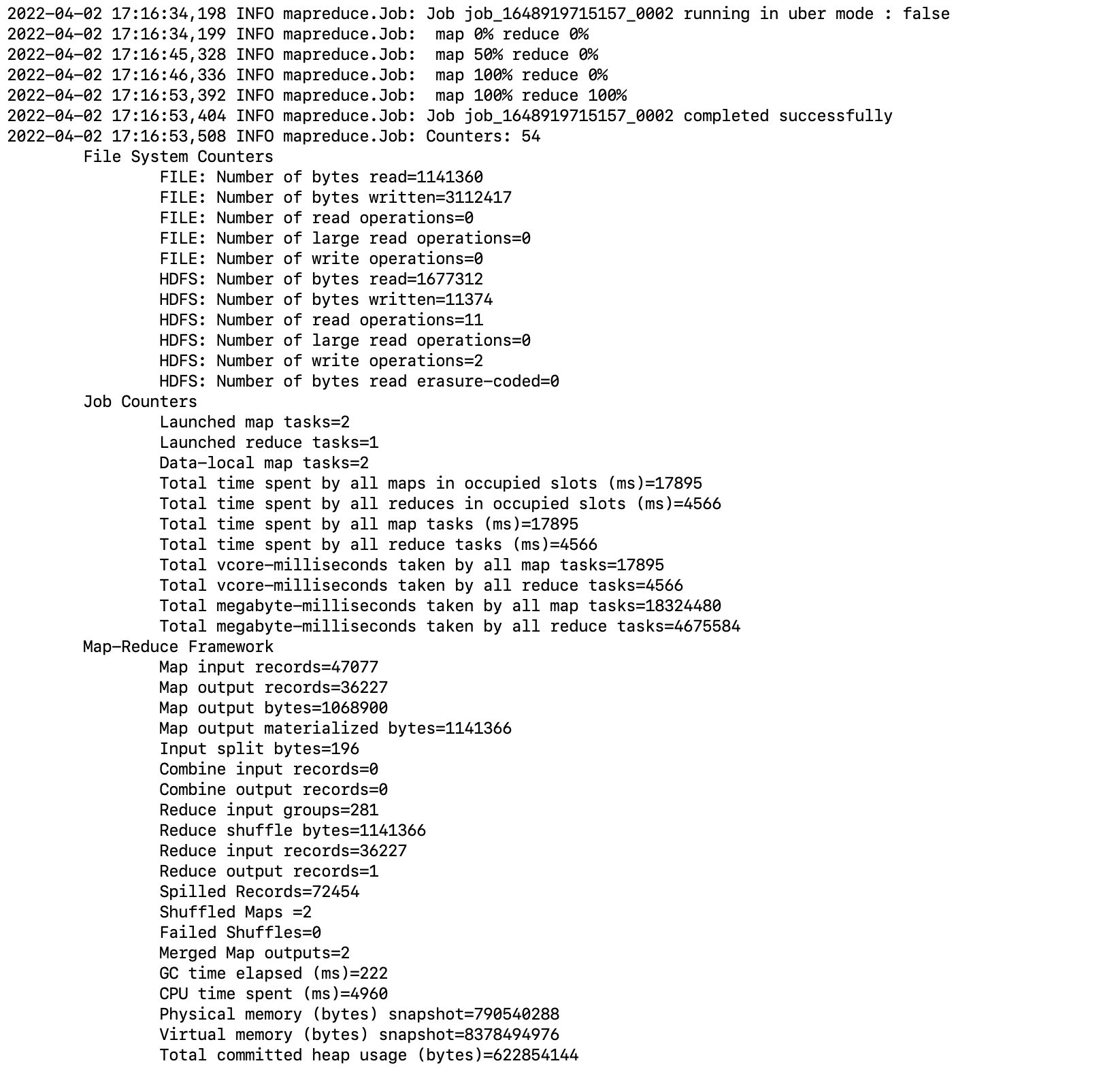
print(reducer\_2\_output)

**Mapper 1 and Reducer 1**





**Mapper 2 and Reducer 2**





**Part 2:** For each player, we define the comfortable zone of shooting is a matrix of,

{SHOT DIST, CLOSE DEF DIST, SHOT CLOCK}

Please develop a MapReduce-based algorithm to classify each player’s records into 4 comfortable zones. Considering the hit rate, which zone is the best for James Harden, Chris Paul, Stephen Curry and Lebron James.

**Mapper 1:** Creating a random cluster and then calculating the euclidean distance of each data point from each of the clusters and assigning it the key of the cluster with minimum distance. By using K -means clustering, we are finding 4 clusters for the matrix {SHOT DIST, CLOSE DEF DIST, SHOT CLOCK}.

import sys

zones\_mapper = sys.argv[1]

zones\_mapper = [eval(dp) for dp in zones\_mapper.split('Z')[1].strip('\_').split('\_')]

zones\_mapper = {1:[zones\_mapper[0],zones\_mapper[1],zones\_mapper[2]],

2:[zones\_mapper[3],zones\_mapper[4],zones\_mapper[5]],

3:[zones\_mapper[6],zones\_mapper[7],zones\_mapper[8]],

4:[zones\_mapper[9],zones\_mapper[10],zones\_mapper[11]]}

def euclidean\_distance(A, B):

return sum((a-b)\*\*2 for a, b in zip(A[:], B[:])) \*\* (1/2)

mapper\_1\_output = {1:[0,[0,0,0]],

2:[0,[0,0,0]],

3:[0,[0,0,0]],

4:[0,[0,0,0]]}

for line in sys.stdin:

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

line\_len = len(line)

if line\_len == 23:

try:

SHOT\_DIST = float(line[12].strip('"'))

CLOSE\_DEF\_DIST= float(line[-5].strip('"'))

SHOT\_CLOCK = float(line[9].strip('"'))

if SHOT\_DIST < 29 and CLOSE\_DEF\_DIST < 11 and SHOT\_CLOCK < 26: # removing outliers

data = [SHOT\_DIST, CLOSE\_DEF\_DIST, SHOT\_CLOCK]

data\_centroids\_distances = {1: euclidean\_distance(data, zones\_mapper[1]),

2: euclidean\_distance(data, zones\_mapper[2]),

3: euclidean\_distance(data, zones\_mapper[3]),

4: euclidean\_distance(data, zones\_mapper[4])}

data\_cluster\_key = min(data\_centroids\_distances, key = data\_centroids\_distances.get) #argmin

#data\_cluster\_distance = data\_centroids\_distances[data\_cluster\_key]

mapper\_1\_output[data\_cluster\_key][0] += 1 # counter

mapper\_1\_output[data\_cluster\_key][1][0] += data[0] #sum all SHOT\_DIST

mapper\_1\_output[data\_cluster\_key][1][1] += data[1] #sum all CLOSE\_DEF\_DIST

mapper\_1\_output[data\_cluster\_key][1][2] += data[2] #sum all SHOT\_CLOCK

except:

continue

combiner\_1\_input = mapper\_1\_output

#combiner\_1\_output = []

for key, values in combiner\_1\_input.items():

print('{key}\t{values}'.format(key=key, values=values))

**Reducer 1**: Finding the centroid of each cluster to update the new centroid in the next for loop which is implemented in the test.sh file.

import sys

reducer\_1\_output = {}

for line in sys.stdin:

key, values = line.split('\t')

values = eval(values)

count = values[0]

sum\_SHOT\_DIST = values[1][0]

sum\_CLOSE\_DEF\_DIST = values[1][1]

sum\_SHOT\_CLOCK = values[1][2]

reducer\_1\_output[int(key)] = [sum\_SHOT\_DIST/count, sum\_CLOSE\_DEF\_DIST/count, sum\_SHOT\_CLOCK/count]

output = ''

for key, values in reducer\_1\_output.items():

for value in values:

output = output + '\_' +str(value)

output = 'ClusterZ' + output[1:]

print(output)

**Mapper 2:** Filtering the data set just for the required players, calculating the euclidean distance of each data point from each of the 4 clusters and assigning it the key of the cluster with minimum distance. In this way we have 4 clusters for each player for the matrix {SHOT DIST, CLOSE DEF DIST, SHOT CLOCK} .

import sys

zones\_mapper = sys.argv[1]

zones\_mapper = [eval(dp) for dp in zones\_mapper.split('Z')[1].strip('\_').split('\_')]

zones\_mapper = {1:[zones\_mapper[0],zones\_mapper[1],zones\_mapper[2]],

2:[zones\_mapper[3],zones\_mapper[4],zones\_mapper[5]],

3:[zones\_mapper[6],zones\_mapper[7],zones\_mapper[8]],

4:[zones\_mapper[9],zones\_mapper[10],zones\_mapper[11]]}

def euclidean\_distance(A, B):

return sum((a-b)\*\*2 for a, b in zip(A[:], B[:])) \*\* (1/2)

players = ['stephen curry', 'james harden', 'chris paul','lebron james']

for line in sys.stdin:

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

line\_len = len(line)

player = line[-2]

if line\_len == 23 and player in players:

try:

player = player.split(' ')

player = player[0]+player[1]

shot = 1 if line[14] == 'made' else 0

SHOT\_DIST = float(line[12].strip('"'))

CLOSE\_DEF\_DIST = float(line[-5].strip('"'))

SHOT\_CLOCK = float(line[9].strip('"'))

data = [SHOT\_DIST, CLOSE\_DEF\_DIST, SHOT\_CLOCK]

data\_centroids\_distances = {1: euclidean\_distance(data, zones\_mapper[1]),

2: euclidean\_distance(data, zones\_mapper[2]),

3: euclidean\_distance(data, zones\_mapper[3]),

4: euclidean\_distance(data, zones\_mapper[4])}

data\_cluster\_key = min(data\_centroids\_distances, key = data\_centroids\_distances.get) #argmin

print(player +"\_"+str(data\_cluster\_key)+"\t"+str(shot))

except:

continue

**Reducer 2**: Finding the count for the hit rate by counting the shots against each player within each cluster.

import sys

dict\_score\_count = {}

for line in sys.stdin:

record = line.split('\t')

data,count = record[0], record[1]

try:

count = int(count)

dict\_score\_count[data] = [dict\_score\_count.get(data, [0,0])[0] + count, dict\_score\_count.get(data, [0,0])[1]+1]

except ValueError:

pass

for key, value in dict\_score\_count.items():

print(key+'\t'+str(value))

**Mapper 3:** Calculating hit rate for each player.

import sys

for entry in sys.stdin:

player\_cluster, result\_shots = entry.split('\t')

try:

result\_shots = eval(result\_shots)

if result\_shots[1] == 1 and result\_shots[0] == 0:

continue

else:

hit\_rate = result\_shots[0]/result\_shots[1]

except:

pass

print(player\_cluster+'\t'+str(hit\_rate))

**Reducer 3:** Sorting and shuffling the output of the reducer to get the cluster with the highest hit rate.

import sys

reducer\_3\_output = {}

for entry in sys.stdin:

player\_cluster, hit\_rate = entry.split('\t')

player, cluster = player\_cluster.split('\_')

hit\_rate = float(hit\_rate)

if player not in reducer\_3\_output:

reducer\_3\_output[player] = [cluster, hit\_rate]

elif hit\_rate > reducer\_3\_output[player][1]:

reducer\_3\_output[player] = [cluster, hit\_rate]

else:

continue

for key, value in reducer\_3\_output.items():

print(key+'\t'+'Cluster: '+str(value[0])+' | Hit Rate: '+str(value[1]))