School of Computer Science

Student Name: Hassan Albujasim

Student Number : 1546488

Coding Project Report

Data Management and Big Data

abstract

This project demands a series of exercises for data management and manipulation. Our progress is judged based on the correctness and efficiency of our resolution of each problem, although correctness of answer is always accepted. The goal is to create the best possible solution.

The demands are largely based on laboratories carried out through the Autumn 2021 semester at UCD. The report will be broken to four sections, which include:

1. Bash & Data Management.

2. Hadoop Graph Processing.

3. Spark.

4. GraphX.

In this paper I will provide the answer to the question when requested, including code snippets or a reference to an attached script for larger files. The report will include a short description of the code and how it operates.

Part 1:

1.1 Bash & Data Management

check final pages for reference of full script.

1. There is an error in the name column - every name starts with an erroneous string "#]". Write a command/script to remove this string from the name column.

2. Currently the database is delineated by a "-" character. Write a command/script to convert this file into a comma delimited file.

3. There are two columns in this dataset with non-useful values. Write a script/command that identifies the columns that do not change, and removes them from the output.

4.Lastly, the country names in this dataset have been incorrectly added as country codes. Using the dictionary file provided (Link above) find and replace each country code with its correct country name.

Answer:

The script is called `clean_bash.sh`

Part 1&2:

~~~~~~

```
touch clean_bash.csv
while read line
do
      #part 1, part 2, part 3
      echo "$line" | sed 's/#]//' | sed 's/-/,/g' | sed ':a;s/^\(\([^"]*,\?\|"[^",]*",\?\)*"[^",]*\),/\1 /;ta' | tr -d ""

done < bashdm.csv > clean_bash.csv
```

~~~~~~

What happens is I loop through the file and write to a newly created one. Using sed and piping commands to change each line with first

- `echo "$line" | sed 's/#]//'`

Which replaces all "#]" with empty space.

- `| sed 's/-/,/g'`

This replaces all '-' with a comma ',' globally.

Finally to clean the double barrel names as they are replaced with commas too

- `| sed ':a;s/^\(\([^"]*,\?\|"[^",]*",\?\)*"[^",]*\),/\1 /;ta'`

This removes all commas from within quotes.

- `| tr -d ""`

And then remove quotes.

For part 3:

I create a list of the bad columns by checking for columns with word count of 1 which proves they are constant.

~~~~~~

```
num_cols=$(head -1 clean_bash.csv | sed -e 's/[^,]//g' | wc -c)
list=()
for ((i=1; i <= $num_cols; i++)); do
        if [[ $(cut -d, -f $i clean_bash.csv | sed '1d'| sort | uniq | wc -l) -eq 1 ]]
        then
                list+=($i)
        fi
done
```

~~~~~~

I use the list to cut them and return the compliment columns, and write them to an output array.

~~~~~~

```
badcol=" "
deli=' '
for item in "${list[@]}"; do
        badcol="$badcol$deli$item"
        deli=","
done
cut -d, -f$badcol --complement clean_bash.csv > output.csv
```

~~~~~~

Final Part 4:

this loops through two files and returns the desired country name from dictionary.csv. it creates a dictionary of col2 in the file with the value of the country name. Then we return this into col4 if col4 of our output file matches.

~~~~~~

```
awk -f"&|," 'fnr==nr{
if(nr>1) a[$2]=$3
next}
{if(fnr==1) next
        $4 = ($4 in a ? a[$4] : "not found") }
1' ofs=, dictionary.csv output.csv > last.csv

sed -i '1s/^/index,name,age,country,height,hair_colour\n/' last.csv
```

~~~~~~

The Final line adds the header back to the top as the awk skips it and removes it.

1.2 Data Management Tasks

1.Create 2 Bash scripts, one for SQL and one for MongoDB, that inserts your cleaned records into the respective database software. For SQL, you should generate the table within your Bash script.

My two scripts for this step are called: `sql_bash.sh` and `mongo_bash.sh`
I created my scripts locally on a linux machine and they do what the question asks quite simply with nothing fancy. For mySQL we had to create a table before inputting values whereas with noSQL mongodb it happens automatically as you add the values.

With SQL we also had to change “INDEX” to “indx” as it seems to be a key word and throws an error.

2&3. (SQL) Find the average height per country. Find the maximum height per hair colour

~~~~~~

```
#!/bin/bash
sudo mysql << EOF
USE mysql
USE people

select country, avg(height) from general_info group by country;
select hair_colour, max(height) from general_info group by hair_colour;
EOF
```

~~~~~~

The first result returns all the countries and their averages beside them and it is a long list. Trust me it works… try it? It begins like this.

~~~~~~

country	avg(height)
Malaysia	161.2500
Qatar	165.1667
Ireland	172.8571

~~~~~~

The answer to the second question/query looks like this.

~~~~~~

hair_colour	max(height)
White	200
Black	200
Dyed	199
Ginger	199
Blonde	200
Brown	200

~~~~~~

4. (MongoDB) Write a short script that adds a new characteristic to each person - an ID number. You may generate this number however you like (e.g. a counter) within your script

In this section I followed documentation by creating a counting object which takes the premade '_id: : "userid" ' for each mongodb entry and creates an auto incrementing sequence. Following is a snippet.

The rest of the program is the same as mongo_bash.sh, but we insert a new collection with whatever name (called 'id' here) and this creates a new table with auto incrementing objects with "findAndModify". The function is called on the "_id" as shown below.

~~~~~~

```
db.counters.insert( { _id: "userid", seq: 0 } )

function getNextSequence(name) {
var ret = db.counters.findAndModify(
{ query: { _id: name }, update: { $inc: { seq: 1 } }, new: true } ); return ret.seq; }

db.id.insert({_id: getNextSequence("userid"),
INDEX: """${var[0]}""",
```

~~~~~~

5. (MongoDB) Find the name of the person with the lowest value for height.

~~~~~~

```
#!/bin/bash
mongo 127.0.0.1/people --eval 'db.id.aggregate(
 [ { $group: { _id: "$Name", Height: { $min: "$Height" } } }, {"$sort": { "Height": 1 } },
{"$limit":1}] )'
```

~~~~~~

This finds the names of the people and sorts them by height and takes 1. Which is the person with the lowest height. Answer will look like this:
{ "_id" : "Ryan Hall", "Height" : "139" }

2. Simple Hadoop Graph Processing

check final pages for reference of full script.

1.Create a list of the number of routes that connect to each harbour

2. Route "Wolfsbane Nine" is associated with only one harbour - what is it?

3. What harbours are connected by route Carnation Sixty-seven(No.1223).

4. Which harbours fielded emergency routes - these are routes whose route number begins with "911"

5. "Midnightblue-Epsilon" is connected to two other harbours by a route- what are they?

Answers:

1. Here we basically do a harbour count. We take the column we want using an iterator to break the line of the csv and count index 0. The harbour names.

~~~~~~

```
while (itr.hasMoreTokens()) {
        String[] split = itr.nextToken().split(",");
        key1.set(split[0].trim());
        context.write(harbour, one);
}
```

~~~~~~

2. Here we use the same logic but we just filter for “Wolfsbane Nine” first then return the harbour

~~~~~~

```
while (itr.hasMoreTokens()) {
        String[] split = itr.nextToken().split(",");
        if (split[2].trim().equals("Wolfsbane_Nine")){
                key1.set(split[0].trim());
                context.write(key1, one); }
}
```

~~~~~~

3. Again, we filter for the Route “Carnation_Sixty-seven” and we count for the harbours.

~~~~~~

```
        while (itr.hasMoreTokens()) {
                String[] split = itr.nextToken().split(",");
                if (split[2].trim().equals("Carnation_Sixty-seven")){
                        key1.set(split[0].trim());
                 context.write(key1, one);
                }
        }
```

~~~~~~

4. We filter for Routes starting with “911” and we count for the harbours.

~~~~~~

```
        while (itr.hasMoreTokens()) {
                String[] split = itr.nextToken().split(",");
                if (split[3].trim().startsWith("911")){
                        key1.set(split[0].trim());
                 context.write(key1, one);
                }
        }
```

~~~~~~

5. Here we emit the harbours and their route number together.

Port	RouteNo
A	1
B	2
C	1
D	2
E	1

Then we reduce, to connect the RouteNo to the ports associated with them by joining values. Making something like this

routeNo	ports
1	A || C || E
2	B || D

Then we filter for just “Midnightblue-Epsilon”
Script is at the very back.
My initial attempt followed the same logic as before but that only worked for our question, because “Midnightblue-Epsilon” came before the others in the csv. The script looked like this. Where it does a similar thing by filtering for Midnight port first, then takes its route as key to map the other connections. This will also be included and called “Connecting2.java”

~~~~~~

```
while (itr.hasMoreTokens()) {
        String[] split = itr.nextToken().split(",");
        if (split[0].trim().equals("Midnightblue-Epsilon")){
                route = new String(split[2].trim());
        }
        if(split[2].trim().equals(route)){
                key1.set(split[0].trim());
                context.write(key1, one);

        }
}
```

~~~~~~

3. Spark

1. Determine the number of records the dataset has in total.

2. Find the restaurant with the highest number of reviews.

3. Determine the restaurant with the longest name.

4. Find the number of reviews for each region.

5. Determine the most frequently occurring term in the review column that doesn’t include one of the following stop words: ”A”, ”The”, ”of”

Answers:

1. `inputFile.count` will return the number of records.

2. `reviewsDf.select("Restaurant").where(reviewsDf("`No.Reviews`") === reviewsDf.agg(max("`No.Reviews`")).first()(0)).show()` This returns the highest rated restaurant. With agg(max(reviews) for a restaurant and we return just index 0.

3. `val longRest = spark.sql("Select Restaurant from reviewsDf WHERE length(Restaurant) = (SELECT max(length(Restaurant)) FROM reviewsDf)").show()` simple spark sql that does what the question asks. Return the longest name for restaurants.

4. `val regionRev = reviewsDf.groupBy('Region).sum("`No.Reviews`").show()` again just sum all reviews and group by region.

5. First we take the fifth column which is the only one we wanna work with.
   Then in spark we import stop word remover and remove the words included.
   Transform the rdd and remove the stop words with our stopremover function.
   Check it worked. Make a copy of the file, map reduce to find the other stop words.

   `sparkrdd.flatMap(line => line.split(",")(0).split(" ")).map(word => (word, 1)).reduceByKey( + _).sortBy(T => T._2,false).first()`

   Answer should be (String, Int) = (and,379).

4. GraphX

1. Import the data and create a graph representing the data

2. Generate an array of each harbour's connected routes - consult the spark documenta-tion to identify the most suitable method for this. Alternatively, you may use Spark commands to generate this information.

3. Which harbour(s) is/are served by route "Porium Thirty-one"?

4. Which harbour has the most routes associated with it. If there are multiple harbours with the same number of routes, list them all.

5. Which harbour is connected to the most other harbours - for this, you can transform your earlier outputs, or use a new GraphX method. If there are multiple harbours with the same number of connections, list them all. You may find the above edge data useful!

For my sanity I actually ran this with a jupyter notebook for scala because finding the commands after or changing them proved very difficult.
I did this with ``docker pull jupyter/all-spark-notebook`` then ``docker run -p 8888:8888 jupyter/all-spark-notebook:latest``
I would clean the files locally on my machine and upload them to do the work with the notebook. All the commands are in the README.txt and cleaned files are with it.
Answers:

1. This was the hardest part of the task. We had to use both data sets. We created the harbours(vertices) with the first and the routes (edges) with the other. The readme describes the steps very clearly. We had to use both because we needed to create the from, to relationship between the vertices by creating a mapper function to tie the port name to its number from the second csv. Then we can create the graph.
2. This is simply a collectEdges() command for either direction. Answer is:

res4: Array[(org.apache.spark.graphx.VertexId,
Array[org.apache.spark.graphx.Edge[String]])] =
Array((8996,Array(Edge(6797,8996,Mimosa_Three_hundred_and_twenty-one),
Edge(8961,8996,Mimosa_Three_hundred_and_twenty-one), Edge(8996,0,Safflower_Two),
Edge(8996,6797,Mimosa_Three_hundred_and_twenty-one),
Edge(8996,8961,Mimosa_Three_hundred_and_twenty-one))))

3. This is another quite simple command where we look for the origin, destination and routes and filter by the route name has "Porium_Thirty-one"
    - graph.edges.filter { case (Edge(origin, destination, route))=> route == "Porium_Thirty-one"}.foreach(println)``

4. Simply, define a reduce operation to compute the highest degree vertex
    - def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
      if (a._2 > b._2) a else b
      }
    - val maxDegrees: (VertexId, Int) = graph.degrees.reduce(max)

   Answer: maxDegrees: (org.apache.spark.graphx.VertexId, Int) = (0,373)
   Seems nowhere is a popular place to come and go from.
5. Last one we collect the neighbours going either way, in or out.
    - graph.collectNeighbors(EdgeDirection.Either).take(1)
    - val neigbours = graph.collectNeighborIds(EdgeDirection.In).sortBy(_._1, ascending=false)

   Then we Map:
    - val lenMap = neigbours.map { case (id, neighbours) => (id, neighbours.distinct.length) }

   The id of the harbour and length of their distinct neighbours.
   and sort by the length.

    - lenMap.sortBy(_._2, ascending=false).take(3)

   Result for the above operations is :
    - res14: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((6176,9), (8930,7), (7987,7))

BIG SCRIPTS FOR PART 1 - Data Management:

Name : `sql_bash.sh`

~~~~~~

```
#!/bin/bash
sudo mysql << EOF
USE mysql
create database IF NOT EXISTS people;
USE people;

CREATE TABLE IF NOT EXISTS general_info(indx SMALLINT NOT NULL
,name VARCHAR(50) NOT NULL
,age SMALLINT NOT NULL
,country VARCHAR(50) NOT NULL
,height INT(10) NOT NULL
,hair_colour VARCHAR(50) NOT NULL)
EOF

# insert all values
input="last.csv" sed 1d $input | while IFS=, read -r indx name age country height hair_color

do sudo mysql -h localhost -D "people" -e "insert into general_info values('$indx', '$name', '$age',
'$country', '$height', '$hair_color')"
done
```

~~~~~~

Name : `mongo_bash.sh`

~~~~~~

```
#!/bin/bash

input="last.csv"

sed 1d $input | while IFS="," read -r -a var

do

mongo 127.0.0.1/people --eval 'db.info.insert({INDEX: """${var[0]}"""
,Name: """${var[1]}"""
,Age: """${var[2]}"""
,Country: """${var[3]}"""
,Height: """${var[4]}"""
,Hair_Colour: """${var[5]}"""})'

done
```

~~~~~~

BIG SCRIPTS FOR PART 2. Hadoop Q5: called “Connecting.java”

~~~~~~

```
public class Connecting {
```
~~~~~~

```
	public static class TokenizerMapper extends Mapper<Object, Text, Text, Text> {

	private Text routeNo = new Text();
	private Text port = new Text();

	public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
	StringTokenizer itr = new StringTokenizer(value.toString());
	while (itr.hasMoreTokens()) {

	StringTokenizer itr = new StringTokenizer(value.toString());

	routeNo.set(split[3].trim());

	port.set(split[0].trim());

	context.write(routeNo, port);

	}
	}
	}

	public static class PortsPerRouteReducer extends Reducer<Text, Text, Text, Text> {
	private Text result = new Text();

	public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
	// empty set
	Set<String> ports = new HashSet<>();

	for (Text port : values) {
	ports.add(port.toString());
	}

	String portString = String.join("|| ", ports);

	//filter output for only one we want
	if (portString.contains("Midnightblue-Epsilon")) {
		// write to context
	result.set(portString);
	context.write(key, result);
	}
	}
	}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```