* **Download** the activity-data from canvas and upload it to your S3 bucket
* Using pySpark, **load the data** as Spark structured streaming data
* Estimate (grouped by "gt") the “gt” count and “x”, “y”, and ”z” **mean**values in a loop with various tumble window sizes (5 min, 15 min, 30 min) and append those to the results table.
  + You can pick maxFilesPerTrigger as 5
  + Please continue querying until you have read the entire data (try to come up with a reasonable sleep value and loop range)
  + Try (not required for submission) to plot a chart with these time varying values and the static values and think of the trend.

Code:

static = spark.read.json("s3://671sparkzhua273/activity-data/data/")

static.take(2)

dataSchema = static.schema

path = "s3://671sparkzhua273/activity-data/data/"

streaming = spark.readStream.schema(dataSchema).option("maxFilesPerTrigger", 5).json(path)

withEventTime = streaming.selectExpr("\*","cast(cast(Creation\_Time as double)/1000000000 as timestamp) as event\_time")

from pyspark.sql.functions import window, col

activityMeans = withEventTime.groupBy(window(col("event\_time"), "5 minutes"),"gt")\

.agg({'gt':'count', 'x':'avg', 'y':'avg', 'z':'avg'})

# QueryQ1.stop()

QueryQ1 = activityMeans\

.writeStream.queryName("pyevents\_per\_window").format("memory")\

.option("checkpointLocation","s3://671sparkzhua273/activity-data/test13").outputMode("complete").start()

from time import sleep

for x in range(19):

spark.sql("SELECT \* FROM pyevents\_per\_window").show()

sleep(15)

( I then change the window to 15min, 30 min and rerun the code. All of them took around 19 loops with 15s sleep to reach the final result )

Text, table

Description automatically generated

5 min window

A screenshot of a computer

Description automatically generated with low confidence

15 min window

A screenshot of a computer

Description automatically generated with low confidence

30 min window

* For tumble window of 15 min:
  + If during any tumble window “sit” count is more than “stand” count, print “standing recommended”

Code:

# Q2(A) standing recommendation

from pyspark.sql import functions as F

from pyspark.sql.window import Window

t1 = static.selectExpr("\*","cast(cast(Creation\_Time as double)/1000000000 as timestamp) as event\_time")

df = t1.groupBy(window(col("event\_time"), "15 minutes"),"gt").agg({'gt':'count', 'x':'avg', 'y':'avg', 'z':'avg'})

my\_window = Window.partitionBy('window').orderBy(["gt"])

dff = df.filter((df.gt=="sit")|(df.gt=="stand"))\

.withColumn("prev\_value", F.lag('count(gt)').over(my\_window))\

.select("window","gt","count(gt)","prev\_value")

dff = dff.withColumn("count", F.count('gt').over(Window.partitionBy('window')))

dff = dff.withColumn("diff", F.when(F.isnull(col('count(gt)') - col('prev\_value')), 1)\

.otherwise(col('count(gt)') - col('prev\_value')))

dff = dff.withColumn("recommand\_standing", F.when(col('diff')<0, "yes")\

.otherwise(F.when( (col('count')==1) & (col('gt')=="sit"), "yes")\

.otherwise("no")))

w2 = Window.partitionBy("window").orderBy(col("gt").desc())

Table

Description automatically generated

dff.withColumn("row",F.row\_number().over(w2))\

.filter(col("row") == 1).drop("row")\

.select("window","recommand\_standing").show()

* + If the average distance moved [sqrt((x2-x1)2+(y2-y1)2+(z2-z1)2)] during any two consecutive tumble windows is smaller than previous two tumble windows print “move recommended”
    - For simplicity, you can compute the distance between the mean values of x, y, z from one window to the next (versus taking distance for each user)

Code:

# Q2(B) moving recommendation

df2 = df.groupBy("window").agg({'avg(x)':'avg', 'avg(y)':'avg', 'avg(z)':'avg'}).sort("window")

dff2 = df2.withColumn("prev\_x", F.lag('avg(avg(x))').over(Window.partitionBy().orderBy(["window"])))

dff2 = dff2.withColumn("prev\_y", F.lag('avg(avg(y))').over(Window.partitionBy().orderBy(["window"])))

dff2 = dff2.withColumn("prev\_z", F.lag('avg(avg(z))').over(Window.partitionBy().orderBy(["window"])))

dff2 = dff2.withColumn("dist", F.sqrt(F.pow(col("avg(avg(x))")-col("prev\_x"),2) +\

F.pow(col("avg(avg(y))")-col("prev\_y"),2) +\

F.pow(col("avg(avg(z))")-col("prev\_z"),2)))

dff2 = dff2.withColumn("prev\_dist", F.lag('dist').over(Window.partitionBy().orderBy(["window"])))

dff2 = dff2.withColumn("recommand\_moving", F.when(col('dist')<col('prev\_dist'), "yes")\

.otherwise("no"))

Table

Description automatically generateddff2.select("window","recommand\_moving").show()