# DSCI 551 – Spring 2022

HW5 (Hadoop MapReduce & Spark), 100 points

Due: April 22, Friday (end of day, 11:59pm)

In this homework, we will consider the churn data set again (as in hw1). You are given two versions of the file: churn4hadoop.csv and churn.csv. The former has not header, to be used for Hadoop question below; the latter has header used in Spark.

1. [Hadoop MapReduce, 40 points] Complete the provided Churn.java by supplying the missing code as indicated in the source file, so that it answers the following SQL query.

   toks[8] toks[5]

   Select InternetService, max(tenure)

   From Churn

   Where churn = "Yes" → map

   Group by InternetService

   Having count(*) > 200; → reduce

   directory has churn4hadoop.csv

   Execution format: hadoop jar churn.jar Churn input output

   Where the input directory contains a single file: churn4hadoop.csv.

2. [40 points] For each of the following SQL queries, write a Spark script that finds the answer to the query. Note to read a csv file with header into Spark as a dataframe, proceed as follows:

   churn = spark.read.csv('churn.csv', header=True)

   You will also need to import this:

   import pyspark.sql.functions as fc

   a) select count(*)

   from churn

   where gender = 'Male' and churn = 'Yes';

   churn.filter("gender='Male' and churn='Yes'").count( )

   churn.rdd.filter(lambda r: r["churn"]=="Yes" and r["gender"]=="Male").count( )

   b) select gender, max(TotalCharges)

   from churn

   where churn = "Yes"

   group by gender;

   filter( )

   churn.rdd.map(lambda r: (r[gender], r[TC])).groupByKey( ).mapValues(lambda l: max(l)).collect

churn.filter("churn=='Yes'").groupby('gender').agg(fc.max('TotalCharges').alias('max_charge')), Note: you will need to change the data type of TotalCharges from string to double. For example,

churn = churn.withColumn('TotalCharges', fc.col('TotalCharges').cast('double'))

c) select gender, count(*)

from churn

where churn = 'Yes'

group by gender;

```
churn.filter("churn == 'Yes'").groupby('gender')
    .count()
```

d) select churn, contract, count(*) cnt

from churn

group by churn, contract

order by churn, cnt desc;

(churn is ascending)

```
churn.groupby(['churn','contract']).agg(fc.count("*").
    alias("cnt")).orderBy(['churn','cnt'],
    ascending=[True, False])
```

e) select gender, churn, count(*)

from churn

group by gender, churn

having count(*) > 1000;

```
churn.groupby(['gender','churn']).count().
 filter('count(*) > 1000').show()
```

3. [20 points] Write a Spark RDD script for each of the following SQL queries.

   a. Same as q2.a.

   b. Same as q2.b.

**Submission**:

- Q1: Churn.java and churn.jar and part-r-00000 under the output directory.
- Q2: submit a text file q2-solution.txt with your scripts and outputs from each script.
- Q3: submit a text file q3-solution.txt with your scripts and outputs from each script.