Note: **I am using 3 nodes clusters.**

Hyperlink to each part		
Part 1.1 Kmeans	Part 1.2 Kmedian	Part 2.2 SQL query

Part 1) Clustering

Part 1.1 – Kmeans and Euclidean distance.

First step - Generate data: kmeans_data.csv and centers.txt . I am doing manually.

Generated data: 2,000,000 rows and 5 columns.

#Create python file.
vim generateKmeans_data.py

```
#!/usr/bin/python

import random

random.seed(1997)
#Generate 2M 5-dimensional data points with random values between 0 and 1.
k_meansData = [[random.random() for column in range(5)] for row in range(2000000)]

with open('kmeans_data.csv', 'w') as f:
    for row in k_meansData:
        f.write(','.join(map(str, row)) + '\n')
```

#Run the python file to generate data.
python generateKmeans_data.py

#Get row count to make sure.
cat kmeans_data.csv | wc -l

```
[ec2-user@ip-172-31-84-48 ~]$ cat kmeans_data.csv | wc -l
2000000
```

Generated initial centroid: 5 rows. Using random selection of data points.

Note: I attempted to use Kmeans++ to initiate centers. This method supposed to reduce the potential of picking two close starting centroids. However, it took me 40 minutes and the code were still running so I opt out back to random points selection.

```
#!/usr/bin/python
import random
import csv

def clusterCenters(data, k):
    '''Get the initial cluster centers using random points selection.'''
    random.seed(1997)
    return random.sample(data, k)

with open('kmeans_data.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    #Extract individual data from each cell.
    data = [[float(value) for value in row] for row in reader]

#Chosen number of k clusters.
kNum = 5
initialCenters = clusterCenters(data, kNum)
with open('centers.txt', 'w') as f:
    for center in initialCenters:
        f.write(','.join(map(str, center)) + '\n')
```

#Create python file.
vim generateCenters.py

#Run the python file to generate data.
python generateCenters.py

#Get row count to make sure.
cat centers.txt | wc -l

```
[ec2-user@ip-172-31-84-48 ~]$ cat centers.txt | wc -l
5
```

#Get info of the initial centroids.
#Since I set seed = 1997 for reproducibility, should have the same initial centroids value every time running the code.
cat centers.txt

```
[ec2-user@ip-172-31-84-48 ~]$ cat centers.txt
0.4887727605392773,0.9618161430929605,0.2280295683524678,0.7563450827897548,0.511982584280626
0.2604962542619079,0.04646010403729672,0.48344166693762614,0.45196135466419884,0.06976430043427806
0.19226415529331586,0.08673205714755505,0.9263951096210629,0.4672350834978266,0.8398714697055457
0.5521594137917458,0.8626278724008317,0.523453025364487,0.5553843179244647,0.7274553513797444
0.6640941849654838,0.8239596229293283,0.14307179562804817,0.8300618801462052,0.5078786532062929
```

#Create HDFS directory 'data'.
hadoop fs -mkdir /data

#Copy three tables to HDFS for processing
hadoop fs -put kmeans_data.csv /data

Important Notes:

✍ If you want to copy my code to run Hadoop job, please open FinalExam-CSC555.py/ .pynb/ .html file.

✍ With the way my code set up, if running Hadoop Streaming separately, output from previous Map Reduce once downloaded need to be moved to a different folder 'centers' at local machine in order to run the next Map Reduce. I want to store my updated centroids after each iteration.

✍ I know there are other more convenient ways, but I want to follow my logic and make it work.

Second step - Mapper and Reducer.

Kmeans Mapper : Role -

✍ Create a dictionary from 'centers.txt' file, or file store centroid information.

✍ For each line of input file, calculate Euclidean distance between that line/data point to each centroid.

✍ Assign that data point to nearest centroid, i.e., smallest distance by obtaining respective index.

✍ Output has key is the cluster number. And values are rows from original data.

#Create Kmeans Mapper.

vim kmapper.py

```python
#!/usr/bin/python
import sys

#Detect whichever centers file being used for current iteration.
input_centerFiles = ['centers.txt', 'centers1.txt', 'centers2.txt', 'centers3.txt', 'centers4.txt']

#Try to open the file, if not work then pass.
for input_centerFile in input_centerFiles:
    try:
        with open(input_centerFile, 'r') as centerFile:
            centersDict = {}
            for i, line in enumerate(centerFile):
                centersDict[i]=[float(center) for center in line.strip().split(',')]
            break
    except FileNotFoundError:
        pass

for dataPoint in sys.stdin:
    #Get the five features, stored as a list.
    featuresList = [float(feature) for feature in dataPoint.strip().split(',')]

    #Calculate Euclidean distance between current data point and each centroid.
    euclidDistance_list = [sum((c - f) ** 2 for c, f in zip(centroid, featuresList))
                           for centroid in centersDict.values()]

    #Get the cluster number associated with smallest distance.
    clusterID = euclidDistance_list.index(min(euclidDistance_list))
    #Output format: 'clusterNumber|feature1|feature2|feature3|feature4|feature5'
    print(f"{clusterID + 1}|{'|'.join(map(str, featuresList))}")
```

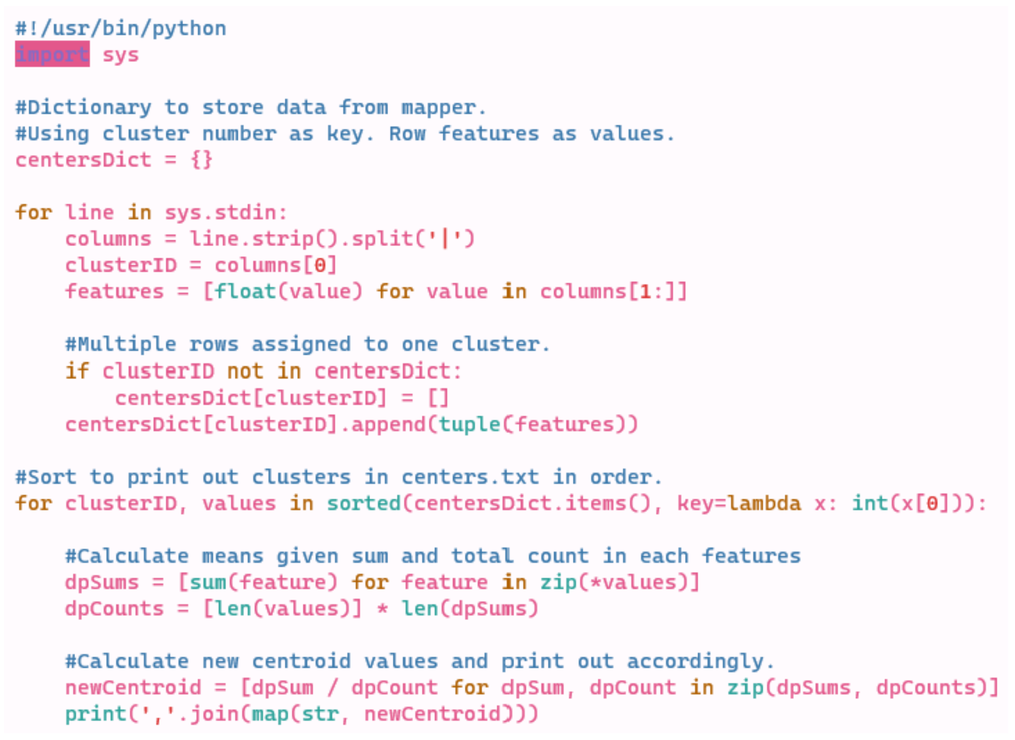
Kmeans Reducer: Role -

✍ For each line of input file, retrieve and store again as a dictionary, given same data schema like mapper.

✍ Calculate the mean values of each feature within a cluster. --Should be 5 means values.--

✍ Output format will be same as the original ‘centers.txt’.

#Create Kmeans Reducer.
vim kreducer.py



```python
#!/usr/bin/python
import sys

#Dictionary to store data from mapper.
#Using cluster number as key. Row features as values.
centersDict = {}

for line in sys.stdin:
    columns = line.strip().split('|')
    clusterID = columns[0]
    features = [float(value) for value in columns[1:]]

    #Multiple rows assigned to one cluster.
    if clusterID not in centersDict:
        centersDict[clusterID] = []
    centersDict[clusterID].append(tuple(features))

#Sort to print out clusters in centers.txt in order.
for clusterID, values in sorted(centersDict.items(), key=lambda x: int(x[0])):

    #Calculate means given sum and total count in each features
    dpSums = [sum(feature) for feature in zip(*values)]
    dpCounts = [len(values)] * len(dpSums)

    #Calculate new centroid values and print out accordingly.
    newCentroid = [dpSum / dpCount for dpSum, dpCount in zip(dpSums, dpCounts)]
    print(','.join(map(str, newCentroid)))
```

Hadoop Streaming separated commands: --MUST follow this flow to get the code work—

#At this point, the original ‘centers.txt’ should be at root directory. i.e, /home/ec2-user/
#Check content of centers.txt
cat centers.txt

1st Iteration MR:
time hadoop jar hadoop-streaming-2.6.4.jar -input /data/kmeans_data.csv -output /data/Part1_Output1_Means -mapper kmapper.py -reducer kreducer.py -file kmapper.py -file kreducer.py -file centers.txt

```
23/11/21 15:26:09 INFO streaming.StreamJob: Output directory: /data/Part1_Output1_Means

real    0m50.775s
user    0m3.834s
sys     0m0.355s
```

#Download 1st MR output to local machine.
hadoop fs -get /data/Part1_Output1_Means/part-00000

#Create directory 'centersMeans' to store centroid text file.
mkdir centersMeans

#Move the original centers.txt file there -MUST be done to run next iteration.--
mv centers.txt centersMeans/

#Rename 1st MR output, currently part-00000 to centers1.txt
mv part-00000 centers1.txt
cat centers1.txt

```
[ec2-user@ip-172-31-84-48 ~]$ cat centers1.txt
0.2916964619879541,0.8013903373038839,0.28989915829549995,0.6321136696873195,0.3652142948148682
0.4333260667569702,0.2633215038024255,0.47906778000966826,0.4442971904761 2715,0.24803868890197278
0.38139367542500807,0.23079997703556399,0.7079012419152251,0.4746528876682291,0.7376847332302652
0.5604831110694299,0.6807710708166049,0.58768341371428,0.42938239968727326,0.625544130298499
0.7349985892679249,0.5314514223951852,0.22613097074556807,0.7210687726327598,0.4618907684899624
```

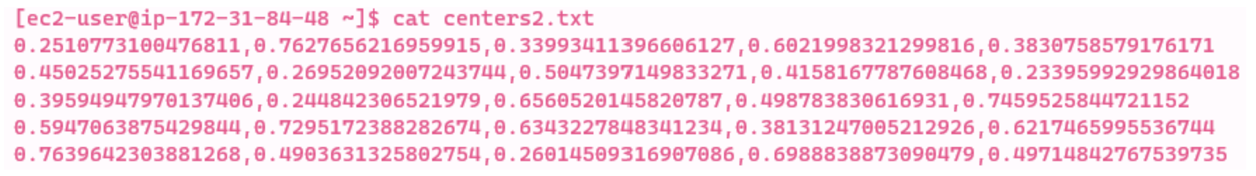
2nd Iteration MR:
time hadoop jar hadoop-streaming-2.6.4.jar -input /data/kmeans_data.csv -output /data/Part1_Output2_Means -mapper kmapper.py -reducer kreducer.py -file kmapper.py -file kreducer.py -file centers1.txt

```
23/11/21 15:28:54 INFO streaming.StreamJob: Output directory: /data/Part1_Output2_Means

real    0m55.247s
user    0m3.782s
sys     0m0.414s
```

#Download 2nd MR output to local machine.
hadoop fs -get /data/Part1_Output2_Means/part-00000

#Move centers1.txt file to 'centersMeans' directory -MUST be done to run next iteration.--
mv centers1.txt centersMeans/

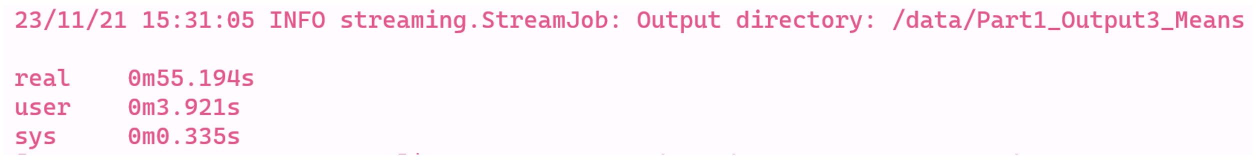
#Rename 2nd MR output, currently part-00000 to centers2.txt
mv part-00000 centers2.txt
cat centers2.txt



```
[ec2-user@ip-172-31-84-48 ~]$ cat centers2.txt
0.2510773100476811,0.7627656216959915,0.33993411396606127,0.6021998321299816,0.3830758579176171
0.45025275541169657,0.26952092007243744,0.5047397149833271,0.4158167787608468,0.23395992929864018
0.39594947970137406,0.244842306521979,0.6560520145820787,0.498783830616931,0.7459525844721152
0.5947063875429844,0.7295172388282674,0.6343227848341234,0.38131247005212926,0.6217465995536744
0.7639642303881268,0.4903631325802754,0.26014509316907086,0.6988838873090479,0.49714842767539735
```

3rd Iteration MR:

time hadoop jar hadoop-streaming-2.6.4.jar -input /data/kmeans_data.csv -output /data/Part1_Output3_Means -mapper kmapper.py -reducer kreducer.py -file kmapper.py -file kreducer.py -file centers2.txt

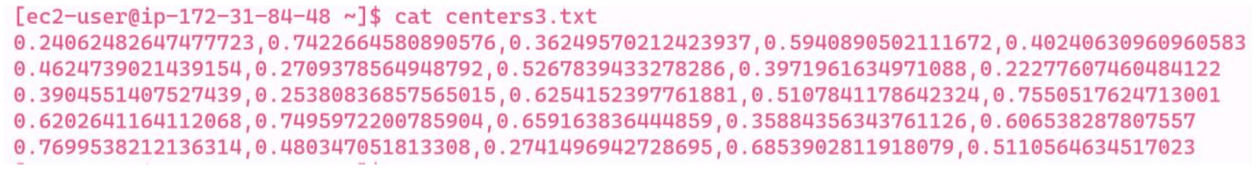


```
23/11/21 15:31:05 INFO streaming.StreamJob: Output directory: /data/Part1_Output3_Means

real    0m55.194s
user    0m3.921s
sys     0m0.335s
```

#Download 3rd MR output to local machine.
hadoop fs -get /data/Part1_Output3_Means/part-00000

#Move centers2.txt file to 'centersMeans' directory -MUST be done to run next iteration.--
mv centers2.txt centersMeans/

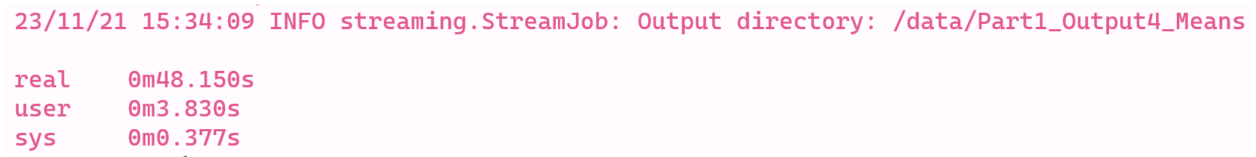
#Rename 3rd MR output, currently part-00000 to centers3.txt
mv part-00000 centers3.txt
cat centers3.txt



```
[ec2-user@ip-172-31-84-48 ~]$ cat centers3.txt
0.24062482647477723,0.7422664580890576,0.36249570212423937,0.5940890502111672,0.40240630960960583
0.4624739021439154,0.2709378564948792,0.5267839433278286,0.3971961634971088,0.22277607460484122
0.3904551407527439,0.25380836857565015,0.6254152397761881,0.5107841178642324,0.7550517624713001
0.6202641164112068,0.7495972200785904,0.659163836444859,0.35884356343761126,0.606538287807557
0.7699538212136314,0.480347051813308,0.2741496942728695,0.6853902811918079,0.5110564634517023
```

4th MR:

time hadoop jar hadoop-streaming-2.6.4.jar -input /data/kmeans_data.csv -output /data/Part1_Output4_Means -mapper kmapper.py -reducer kreducer.py -file kmapper.py -file kreducer.py -file centers3.txt

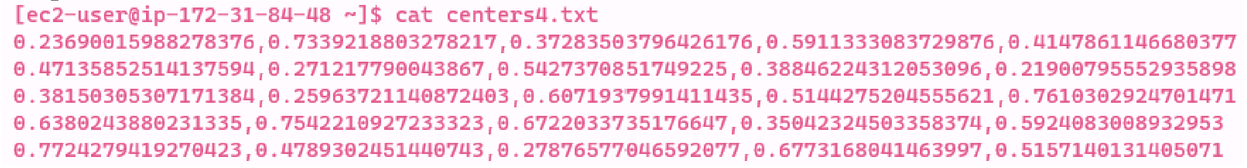


```
23/11/21 15:34:09 INFO streaming.StreamJob: Output directory: /data/Part1_Output4_Means

real    0m48.150s
user    0m3.830s
sys     0m0.377s
```

#Download 4th MR output to local machine.
hadoop fs -get /data/Part1_Output4_Means/part-00000

#Move centers3.txt file to 'centers' directory -MUST be done to run next iteration.--
mv centers3.txt centersMeans/

#Rename 4th MR output, currently part-00000 to centers4.txt
mv part-00000 centers4.txt



```
[ec2-user@ip-172-31-84-48 ~]$ cat centers4.txt
0.23690015988278376,0.7339218803278217,0.37283503796426176,0.5911333083729876,0.4147861146680377
0.47135852514137594,0.271217790043867,0.5427370851749225,0.38846224312053096,0.21900795552935898
0.38150305307171384,0.25963721140872403,0.6071937991411435,0.5144275204555621,0.7610302924701471
0.6380243880231335,0.7542210927233323,0.6722033735176647,0.35042324503358374,0.5924083008932953
0.7724279419270423,0.4789302451440743,0.27876577046592077,0.6773168041463997,0.5157140131405071
```

#Move centers4.txt file to 'centersMeans' directory.
mv centers4.txt centersMeans/

#Go to 'centers' directory, all centroid outputs are store here.
cd centersMeans
ls

#Get latest centroid output.
cat centers4.txt

#Go back to root directory to run next problem/part.
cd

Total Time: 209.546 seconds, approximately 3.5 mins.

Using MapReduceChain.java: Full script in Kmeans_MapReduceChain.java.

Note: Java code using same logic as done manually. It took me a lot of time to compile this code. You will see exceptions raise when running the code, that exception is to check if there is output from previous map reduce to download to local directory. The code is still working correctly.

#Run python file again to regenerate initial centroid.
#Since I am using seed, it should generate the same file.
python generateCenters.py

vim MapReduceChain.java

javac -cp ~/hadoop-streaming-2.6.4.jar:$(hadoop classpath) MapReduceChain.java

jar -cf MapReduceChain.jar MapReduceChain.class

jar xvf MapReduceChain.jar META-INF/MANIFEST.MF

nano META-INF/MANIFEST.MF

-- Paste in: Class-Path: /home/ec2-user/hadoop-streaming-2.6.4.jar --

jar cmf META-INF/MANIFEST.MF MapReduceChain.jar MapReduceChain.class

time hadoop jar MapReduceChain.jar MapReduceChain

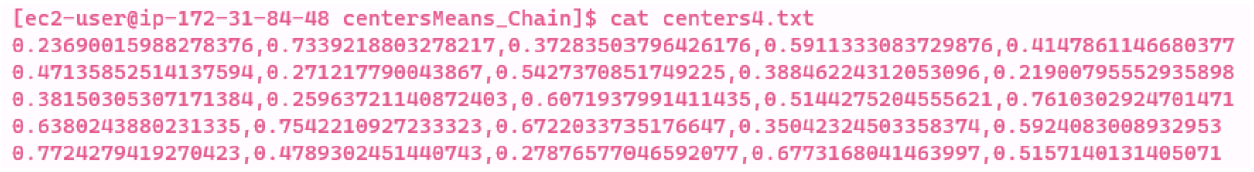
#Download 4th MR output to local machine.
hadoop fs -get /Part1_Output4_Means/part-00000

#Rename 4th MR output, currently part-00000 to centers4.txt
mv part-00000 centers4.txt

#Move centers4.txt file to ‘centersMeans_Chain’ directory.
mv centers4.txt centersMeans_Chain/

#Go to centersMeans_Chain directory. All centers data should be there.
cd centersMeans_Chain
ls

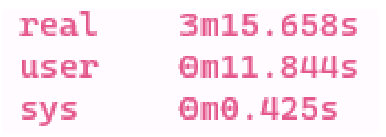
#Get latest centroid output.
cat centers4.txt



```
[ec2-user@ip-172-31-84-48 centersMeans_Chain]$ cat centers4.txt
0.23690015988278376,0.7339218803278217,0.37283503796426176,0.5911333083729876,0.4147861146680377
0.47135852514137594,0.271217790043867,0.5427370851749225,0.38846224312053096,0.21900795552935898
0.38150305307171384,0.25963721140872403,0.6071937991411435,0.5144275204555621,0.7610302924701471
0.6380243880231335,0.7542210927233323,0.6722033735176647,0.35042324503358374,0.5924083008932953
0.7724279419270423,0.4789302451440743,0.27876577046592077,0.6773168041463997,0.5157140131405071
```

#Go back to root directory to run next problem/part.
cd

Time:



```
real    3m15.658s
user    0m11.844s
sys     0m0.425s
```

Part 1.2 – Kmedian and Manhattan distance.

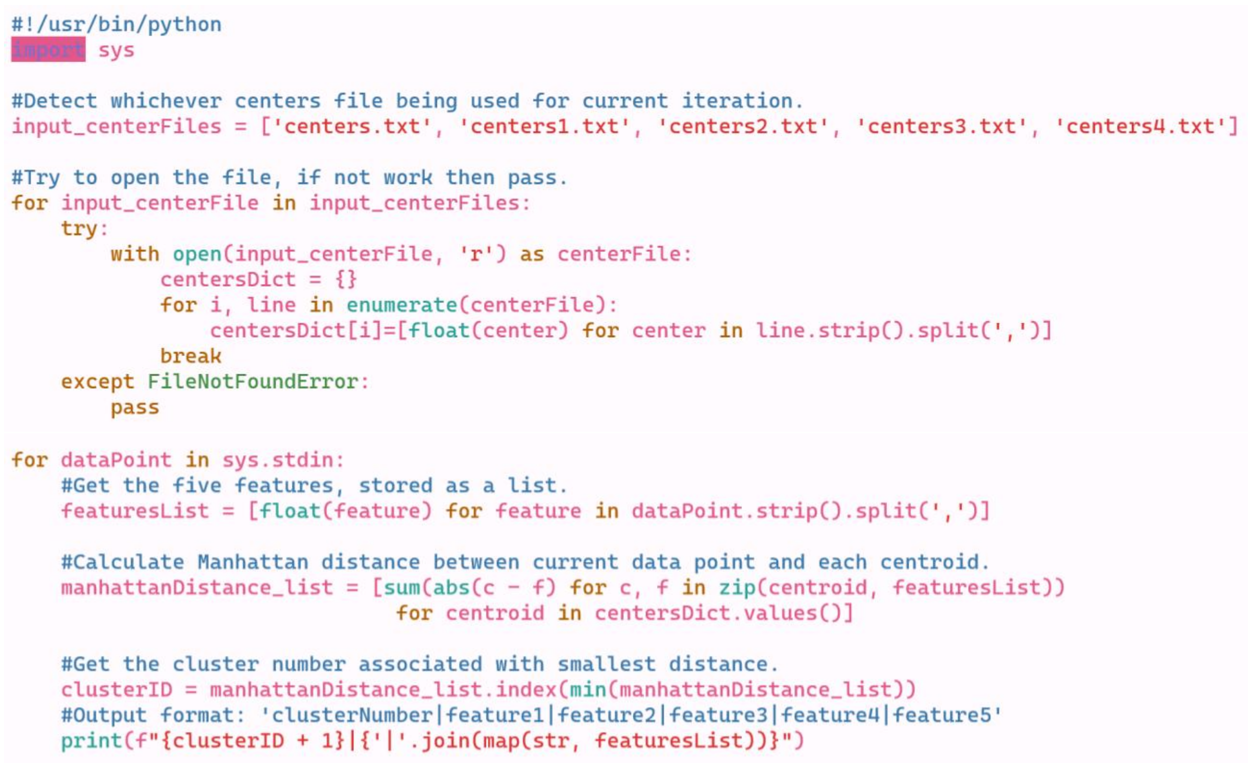
Second step - Mapper and Reducer.

Kmedian Mapper : Role -

✍ Same logic as Kmeans, the only thing different is using Manhattan distance.

#Create KmedianMapper.

vim kmedian_mapper.py



```python
#!/usr/bin/python
import sys

#Detect whichever centers file being used for current iteration.
input_centerFiles = ['centers.txt', 'centers1.txt', 'centers2.txt', 'centers3.txt', 'centers4.txt']

#Try to open the file, if not work then pass.
for input_centerFile in input_centerFiles:
    try:
        with open(input_centerFile, 'r') as centerFile:
            centersDict = {}
            for i, line in enumerate(centerFile):
                centersDict[i]=[float(center) for center in line.strip().split(',')]
            break
    except FileNotFoundError:
        pass

for dataPoint in sys.stdin:
    #Get the five features, stored as a list.
    featuresList = [float(feature) for feature in dataPoint.strip().split(',')]

    #Calculate Manhattan distance between current data point and each centroid.
    manhattanDistance_list = [sum(abs(c - f) for c, f in zip(centroid, featuresList))
                              for centroid in centersDict.values()]

    #Get the cluster number associated with smallest distance.
    clusterID = manhattanDistance_list.index(min(manhattanDistance_list))
    #Output format: 'clusterNumber|feature1|feature2|feature3|feature4|feature5'
    print(f"{clusterID + 1}|{'|'.join(map(str, featuresList))}")
```

Kmedian Reducer: Role -

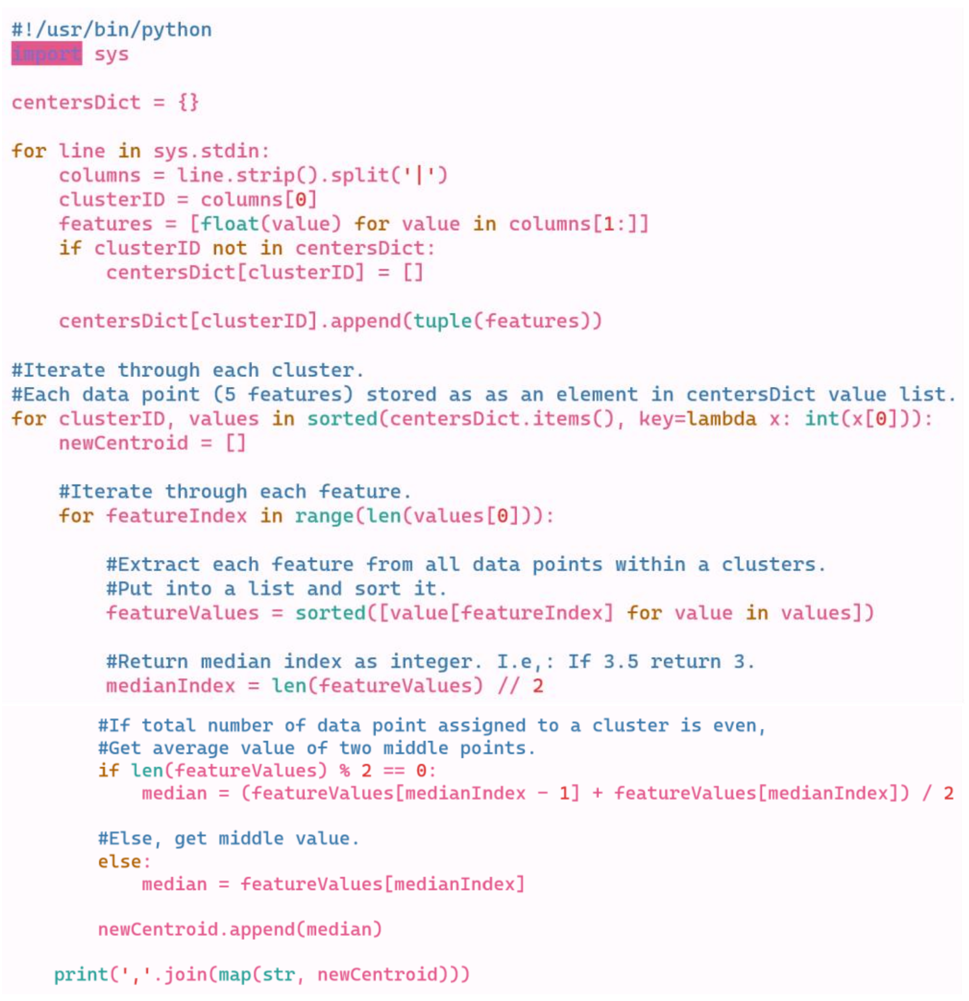
✍ For each line of input file, retrieve and store again as a dictionary, given same data schema like mapper.

✍ Calculate the median values of each feature within a cluster. --Should be 5 median values.—

✍ Manual median value retrieval logic. If length of sorted list is odd, median value = middle value. Else, if length of sorted list is even, median value = average of two middle values.

✍ Output format will be same as the original ‘centers.txt’.

#Create Kmedianreducer.
vim kmedian_reducer.py

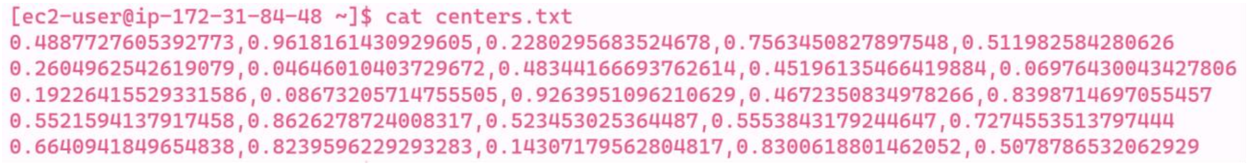


```python
#!/usr/bin/python
import sys

centersDict = {}

for line in sys.stdin:
    columns = line.strip().split('|')
    clusterID = columns[0]
    features = [float(value) for value in columns[1:]]
    if clusterID not in centersDict:
        centersDict[clusterID] = []

    centersDict[clusterID].append(tuple(features))

#Iterate through each cluster.
#Each data point (5 features) stored as as an element in centersDict value list.
for clusterID, values in sorted(centersDict.items(), key=lambda x: int(x[0])):
    newCentroid = []

    #Iterate through each feature.
    for featureIndex in range(len(values[0])):

        #Extract each feature from all data points within a clusters.
        #Put into a list and sort it.
        featureValues = sorted([value[featureIndex] for value in values])

        #Return median index as integer. I.e,: If 3.5 return 3.
        medianIndex = len(featureValues) // 2

        #If total number of data point assigned to a cluster is even,
        #Get average value of two middle points.
        if len(featureValues) % 2 == 0:
            median = (featureValues[medianIndex - 1] + featureValues[medianIndex]) / 2

        #Else, get middle value.
        else:
            median = featureValues[medianIndex]

        newCentroid.append(median)

    print(','.join(map(str, newCentroid)))
```

Hadoop Streaming separated commands: --MUST follow this flow to get the code work—

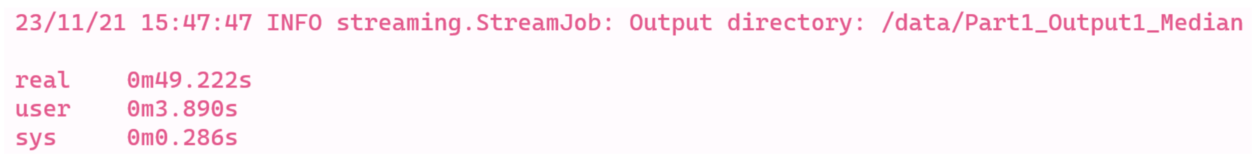
#Run python file again to regenerate initial centroid.
#Since I am using seed, it should generate the same file.
python generateCenters.py

#At this point, the original 'centers.txt' should be at root directory. i.e, /home/ec2-user/
#Check content of centers.txt
cat centers.txt



```
[ec2-user@ip-172-31-84-48 ~]$ cat centers.txt
0.4887727605392773,0.9618161430929605,0.2280295683524678,0.7563450827897548,0.511982584280626
0.2604962542619079,0.04646010403729672,0.48344166693762614,0.45196135466419884,0.06976430043427806
0.19226415529331586,0.08673205714755505,0.9263951096210629,0.4672350834978266,0.8398714697055457
0.5521594137917458,0.8626278724008317,0.523453025364487,0.5553843179244647,0.7274553513797444
0.6640941849654838,0.8239596229293283,0.14307179562804817,0.8300618801462052,0.5078786532062929
```

1st Iteration MR:
time hadoop jar hadoop-streaming-2.6.4.jar -input /data/kmeans_data.csv -output /data/Part1_Output1_Median -mapper kmedian_mapper.py -reducer kmedian_reducer.py -file kmedian_mapper.py -file kmedian_reducer.py -file centers.txt



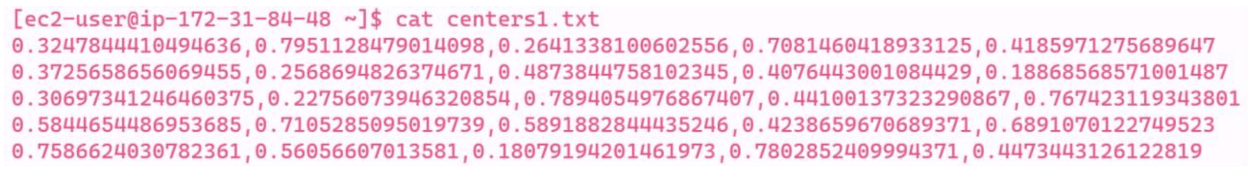
```
23/11/21 15:47:47 INFO streaming.StreamJob: Output directory: /data/Part1_Output1_Median

real    0m49.222s
user    0m3.890s
sys     0m0.286s
```

#Download 1st MR output to local machine.
hadoop fs -get /data/Part1_Output1_Median/part-00000

#Create directory ‘centersMedian’ to store centroid text file.
mkdir centersMedian

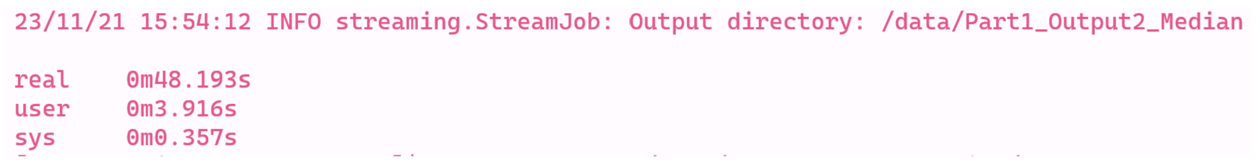
#Move the original centers.txt file there -MUST be done to run next iteration.--
mv centers.txt centersMedian/

#Rename 1st MR output, currently part-00000 to centers1.txt
mv part-00000 centers1.txt
cat centers1.txt



```
[ec2-user@ip-172-31-84-48 ~]$ cat centers1.txt
0.3247844410494636,0.7951128479014098,0.2641338100602556,0.7081460418933125,0.4185971275689647
0.3725658656069455,0.2568694826374671,0.4873844758102345,0.4076443001084429,0.18868568571001487
0.30697341246460375,0.22756073946320854,0.7894054976867407,0.44100137323290867,0.767423119343801
0.5844654486953685,0.7105285095019739,0.5891882844435246,0.4238659670689371,0.6891070122749523
0.7586624030782361,0.56056607013581,0.18079194201461973,0.7802852409994371,0.4473443126122819
```

2nd Iteration MR:
time hadoop jar hadoop-streaming-2.6.4.jar -input /data/kmeans_data.csv -output /data/Part1_Output2_Median -mapper kmedian_mapper.py -reducer kmedian_reducer.py -file kmedian_mapper.py -file kmedian_reducer.py -file centers1.txt



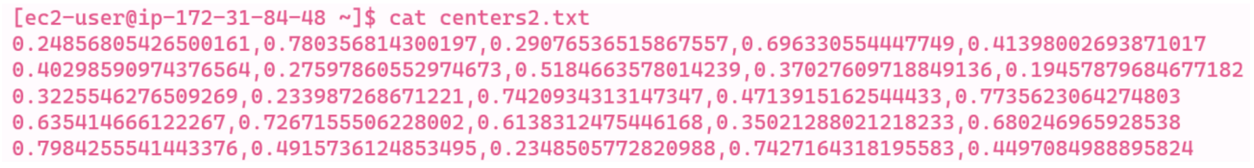
```
23/11/21 15:54:12 INFO streaming.StreamJob: Output directory: /data/Part1_Output2_Median

real    0m48.193s
user    0m3.916s
sys     0m0.357s
```

#Download 2nd MR output to local machine.
hadoop fs -get /data/Part1_Output2_Median/part-00000

#Move centers1.txt file to ‘centersMedian’ directory -MUST be done to run next iteration.--
mv centers1.txt centersMedian/

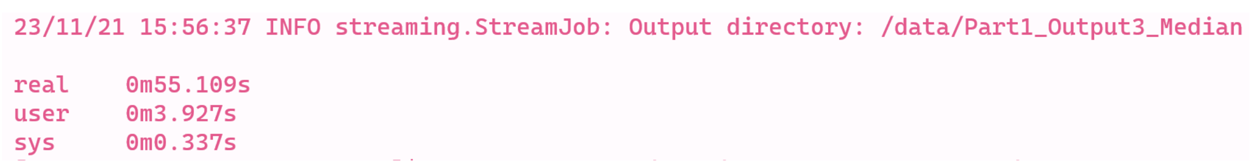
#Rename 2nd MR output, currently part-00000 to centers2.txt
mv part-00000 centers2.txt

cat centers2.txt



```
[ec2-user@ip-172-31-84-48 ~]$ cat centers2.txt
0.24856805426500161,0.780356814300197,0.29076536515867557,0.696330554447749,0.41398002693871017
0.40298590974376564,0.27597860552974673,0.5184663578014239,0.37027609718849136,0.19457879684677182
0.3225546276509269,0.233987268671221,0.7420934313147347,0.4713915162544433,0.773562306427480З
0.635414666122267,0.7267155506228002,0.6138312475446168,0.35021288021218233,0.680246965928538
0.7984255541443376,0.4915736124853495,0.2348505772820988,0.7427164318195583,0.4497084988895824
```

3rd Iteration MR:
time hadoop jar hadoop-streaming-2.6.4.jar -input /data/kmeans_data.csv -output /data/Part1_Output3_Median -mapper kmedian_mapper.py -reducer kmedian_reducer.py -file kmedian_mapper.py -file kmedian_reducer.py -file centers2.txt

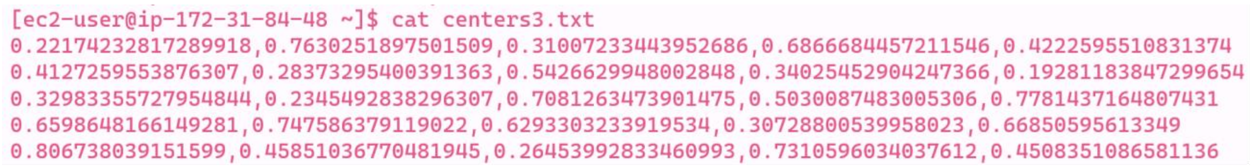


```
23/11/21 15:56:37 INFO streaming.StreamJob: Output directory: /data/Part1_Output3_Median

real    0m55.109s
user    0m3.927s
sys     0m0.337s
```

#Download 3rd MR output to local machine.
hadoop fs -get /data/Part1_Output3_Median/part-00000

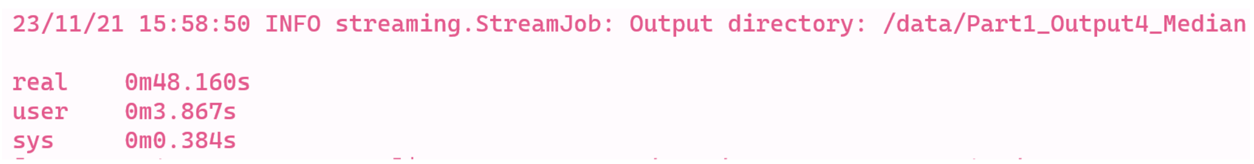
#Move centers2.txt file to 'centersMedian' directory -MUST be done to run next iteration.--
mv centers2.txt centersMedian/

#Rename 3rd MR output, currently part-00000 to centers3.txt
mv part-00000 centers3.txt
cat centers3.txt



```
[ec2-user@ip-172-31-84-48 ~]$ cat centers3.txt
0.22174232817289918,0.7630251897501509,0.31007233443952686,0.6866684457211546,0.4222595510831374
0.4127259553876307,0.28373295400391363,0.5426629948002848,0.34025452904247366,0.19281183847299654
0.32983355727954844,0.2345492838296307,0.7081263473901475,0.5030087483005306,0.7781437164807431
0.6598648166149281,0.747586379119022,0.6293303233919534,0.30728800539958023,0.66850595613349
0.806738039151599,0.45851036770481945,0.26453992833460993,0.7310596034037612,0.450835108658113б
```

4th Iteration MR:
time hadoop jar hadoop-streaming-2.6.4.jar -input /data/kmeans_data.csv -output /data/Part1_Output4_Median -mapper kmedian_mapper.py -reducer kmedian_reducer.py -file kmedian_mapper.py -file kmedian_reducer.py -file centers3.txt

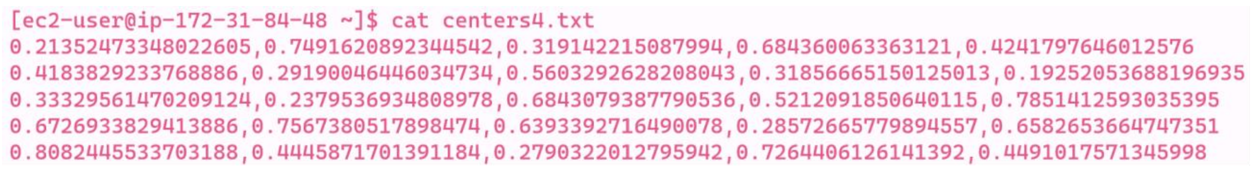


```
23/11/21 15:58:50 INFO streaming.StreamJob: Output directory: /data/Part1_Output4_Median

real    0m48.160s
user    0m3.867s
sys     0m0.384s
```

#Download 4th MR output to local machine.
hadoop fs -get /data/Part1_Output4_Median/part-00000

#Move centers3.txt file to 'centers' directory -MUST be done to run next iteration.--
mv centers3.txt centersMedian/

#Rename 4th MR output, currently part-00000 to centers4.txt
mv part-00000 centers4.txt
cat centers4.txt



```
[ec2-user@ip-172-31-84-48 ~]$ cat centers4.txt
0.21352473348022605,0.7491620892344542,0.319142215087994,0.684360063363121,0.4241797646012576
0.4183829233768886,0.29190046446034734,0.5603292628208043,0.31856665150125013,0.19252053688196935
0.33329561470209124,0.2379536934808978,0.6843079387790536,0.5212091850640115,0.7851412593035395
0.6726933829413886,0.7567380517898474,0.6393392716490078,0.28572665779894557,0.6582653664747351
0.8082445533703188,0.4445871701391184,0.2790322012795942,0.7264406126141392,0.4491017571345998
```

#Move centers4.txt file to 'centers' directory.
mv centers4.txt centersMedian/

#Go to 'centers' directory, all centroid outputs are store here.
cd centersMedian
ls

#Get latest centroid output.
cat centers4.txt

#Go back to root directory to run next problem/part.
cd

Total Time: 215.941 seconds, approximately 3.35 mins

Using MapReduceChain.java: Full script in Kmedian_MapReduceChain.java.

Note: Same logic as Kmeans_MapReduceChain.java. You will see exceptions raise when running the code, it should still give correct output.

#Run python file again to regenerate initial centroid.
#Since I am using seed, it should generate the same file.
python generateCenters.py

vim MapReduceChain.java

-- ESC+gg+dG—if needed to replace content in previous step. --

javac -cp ~/hadoop-streaming-2.6.4.jar:$(hadoop classpath) MapReduceChain.java

jar -cf MapReduceChain.jar MapReduceChain.class

jar xvf MapReduceChain.jar META-INF/MANIFEST.MF nano META-INF/MANIFEST.MF

-- Paste in: Class-Path: /home/ec2-user/hadoop-streaming-2.6.4.jar --

jar cmf META-INF/MANIFEST.MF MapReduceChain.jar MapReduceChain.class

time hadoop jar MapReduceChain.jar MapReduceChain

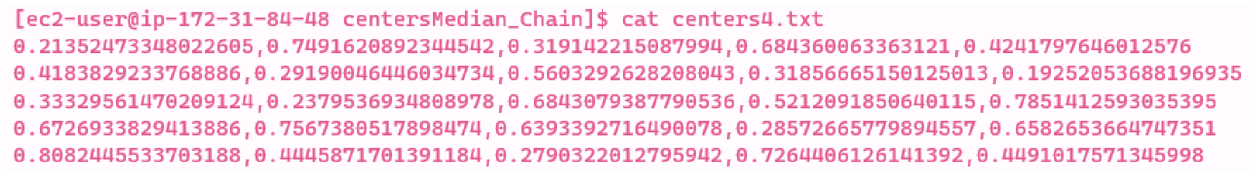
#Download 4th MR output to local machine.
hadoop fs -get /Part1_Output4_Median/part-00000

#Rename 4th MR output, currently part-00000 to centers4.txt
mv part-00000 centers4.txt

#Move centers4.txt file to 'centersMedian_Chain' directory.
mv centers4.txt centersMedian_Chain/

#Go to centersMeans_Chain directory. All centers data should be there.
cd centersMedian_Chain
ls

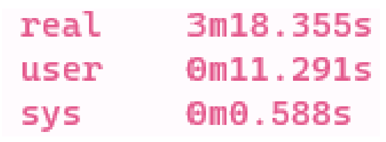
#Get latest centroid output.
cat centers4.txt



```
[ec2-user@ip-172-31-84-48 centersMedian_Chain]$ cat centers4.txt
0.21352473348022605,0.7491620892344542,0.319142215087994,0.684360063363121,0.4241797646012576
0.4183829233768886,0.29190046446034734,0.5603292628208043,0.31856665150125013,0.19252053688196935
0.33329561470209124,0.2379536934808978,0.6843079387790536,0.5212091850640115,0.7851412593035395
0.6726933829413886,0.7567380517898474,0.6393392716490078,0.28572665779894557,0.6582653664747351
0.8082445533703188,0.4445871701391184,0.2790322012795942,0.7264406126141392,0.4491017571345998
```

#Go back to root directory to run next problem/part.
cd

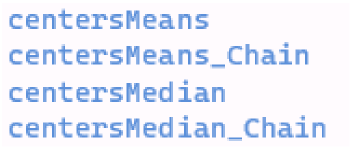
Time:



```
real    3m18.355s
user    0m11.291s
sys     0m0.588s
```

If you run everything both ways, at this step there should be 4 folders at root directory.

#Get folders name.
ls



```
centersMeans
centersMeans_Chain
centersMedian
centersMedian_Chain
```

Part 2) Two-Step MapReduce

Pre-requisite steps: Since I already been doing homework with sample data from 3 tables: dwdate, lineorder, and part. Took me times to extract and successfully utilize. I want to make sure table data from http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/ have the same schema/information as the ones we have been using in class, so I can continue using those sample data.

---OPTIONAL TO RUN THIS STEP---

```
#Get provided schema and data for final exam.
wget http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/
#Look up content.
cat index.html
#Get each table within SSBM1 directory.
wget http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/part.tbl
wget http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/lineorder.tbl
wget http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/dwdate.tbl

#Cross check that these table's content is identical to tables' we have been working with.
cmp -s part.tbl part.tbl.1 && echo "Files are identical" || echo "Files are different"
    ➔ Files are identical
cmp -s lineorder.tbl lineorder.tbl.1 && echo "Files are identical" || echo "Files are different"
    ➔ Files are identical
cmp -s dwdate.tbl dwdate.tbl.1 && echo "Files are identical" || echo "Files are different"
    ➔ Files are identical


#Create HDFS directory 'data'. If already run Part1, directory already exists.
hadoop fs -mkdir /data

#Copy three tables to HDFS for processing
hadoop fs -put lineorder.tbl /data
hadoop fs -put part.tbl /data
hadoop fs -put dwdate.tbl /data
```

Note: If you want to copy my code to run Hadoop job, please open FinalExam-CSC555.py/ .pynb/ .html file.

By breaking down the query, it makes sense for me to start writing code.

2 Map Reduce Phases	
First MapReduce Job **Join of Lineorder & Part tables**	**Second MapReduce Job** **Map-side join with Dwdate table and Groupby**
1st Mapper	**2nd Mapper**
✍ Extract relevant columns from lineorder ✍ Extract relevant column from part tables, tackle p_brand1 = 'MFGR#2123'	✍ Extract relevant columns from dwdate, tackle d_sellingseason = 'Fall' ✍ Join output from 1st MapReduce with dwdate table based on lo_orderdate = d_datekey
1st Reducer	**2nd Reducer**
✍ Join lineoder and part table based on lo_partkey = p_partkey	✍ Perform aggregation using group by d_year, p_category

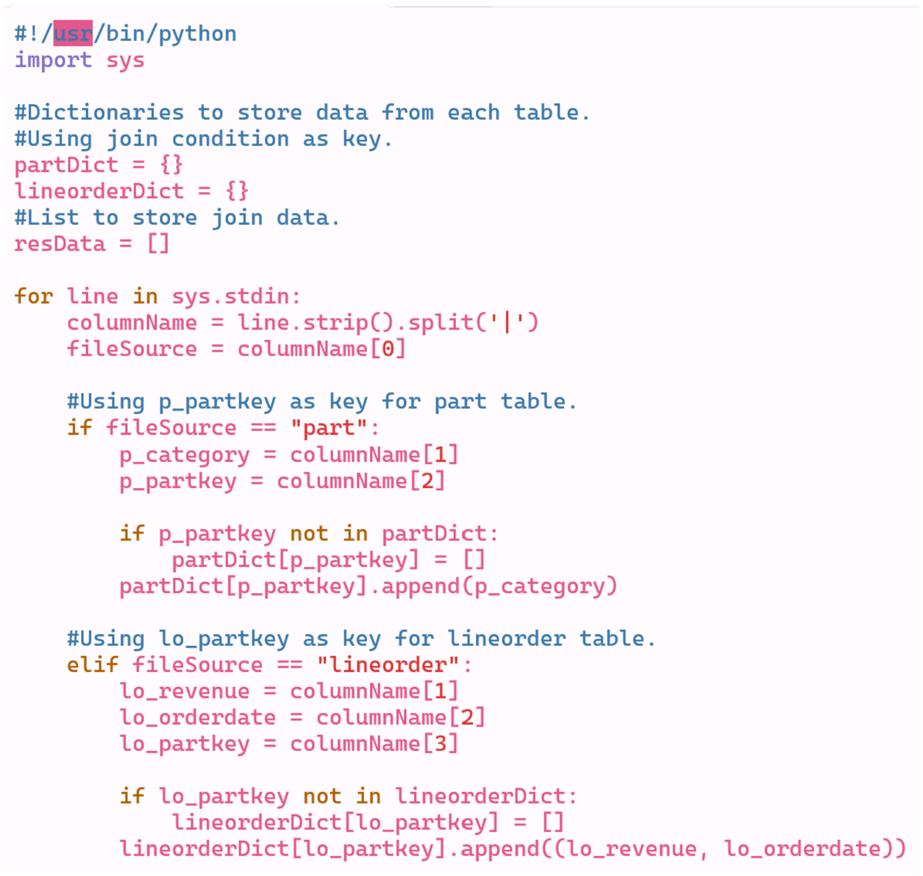
Mapper 1:

#Create 1st Mapper.
vim part2_mapper1.py

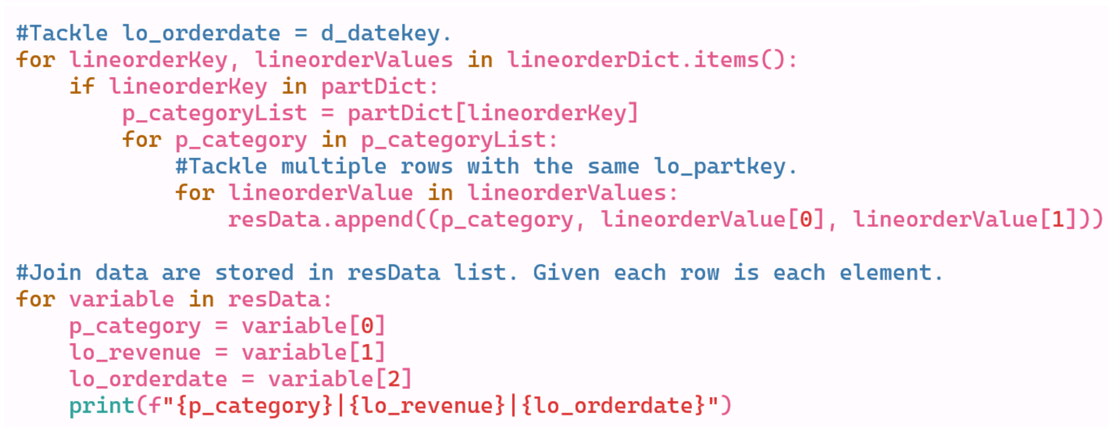
```
#!/usr/bin/python
import sys
for line in sys.stdin:
    columnName = line.strip().split('|')
    #Extract relevant columns in lineorder table.
    if columnName[1].isdigit():
        lo_revenue = columnName[12]
        lo_orderdate = columnName[5]
        lo_partkey = columnName[3]
        print(f"lineorder|{lo_revenue}|{lo_orderdate}|{lo_partkey}")

    #Extract relevant columns in part table.
    #Tackle condition p_brand1 = 'MFGR#2123'.
    else:
        p_category = columnName[3]
        p_partkey = columnName[0]
        p_brand1 = columnName[4]
        if p_brand1 == 'MFGR#2123':
            print(f"part|{p_category}|{p_partkey}")
```

Reducer 1:

#Create 1st Reducer
vim part2_reducer1.py



```python
#!/usr/bin/python
import sys

#Dictionaries to store data from each table.
#Using join condition as key.
partDict = {}
lineorderDict = {}
#List to store join data.
resData = []

for line in sys.stdin:
    columnName = line.strip().split('|')
    fileSource = columnName[0]

    #Using p_partkey as key for part table.
    if fileSource == "part":
        p_category = columnName[1]
        p_partkey = columnName[2]

        if p_partkey not in partDict:
            partDict[p_partkey] = []
        partDict[p_partkey].append(p_category)

    #Using lo_partkey as key for lineorder table.
    elif fileSource == "lineorder":
        lo_revenue = columnName[1]
        lo_orderdate = columnName[2]
        lo_partkey = columnName[3]

        if lo_partkey not in lineorderDict:
            lineorderDict[lo_partkey] = []
        lineorderDict[lo_partkey].append((lo_revenue, lo_orderdate))
```



```python
#Tackle lo_orderdate = d_datekey.
for lineorderKey, lineorderValues in lineorderDict.items():
    if lineorderKey in partDict:
        p_categoryList = partDict[lineorderKey]
        for p_category in p_categoryList:
            #Tackle multiple rows with the same lo_partkey.
            for lineorderValue in lineorderValues:
                resData.append((p_category, lineorderValue[0], lineorderValue[1]))

#Join data are stored in resData list. Given each row is each element.
for variable in resData:
    p_category = variable[0]
    lo_revenue = variable[1]
    lo_orderdate = variable[2]
    print(f"{p_category}|{lo_revenue}|{lo_orderdate}")
```

Mapper 2: #Create 2nd Mapper - vim part2_mapper2.py

```
#!/usr/bin/python
import sys
import os

#Dictionary to store data for dwdate table.
dwdateDict = {}

with open('dwdate.tbl', 'r') as cacheFile:
    for line in cacheFile:
        columnName = line.strip().split('|')
        d_datekey = columnName[0]
        d_year = columnName[4]
        d_sellingseason = columnName[12]

        #Tackle d_sellingseason = 'Fall'.
        if d_sellingseason == 'Fall':
            #Incase there are multiple d_year associated with one d_datekey.
            if d_datekey not in dwdateDict:
                dwdateDict[d_datekey] = []
            dwdateDict[d_datekey].append(d_year)

#Tackle output from 1st MapReduce.
for line in sys.stdin:
    columnName = line.strip().split('|')

    p_category = columnName[0]
    lo_revenue = columnName[1]
    lo_orderdate = columnName[2]

    #Tackle lo_orderdate = d_datekey.
    if lo_orderdate in dwdateDict:
        d_yearList = dwdateDict[lo_orderdate]
        for d_year in d_yearList:
            print(f"{d_year}|{p_category}|{lo_revenue}")
```

Reducer 2: #Create 2nd Reducer - vim part2_reducer2.py

```
#!/usr/bin/python
import sys

#Initiate counters to calculate sum and respective total count.
revSum = 0
revCount = 0
#Dictionary to store aggregated values.
joinDict = {}

print('sum(lo_revenue)|count(*)|d_year|p_category')

for line in sys.stdin:
    columnName = line.strip().split('|')
    d_year = columnName[0]
    p_category = columnName[1]
    lo_revenue = int(columnName[2])

    #Tackle group by d_year, p_category.
    #Stored as tuple for dictionary key.
    if (d_year, p_category) not in joinDict:
        joinDict[(d_year, p_category)] = {'revSum': 0, 'revCount': 0}

    #Keep appending.
    joinDict[(d_year, p_category)]['revSum'] += lo_revenue
    joinDict[(d_year, p_category)]['revCount'] += 1

#For each item, sum revenue output is already calculated here.
for keys, values in joinDict.items():
    print(f"{values['revSum']}|{values['revCount']}|{keys[0]}|{keys[1]}")
```

Using Hadoop Streaming separated commands:

1st MR: time hadoop jar hadoop-streaming-2.6.4.jar -input /data/lineorder.tbl,/data/part.tbl -output /data/Part2_MR1_Output -mapper part2_mapper1.py -reducer part2_reducer1.py -file part2_mapper1.py -file part2_reducer1.py

```
23/11/21 16:11:27 INFO streaming.StreamJob: Output directory: /data/Part2_MR1_Output

real    0m41.163s
user    0m3.864s
sys     0m0.327s
```

2nd MR: time hadoop jar hadoop-streaming-2.6.4.jar -input /data/Part2_MR1_Output/part-00000 -output /data/Part2_MR2_Output -mapper part2_mapper2.py -reducer part2_reducer2.py -file part2_mapper2.py -file part2_reducer2.py -file dwdate.tbl

```
23/11/21 16:12:24 INFO streaming.StreamJob: Output directory: /data/Part2_MR2_Output

real    0m17.957s
user    0m3.681s
sys     0m0.354s
```

Total Time: 59.12 sec

Using Part2_MapReduceChain.java: Full script in Part2_MapReduceChain.java.

```
JobControl jobControl = new JobControl( "Part2");

String[] job1Args = new String[]
{
    "-mapper"   , "part2_mapper1.py",
    "-reducer"  , "part2_reducer1.py",
    "-input"    , "/data/lineorder.tbl",
    "-input"    , "/data/part.tbl",
    "-output"   , "/Part2_MR1_Output/",
    "-file"     , "part2_mapper1.py",
    "-file"     , "part2_reducer1.py"
};
JobConf job1Conf = StreamJob.createJob(job1Args);
Job job1 = new Job(job1Conf);
jobControl.addJob(job1);

String[] job2Args = new String[]
{
    "-mapper"   , "part2_mapper2.py",
    "-reducer"  , "part2_reducer2.py",
    "-input"    , "/Part2_MR1_Output/part-00000",
    "-output"   , "/Part2_MR2_Output/",
    "-file"     , "part2_mapper2.py",
    "-file"     , "part2_reducer2.py",
    "-file"     , "dwdate.tbl"
};
JobConf job2Conf = StreamJob.createJob(job2Args);
Job job2 = new Job(job2Conf);
job2.addDependingJob(job1);
jobControl.addJob(job2);
```

vim MapReduceChain.java

-- ESC+gg+dG—if needed to replace content in previous step. --

javac -cp ~/hadoop-streaming-2.6.4.jar:$(hadoop classpath) MapReduceChain.java

jar -cf MapReduceChain.jar MapReduceChain.class

jar xvf MapReduceChain.jar META-INF/MANIFEST.MF

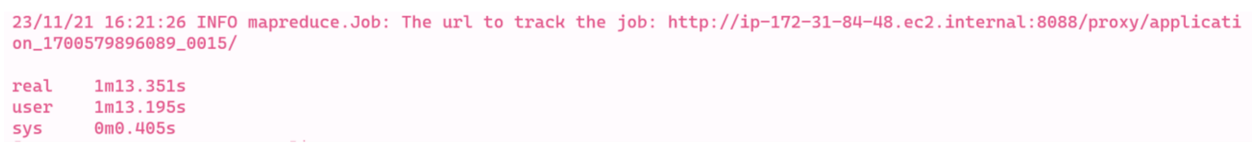
nano META-INF/MANIFEST.MF

-- Paste in: Class-Path: /home/ec2-user/hadoop-streaming-2.6.4.jar --

jar cmf META-INF/MANIFEST.MF MapReduceChain.jar MapReduceChain.class

time hadoop jar MapReduceChain.jar MapReduceChain

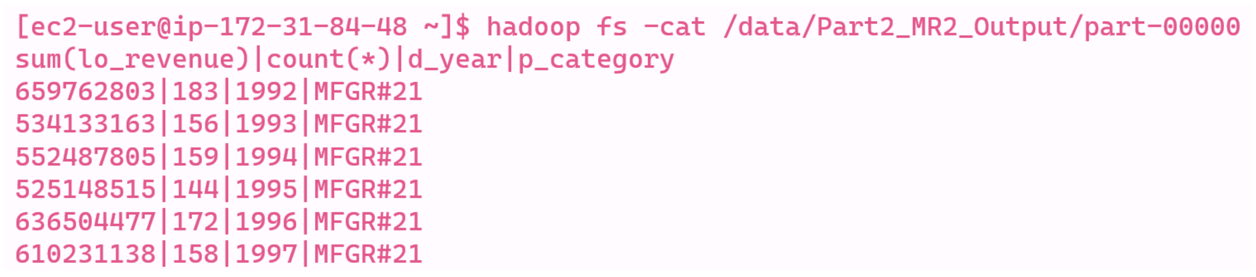
Time:



```
23/11/21 16:21:26 INFO mapreduce.Job: The url to track the job: http://ip-172-31-84-48.ec2.internal:8088/proxy/applicati
on_1700579896089_0015/

real    1m13.351s
user    1m13.195s
sys     0m0.405s
```

Output: 6 rows excluding header.

- Using Hadoop streaming: hadoop fs -cat /data/Part2_MR2_Output/part-00000
- Using MapReduceChain.java: hadoop fs -cat /Part2_MR2_Output/part-00000



```
[ec2-user@ip-172-31-84-48 ~]$ hadoop fs -cat /data/Part2_MR2_Output/part-00000
sum(lo_revenue)|count(*)|d_year|p_category
659762803|183|1992|MFGR#21
534133163|156|1993|MFGR#21
552487805|159|1994|MFGR#21
525148515|144|1995|MFGR#21
636504477|172|1996|MFGR#21
610231138|158|1997|MFGR#21
```