Project 2

Coding Assessment

Jason Ballantyne
13432788

COMP47470: Big Data Programming

UCD School of Computer Science

12/12/2021

Abstract

Each folder contains their respective scripts and a README.txt applying to that question, illustrating how to run the scripts.

The outputs from running the code have been supplied where the question explicitly asks for it.

Part 1.1 Clean-up Tasks

For this section all of the code is stored in scripts which can be found in the folder “Part 1.1” with an accompanying README.txt.

Q1 & Q2

```
#!/bin/bash

# Part 1.1
# This script deals with Q1 & Q2

touch cleaned.csv

while read line
do
    echo "$line" | sed 's/#]//' |  sed 's/-
/,/g' | sed ':a;s/^\(\([^"]*,\?\|"[^",]*",\?\)*"[^",]*\),/\1 /;ta' | tr -d '"'
done < bashdm.csv > cleaned.csv
```

Description:

While loop that reads each line.

sed 's/#]//' - Stream editor for filtering and transforming text, removes the #] and inserts nothing in its place.

The next command which is the longer sed statement does the following:

- :a; is a label for further branch
- The rest is split into 3 enclosed parts: first the 2nd: [^"]*,\?\|"[^",]*",\? match for a string containing no double quote, followed by a coma or a string enclosed by two double quote, without coma and followed by a coma.
- The next part is composed by as many repetitions of the previously described bullet point, followed by 1 double quote and some characters, but no double-quote, nor comas.
- ta will loop to :a if previous s/ command did any changes.

The tr command is used to perform operations like removing repeated characters or converting from upper to lowercase. In this case we use it to remove the quotation marks.

Q3

```
#!/bin/bash

# Part 1.1
# This script deals with Q3

non_unqiue_list=()
data=cleaned.csv

col_number=$(head -1 $data| sed -e 's/[^,]//g' | wc -c)
```

```
for ((i=1 ; i <= $col_number ; i++)); do
    unique_col=$(cut -f$i -d',' $data | tail -n +2 | sort | uniq -c | wc -l)
    if [[ $unique_col -eq 1 ]]; then
       non_unqiue_list+=($i)
    fi
done

IFS=\, eval 'non_unqiue_list="${non_unqiue_list[*]}"'
cut -d"," -f$non_unqiue_list --complement $data > "removed_columns.csv"
```

Description:

Uses the cleaned csv outputted from our original script. The variable col_number will calculate the number of columns. We then for loop over the file, starting at 1 up until the number of columns in the file. We check if each column has unique values (If there is only 1 unique value in the column, it is a constant column) and we add it to a list called non_unique_list. We then use the cut with complement function to print only fields that are not included in our non_unique_list to the outputted file called “removed_columns.csv”.

Q4

```
#!/bin/bash

# Part 1.1
# This script deals with Q4


awk -F"&|," 'FNR==NR{

if(NR>1) a[$2]=$3
next}

{if(FNR==2) next

 $4 = ($4 in a ? a[$4] : "Country") }
1' OFS=, countries.csv removed_columns.csv > fully_cleaned.csv
```

Description:

Using awk -F to set the field separator to either “&” or “,”. NR tells us the total number of records that we've read so far, while FNR gives us the number of records we've read in the current input file. If the value of NR is greater than 1 it will replace the matching “CODE” values with the corresponding “NAME” name from the countries.csv and send the output to a new csv labelled “fully_cleaned.csv”

Result:

INDEX	Name	Age	Country	Height	Hair_Colour
1	Anthony Gentry	49	Qatar	162	Black
2	Marcia Jones	75	Ireland	186	Dyed

This is a snippet of the “fully_cleaned.csv” (found within the “Part 1.1” folder) to illustrate that all four cleaning tasks have been carried out including:

1. The removal of the erroneous string ”#]”.
2. Changing to a comma delimited file.
3. Removal of the columns with non-useful values which included “YLA” & “CONF”.
4. Using the dictionary file to find and replace country code with country name.

Part 1.2 Data management Tasks

All code used in this section can be found in separate scripts in the folder “Part 1.2” with an accompanying README.txt. The outputs of running the code are shown for Q2, Q3, Q4 & Q5 as the questions explicitly ask for it.

Q1.

SQL Database Script:

```
#!/bin/bash
# Part 1.2 Q1
# Script that creates a database,a table and inserts our cleaned csv "fully_cleaned.csv" i
nto MySQL


mysql << EOF
USE mysql;
CREATE DATABASE IF NOT EXISTS People_Info;
USE People_Info;
CREATE TABLE IF NOT EXISTS Detailed_Description(INDEX_A SMALLINT NOT NULL, Name VARCHAR(25
5) NOT NULL,Age SMALLINT NOT NULL,Country VARCHAR(255) NOT NULL,Height SMALLINT NOT NULL,H
air_Colour VARCHAR(255) NOT NULL);
EOF

input="fully_cleaned.csv"
sed 1d $input | while IFS=, read -r INDEX_A Name Age Country Height Hair_Colour

do
mysql -D "People_Info" -
e "insert into Detailed_Description values('$INDEX_A','$Name','$Age','$Country','$Height',
'$Hair_Colour')"
done
```

Description:

If a database called “People_Info” does not already exist, it creates one (This is to allow the re-running of the script as it would otherwise give an error stating the database has already been created). If a table does not exist, it creates one called “Detailed_Description”. The same reasoning has been applied for using “IF NOT EXISTS”. We then create the structure of the database by determining column types.

The input variable is set as our cleaned dataset. “sed 1d” is used to remove the first line of our input file as we have already defined the column names. “IFS=,” is defining our internal field separator as “,”. We then insert our cleaned dataset into the associated columns.

MongoDB Database Script:

```
#!/bin/bash
# Part 1.2 Q1
# Script that inserts our cleaned csv "fully_cleaned.csv" into MongoDB


input="fully_cleaned.csv"

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

do
```

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

done
```

Description:

The input variable is set as our cleaned dataset. “sed 1d” is used to remove the first line of our input file. “IFS=,” is defining our internal field separator as “,”. Insert() is used next, if the collection does not exist, then the insert() method will create the collection. Finally, we are populating the collection by indexing the variable for each column.

Q2.

```
#!/bin/bash
# Part 1.2 Q2
# Script that selects the country & average height, groups by country and ordered by avera
ge height ascending

mysql << EOF
use People_Info;
Select Country, avg(Height) as AverageHeight from Detailed_Description Group by Country Or
der by AverageHeight ASC;

EOF
```

Description:

We use the database we created in our previous MySQL script and run a select query to get the average height per country. We select the country and the average height which we rename to ‘AverageHeight’. It is then grouped by country and finally ordered by average height in an ascending order for readability. The output of this can be seen below:

Country	AverageHeight
United_Arab_Emirates	144.0000
Greece	145.6667
Mayotte	148.0000
Northern_Mariana_Islands	151.5000
Guinea-bissau	152.0000
Honduras	152.0000
Uruguay	152.5000
Georgia	153.5000
Sri_Lanka	154.0000
Morocco	154.5000
Guyana	154.8000
Madagascar	155.0000
Namibia	155.0000
Denmark	155.6667
Tonga	157.0000
Micronesia	157.0000
Dominica	157.1667
Belgium	157.6667
Guinea	158.0000
Vanuatu	159.0000
Kuwait	159.0000
Lesotho	159.2500
Virgin_Islands_(U.S.)	159.5000
Dominican_Republic	159.6667
Andorra	160.5000
Jordan	160.5000
United_States	160.6000
Bhutan	160.8333
Eritrea	161.0000
Tanzania	161.0000
Netherlands_Antilles	161.1250
Angola	161.2500
Equatorial_Guinea	161.5000
Maldives	161.5000
Congo	161.7500
Iraq	162.0000
Brazil	162.0000
Kazakhstan	162.0000
Gibraltar	162.0000
Estonia	162.1667
Bangladesh	162.2500
Egypt	162.4000
Malaysia	162.6667
United_Kingdom	162.6667
Macedonia	163.0000
India	163.0000
Syrian_Arab_Republic	163.3333
Burkina_Faso	163.3333
Saudi_Arabia	163.4000
Trinidad_And_Tobago	163.5000
Togo	163.5000
Mauritius	163.6000
Switzerland	163.6667
Guadeloupe	163.7500
Hungary	164.0000
Cambodia	164.0000
Moldova	164.0000
Norway	164.2000
Zaire	164.3333
French_Polynesia	164.6000
Thailand	165.0000
Qatar	165.1667
Tajikistan	165.3333
Finland	165.5000
Cape_Verde	165.5000
Colombia	165.6667
Cocos_Islands	165.6667
Haiti	165.8000
El_Salvador	165.8333
Israel	166.0000
Mexico	166.0000
Ukraine	166.0000
Costa_Rica	166.3750
Bouvet_Island	166.4000
Niger	166.5000
Iran	166.5556
French_S._Territories	166.6000
Sweden	166.6667
Benin	166.6667
France	166.8000
Cameroon	167.0000
Kenya	167.2000
Kiribati	167.2500
Anguilla	167.4000
Singapore	167.7143
Turkey	167.7500
New_Zealand	168.0000
Christmas_Island	168.0000
Swaziland	168.0000
Uzbekistan	168.3333
St.Pierre	168.3333
Mali	168.5000
Seychelles	168.5000
Belize	168.5000
Gambia	168.5714
Ethiopia	168.6667
Korea_(South)	168.7143
Netherlands	168.7500
Tokelau	168.8000
Lebanon	168.8571
Liechtenstein	169.0000
Macau	169.0000
Guam	169.0000
Nepal	169.0000
Djibouti	169.1667
Slovakia	169.3333
Peru	169.3333
Nicaragua	169.8000
French_Guiana	169.8000
Senegal	169.8750
Botswana	170.1000
Luxembourg	170.1250
Papua_New_Guinea	170.2500
Canada	170.3333
Bulgaria	170.5000
Algeria	170.5000
Mauritania	170.6667
Yemen	170.6667
American_Samoa	170.7143
Bosnia_And_Herzegowina	170.7500
Nauru	170.8571
Philippines	171.0000
Belarus	171.0000
Malawi	171.0000
St._Helena	171.0000
Bermuda	171.0000
Sudan	171.0000
Greenland	171.2857
Poland	171.3333
New_Caledonia	171.5000
Saint_Kitts_And_Nevis	171.5000
Czech_Republic	171.6000
Turkmenistan	171.6667
Uganda	171.6667
Burundi	171.6667
Afghanistan	171.8000
Cuba	172.0000
Rwanda	172.0000
Somalia	172.2000
Falkland_Islands_(Malvinas)	172.2000
Cayman_Islands	172.3333
Suriname	172.3333
Pitcairn	172.6000
Ireland	172.8571
Malta	173.0000
Azerbaijan	173.0000
Ecuador	173.5000
San_Marino	173.5000
Martinique	173.5000
Zambia	173.5714
Bahamas	173.6667
Monaco	173.7500
Cote_D`ivoire	173.8333
China	174.0000
Antarctica	174.0000
Spain	174.0000
Bolivia	174.0000
Laos	174.0000
Solomon_Islands	174.2000
Antigua_And_Barbuda	174.3333
Nigeria	174.4000
Comoros	174.7500
Fiji	174.7500
Puerto_Rico	175.0000
Western_Sahara	175.0000
Sierra_Leone	175.1667
Barbados	175.2500
Niue	175.4000
Brunei_Darussalam	175.4286
Panama	175.5714
Marshall_Islands	175.6000
Australia	175.6667
Romania	175.6667
Zimbabwe	176.0000
Lithuania	176.3750
Korea_(North)	176.3750
Chile	176.5000
Slovenia	176.6667
Faroe_Islands	176.8000
Argentina	177.0000
South_Africa	177.0000
Oman	177.0000
Cook_Islands	177.0000
Japan	177.8333
Kyrgyzstan	178.0000
Chad	178.0000
Armenia	178.0000
Croatia	178.0000
Guatemala	178.1429
Mozambique	178.1667
Russian_Federation	178.2500
Cyprus	178.4000
Yugoslavia	178.5000
Grenada	178.5000
Libya	178.7143
Saint_Lucia	178.8000
Sao_Tome	178.8000
Gabon	179.0000
Paraguay	179.0000
Ghana	179.6250
Austria	180.0000
Austria	180.0000
Vatican_City_State	180.3333
Liberia	181.0000
Samoa	182.0000
Jamaica	182.0000
Hong_Kong	182.3333
Norfolk_Island	182.6667
Palau	182.7500
Virgin_Islands_(British)	183.0000
Mongolia	183.0000
St_Vincent/Grenadines	183.4000
Viet_Nam	183.5000
East_Timor	183.5000
Italy	183.6000
Germany	184.0000
Bahrain	184.7500
Central_African_Rep	185.6000
Indonesia	186.1667
Pakistan	186.3333
Montserrat	187.7500
Portugal	188.8000
Taiwan	189.6000
Iceland	189.6667
Tuvalu	190.6667
Venezuela	193.0000
Myanmar	193.0000
Latvia	194.0000
Reunion	199.0000

228 rows in set (0.00 sec)

Q3.

```
#!/bin/bash
# Part 1.2 Q3
# Script that selects the maximum height, calls it 'Max Height' & hair colour
# Groups the result by hair colour and orders by hair colour ascending

mysql << EOF
use People_Info;

Select MAX(Height) as 'Max Height',Hair_Colour from Detailed_Description Group by Hair_Col
our Order by Hair_Colour ASC;

EOF
```

Description:

Again, we use the database we created in our initial MySQL script and run a select query to find the maximum height per hair colour. We select the maximum height which we rename to 'Max Height' and select hair colour. It is then grouped by hair colour and ultimately ordered by hair colour in an ascending order for readability. The output can be seen below:

```
+------------+-------------+
| Max Height | Hair_Colour |
+------------+-------------+
|        200 | Black       |
|        200 | Blonde      |
|        200 | Brown       |
|        199 | Dyed        |
|        199 | Ginger      |
|        200 | White       |
+------------+-------------+
6 rows in set (0.00 sec)
```

Q4.

This is a snippet of the script as the script is too long; full script is contained in the folder "Part 1.2" called "Q4_MongoIDNumber.sh"

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

return ret.seq;
}
```

Description:

MongoDB reserves the _id field in the top level of all documents as a primary key. _id must be unique, and always has an index with a unique constraint. This script creates an incrementing sequence number for the _id field using counters collection. "sed 1d" is used to remove the first line of our input file. "IFS=," is defining our internal field separator as ",". We use a separate counters collection to track the last number sequence used. The _id field

contains the sequence name and the seq field contains the last value of the sequence. It currently stores the initial value for the userid.

The getNextSequence function accepts a name of the sequence. The function uses the findAndModify() method to automatically increment the seq value and return this new value. We then call the getNextSequence function during our insert to get the value for userid for our new collection. Please see the output below using find() to verify the results demonstrating the "_id" field before and after this function has been applied.

Before:

```
{ "_id" : ObjectId("61b3dec35f0def42cd0ec138"), "INDEX" : "1", "Name" : "Anthony Gentry", "Age" : "49", "Country" : "Qatar", "Height" : "162", "Hair_Colour" : "Black" }
{ "_id" : ObjectId("61b3dec3e5c42ec0ce253d68"), "INDEX" : "2", "Name" : "Marcia Jones", "Age" : "75", "Country" : "Ireland", "Height" : "186", "Hair_Colour" : "Dyed" }
```

After:

```
{ "_id" : 1, "INDEX" : "1", "Name" : "Anthony Gentry", "Age" : "49", "Country" : "Qatar", "Height" : "162", "Hair_Colour" : "Black" }
{ "_id" : 2, "INDEX" : "2", "Name" : "Marcia Jones", "Age" : "75", "Country" : "Ireland", "Height" : "186", "Hair_Colour" : "Dyed" }
```

Q5:

```
#!/bin/bash

# Part 1.2 Q5
# Script to return the name and height of the person with the lowest value for height


mongo 127.0.0.1/PeopleData --eval '
db.newDB.find({}, {Name:1, Height:1, _id:0}).sort({Height: 1}).limit(1)
'
```

Description:

This MongoDB command will find the name and height in our newDB collection. Due to the fact we have sorted height and used "1", this specifies an ascending order (-1 would be used for a descending order). We then limit the results by 1 to receive the name and height of the person with the lowest value for height. The output can be seen below:

```
{ "Name" : "Ryan Hall", "Height" : "139" }
```

Part 2 Simple Hadoop Graph Processing

For this section all of the code is stored in scripts which can be found in the folder "Part 2" with an accompanying README.txt.

The outputs of running the code are shown for Q2, Q3, Q4 & Q5 as the question explicitly asks for it.

Q1.

This is a snippet of the script; full script is contained in the folder "Part 2" called "HarbourQ1.java"

```
     // Creating a StringTokenizer
     StringTokenizer itr = new StringTokenizer(value.toString());
     while (itr.hasMoreTokens()) {
 // Displaying the Tokens
String[] split = itr.nextToken().split(",");
 // Splitting and returning the first column
Route.set(split[0].trim());

       context.write(Route, one);
     }
```

Description:

This class creates a list of the number of routes that connect to each harbour. StringTokenizer creates a new StringTokenizer. While hasMoreTokens() is true (method call returns 'true' if and only if there is at least one token in the string after the current position; false otherwise), it splits the tokens by “,” and stores them. We then return the first column.

Q2.

This is a snippet of the script; full script is contained in the folder “Part 2” called “HarbourQ2.java”

```
while (itr.hasMoreTokens()) {

String[] split = itr.nextToken().split(",");
 // if the route column is equaled to "Wolfsbane_Nine"
 // return the port column to find out the harbour
if (split[2].trim().equals("Wolfsbane_Nine")){
   Route.set(split[0].trim());
   context.write(Route, one);
}
```

Description:

While hasMoreTokens() is true (method call returns 'true' if and only if there is at least one token in the string after the current position; false otherwise), it splits the tokens by “,” and stores them. If the third column (“Route”) equals “Wolfsbane_Nine” (we use trim to eliminate leading and trailing spaces) then return the equivalent harbour in the first column. The output of the code can be seen below for the answer to the one harbour associated with “Wolfsbane_Nine”.

```
Mintcream-Tau
```

Q3.

This is a snippet of the script; full script is contained in the folder “Part 2” called “HarbourQ3.java”

```
while (itr.hasMoreTokens()) {

String[] split = itr.nextToken().split(",");

 // if column route equals Carnation_Sixty-seven
 // return the harbours connected to it
if (split[2].trim().equals("Carnation_Sixty-seven")){
   Route.set(split[0].trim());
   context.write(Route, one);
}
```

Description:

While hasMoreTokens() is true (method call returns 'true' if and only if there is at least one token in the string after the current position; false otherwise), it splits the tokens by “,” and stores them. If the third column (“Route”) equals “Carnation_Sixty-seven” (we use trim to eliminate leading and trailing spaces) then return the equivalent harbours in the first column. The output of running the code and the answer to what harbours are connected by route “Carnation_Sixty-seven” can be seen below:

```
Lightcoral-Pi   1
Seashell-Nu     1
```

Q4.

This is a snippet of the script; full script is contained in the folder "Part 2" called "HarbourQ4.java"

```
while (itr.hasMoreTokens()) {

String[] split = itr.nextToken().split(",");

 // if column route starts with 911
 // return the contents of the port column
if (split[3].trim().startsWith("911")){
   Route.set(split[0].trim());
   context.write(Route, one);
}
```

Description:

While hasMoreTokens() is true (method call returns 'true' if and only if there is at least one token in the string after the current position; false otherwise), it splits the tokens by "," and stores them. If the fourth column ("RouteNo") starts with "911" (we use trim to eliminate leading and trailing spaces) then return the equivalent harbours from the first column. The output of running the code and the answer to which harbours fielded emergency routes can be seen below:

```
Bisque-Mu       1
Burlywood-Epsilon       1
Darkkhaki-Zeta  1
Ghostwhite-Omicron      1
Goldenrod-Sigma 1
Mediumvioletred-Eta     1
Midnightblue-Rho        1
Sandybrown-Iota 1
Seashell-Zeta   1
```

Q5.

This is a snippet of the script; full script is contained in the folder "Part 2" called "HarbourQ5.java"

```
while (itr.hasMoreTokens()) {

   String[] split = itr.nextToken().split(",");
   // If the port column is equalled to Midnightblue-Epsilon
   // Store the contents of the route in the variable key_value
   if (split[0].trim().equals("Midnightblue-Epsilon")){
      key_value = new String(split[2].trim());
   }

   // If the route column is equal to our varibale, key_value
   // Return the ports that belong to this route

   if(split[2].trim().equals(key_value)){
      Route.set(split[0].trim());

   context.write(Route, one);
}
```

Description:

While hasMoreTokens() is true (method call returns 'true' if and only if there is at least one token in the string after the current position; false otherwise), it splits the tokens by "," and stores them. If the first column ("Port") is equalled to "Midnightblue-Epsilon", the equivalent route from the third column is stored in the variable "key_value".

Another if statement is added, if the third column (“Route”) is equalled to our variable, “key_value”, then return the harbours associated with it from the first column (again, we use trim to eliminate leading and trailing spaces). The output of running the code can be seen below:

```
Midnightblue-Epsilon    1
Orangered-Beta  1
Teal-Beta       1
```

The answer to which two other harbours are connected to “Midnightblue-Epsilon” by a route are: “Orangered-Beta” and “Teal-Beta”.

Part 3 Spark

For this section all of the code is stored in scripts which can be found in the folder “Part 3” with an accompanying README.txt.

The outputs of running the code are shown for Q1, Q2, Q3, Q4 & Q5 as it explicitly asks for it.

Q1.

This is a snippet of the script; full script is contained in the folder “Part 3” called “SparkQ1.sh”

```
val RestaurantDF = spark.read.format("com.databricks.spark.csv").option("header", "true").
option("delimiter", ",").option("inferSchema", "true").load("/home/spark.csv")

RestaurantDF.registerTempTable("RestaurantTable")

spark.sql("SELECT COUNT(*) FROM RestaurantTable").collect.foreach(println)
```

Description:

We begin by reading in our spark.csv as a Dataframe. We then register the Dataframe as a temp table in order to perform spark sql commands on it. We then run a spark sql command which counts the number of records that the dataset has in total. Another option is to use the following scala code on the dataframe: “**RestaurantDF.count()”** which returns the same output as above, confirming our answer is correct. The output from running the above and the answer to the question of the number of datasets in total can be seen below:

```
Long = 1000
```

Q2.

This is a snippet of the script; full script is contained in the folder “Part 3” called “SparkQ2.sh”

```
#!/bin/bash
# Script that finds the restaurant with the highest number of reviews

sed -i 's/No.Reviews/No_Reviews/' spark.csv

/spark/bin/spark-shell << EOF

val RestaurantDF = spark.read.format("com.databricks.spark.csv").option("header", "true").
option("delimiter", ",").option("inferSchema", "true").load("/home/spark.csv")
RestaurantDF.registerTempTable("RestaurantTable")
```

```
spark.sql("SELECT Restaurant, No_Reviews from RestaurantTable WHERE No_Reviews IN (SELECT
MAX(No_Reviews) FROM RestaurantTable)").collect.foreach(println)

EOF
```

Description:

The first sed statement replaces the column name “No.Reviews” with “No_Reviews”. This was necessary as the period (“.”) can cause issues in spark sql. As a result, this will be the first line on subsequent scripts in other to deal with this issue if a different script is ran first.

We convert the “spark.csv” to a Dataframe. Then convert it to a temp table in order to carry out Spark sql commands. Our Spark command selects restaurants and the number of reviews from our table. It then states where the number of reviews are equal to the max number of reviews. This will subsequently return the restaurant name and the number of reviews. Please see the output of the code below which answers which restaurant has the highest number of reviews:

```
[Roasted Shallot,1500]
```

Q3.

This is a snippet of the script; full script is contained in the folder “Part 3” called “SparkQ3.sh”

```
spark.sql("SELECT Restaurant FROM RestaurantTable WHERE LENGTH(Restaurant) = (SELECT MAX(L
ENGTH(Restaurant)) FROM RestaurantTable)").collect.foreach(println)
```

Description:

The beginning of this script which is not shown above includes the sed statement to change the column name and the converting of spark.csv into a Dataframe and to a temp table. All of which we have discussed in the previous question.

The select statement selects the restaurant where the length of the restaurant is equal to the max length of the restaurants. This will return the restaurant with the longest name as seen in the output below:

```
[Extraordinary Vegetable Soup Emporium Place]
```

Q4.

This is a snippet of the script; full script is contained in the folder “Part 3” called “SparkQ4.sh”

```
spark.sql("SELECT Region, SUM(No_Reviews) AS SumReviews FROM RestaurantTable GROUP BY Regi
on ORDER BY SumReviews DESC ").collect.foreach(println)
```

The beginning of this script which is not shown above includes the sed statement to change the column name and the converting of spark.csv into a Dataframe and to a temp table. All of which we have discussed in the previous question.

The select statement selects the region and the sum of the number of reviews which is renamed as ‘SumReviews’. The statement then groups by region and then orders by the newly renamed column SumReviews for readability. This will find the number of reviews for each region and can be seen in the output below:

```
[LA,25089] [OK,13650]
[DE,22667] [MD,13396]
[SC,21688] [PA,13378]
[AL,21447] [AZ,13161]
[NC,20676] [MA,13154]
[NM,20624] [TN,13071]
[OH,20306] [ND,13010]
[NJ,20231] [TX,12242]
[WI,19642] [MS,12192]
[NY,18689] [MN,12180]
[KS,18155] [ME,12179]
[IL,17780] [CO,11876]
[MI,17750] [HI,11630]
[WV,17492] [MO,11189]
[NV,16894] [IA,10752]
[IN,16666] [NH,10573]
[AR,16092] [GA,10506]
[AK,15888] [ID,9510]
[OR,15749] [WA,9305]
[CT,15448] [WY,9279]
[FL,15392] [KY,8886]
[MT,14854] [UT,8083]
[RI,14807] [CA,7011]
[VA,14528]
[SD,14403]
[VT,14392]
[NE,13957]
```

Q5.

This is a snippet of the script; full script is contained in the folder "Part 3" called "SparkQ5.sh" & "SparkQ5_1.sh"

```
#!/bin/bash
# Script determines the most frequently occuring term in the review column that
# doesn't include one of the following stop words: "A", "The", "of" - Part 1

sed -i 's/No.Reviews/No_Reviews/' spark.csv

cut -d, -f5 spark.csv > fifthcol.csv

/spark/bin/spark-shell << EOF

import org.apache.spark.ml.feature.StopWordsRemover

val sparkdf = spark.read.text("/home/fifthcol.csv").map( row => row.getString(0).split("""
 """))

val stopremover = new StopWordsRemover().setInputCol("value").setOutputCol("removed").setS
topWords(Array("the", "The", "a", "A", "of", "Of"))

val newsparkdf = stopremover.transform(sparkdf)

newsparkdf.select("removed").map(row => row.getSeq[String](0).mkString(" ")).write.text("/
home/clean_fifthcol")

EOF
```

At this point we rename the text file stored in "/home/clean_fifthcol/part_XXX.txt". This can be done using the following command: *mv part_XXX.txt cleaned_file.csv*

We then run *SparkQ5_1.sh*

```
val spark_rdd = sc.textFile("/home/clean_fifthcol/cleaned_file.csv")

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

Description:

The first sed statement replaces the column name “No.Reviews” with “No_Reviews”. This was necessary as the period (“.”) can cause issues. Our next cut statement extracts the fifth column “Reviewtext” and exports it to a csv called “fifthcol.csv”. We then read in the “fifthcol.csv” into a Dataframe. Our variable “stopremover” is defined listing the stop words we don’t want to see (“A”, “The”, “Of”). Transform is then used for a concise syntax for chaining custom transformations. We then write the output to the “clean_fifthcol” directory.

We must rename the text file contained in here to “cleaned_file.csv” and run SparkQ5_1.sh. This then reads in the file as an RDD and performs a map reduce to return the most frequently occurring term that isn’t “A”, “The”, “of”. The output of this can be seen below:

```
(String, Int) = (and,379)
```

Part 4 GraphX

For this section all of the code is stored in scripts which can be found in the folder “Part 4” with an accompanying README.txt. The outputs of running the code are shown for Q2, Q3, Q4 & Q5 as it explicitly asks for it.

Q1.

This is a snippet of the script; full script is contained in the folder “Part 4” called “GraphX_Q1.sh”

```
// Importing the packages
import org.apache.spark._
import org.apache.spark.rdd.RDD
import org.apache.spark.graphx._
import org.apache.spark.util.IntParam
import org.apache.spark.graphx.util.GraphGenerators

// Creating a mapper using hadoop_mirrored.csv
case class harb_map(HarbourName:String, HarbourNo:Long, Route: String, RouteNo:Long)
def parsingHarbMap(str: String): harb_map = {val line = str.split(","); harb_map(line(0),
line(1).toLong, line(2), line(3).toLong)}
var MapRDD = sc.textFile("./hadoop_mirrored.csv")
val header = MapRDD.first()
MapRDD = MapRDD.filter(row => row != header)
val harbourMapRDD = MapRDD.map(parsingHarbMap).cache()
val NewMapHarbour = harbourMapRDD.flatMap(har => Seq((har.HarbourName, har.HarbourNo))).di
stinct
val harbourMap = NewMapHarbour.map{ case (a, b) => (a -> b) }.collect.toMap

// Creating the vertices using GraphX_Edges.csv
case class Harbour(HarbourName:String, Route: String)
def parsingHarbour(str: String): Harbour = {val line = str.split(","); Harbour(line(1), li
ne(2))}
var harbourRDD = sc.textFile("./GraphX_Edges.csv")
val harbourHeader = harbourRDD.first()
harbourRDD = harbourRDD.filter(row => row != harbourHeader)
```

```
val harbourNewRDD = harbourRDD.map(parsingHarbour).cache()
val harbour = harbourNewRDD.flatMap(h => Seq((harbourMap(h.HarbourName), h.HarbourName))).
distinct

// Creating the edges using new_hadoop_edges.csv
case class Route(Route: String, From:String, To:String)
def parsingRoute(str: String): Route = {val line = str.split(","); Route(line(0), line(1),
 line(2))}
var routeRDD = sc.textFile("./GraphX_Edges.csv")
val routeHeader = routeRDD.first()
routeRDD = routeRDD.filter(row => row != routeHeader)
val newRouteRDD = routeRDD.map(parsingRoute).cache()
val route = newRouteRDD.map(ro =>((harbourMap(ro.From), harbourMap(ro.To), ro.Route))).dis
tinct
val edges = route.map { case (origin, destination, route) => Edge(origin, destination, rou
te) }

// Creating the graph
val nowhere = "nowhere"
val graph = Graph(harbour, edges, nowhere)
```

Description:

We start by import the packages. We then create a mapper using “hadoop_mirrored.csv”. This is done by creating a class, creating a function to parse into the harb_map class. We then load the data into an RDD variable. The header is removed, and we make the mapRDD. Create another RDD using flatmap and then create the mapper.

We then move on to create the vertices using “GraphX_Edges.csv”. The class is created, as well as a function to parse into the harbour class. The data is loaded into an RDD variable. The header is removed, and we make the harbour RDD. Another RDD is created using map and we then create the harbour variable.

Next, we create the edges using “GraphX_Edges.csv”. We create a class, create a function to parse into the route class. We then load the data into an RDD. The header is removed and the routeRDD is created. We create another RDD using map and call it route. Finally we create the edges and call it edges.

Finally, the graph is created.

Q2.

This is a snippet of the file; the file is contained in the folder “Part 4” called “GraphX_Q2.txt”

```
// Code that generates an array of each harbour’s connected routes

graph.triplets.sortBy(_.srcAttr, ascending=true).map(triplet => "Harbour: " + triplet.srcA
ttr + " - Associated Route: " + triplet.attr.toString).collect().distinct.foreach(println)
```

Description:

We use Triplets for this query. There is one triplet for each edge which contains information about both the vertices and the edge information. This class extends the edge class by adding the srcAttr and dstAttr members which contain the source and destination properties. We use this to generate an array of each harbour’s connected routes. The output can be seen below:

```
Harbour: Aliceblue-Delta - Associated Route: Enthusiasm_Forty-two
Harbour: Aliceblue-Iota - Associated Route: Monkshood_Two_hundred_and_twenty-nine
Harbour: Aliceblue-Kappa - Associated Route: Acacia_Ten
Harbour: Aliceblue-Kappa - Associated Route: Iris_One_hundred_and_twenty-seven
```

Q3.

This is a snippet of the file; the file is contained in the folder "Part 4" called "GraphX_Q3.txt"

```
val ReverseMapHarbour = harbourMapRDD.flatMap(har => Seq((har.HarbourNo, har.HarbourName))
).distinct
val reverseMap = ReverseMapHarbour.map{ case (a, b) => (a -> b) }.collect.toMap

graph.edges.filter { case ( Edge(origin, destination, route))=> route == "Porium_Thirty-
one"}.foreach(println)
```

Description:

We create a variable called ReverseMapHarbour using the flat map to reverse the order of Harbour Number and Harbour Name. The mapper is then created under the variable name reverseMap.

Moving onto our filter which queries the edges where the route is "Porium_Thirty-one". This gives us the Harbour Number "8516". We then plug this number into our "reverseMap" and the output can be seen below. This answers the question of which harbour(s) is/are served by route "Porium Thirty-one".

```
String = Yellowgreen-Eta
```

Q4.

This is a snippet of the file; the file is contained in the folder "Part 4" called "GraphX_Q4.txt"

```
val mostRoutes = graph.inDegrees.collect.sortWith(_._2 >_._2).map(x => (reverseMap(x._1),
x._2)).take(3)
```

Description:

The "inDegrees" here refers to the number of arcs incident. That is, the number of arcs directed towards the vertex. More simply, it is the number of edges coming towards a vertex in our directed graph. The output below shows the harbour with the most routes associated with it as well as the count of routes per harbour. As we can see "nowhere" appears first but given this is not a harbour, we will move to the next highest which is "Oldlace-Omicron".

```
mostRoutes: Array[(String, Int)] = Array((nowhere,373), (Oldlace-Omicron,8), (Powderblue-Lamda,7))
```

Q5.

This is a snippet of the file; the file is contained in the folder "Part 4" called "GraphX_Q5.txt"

```
val connectedHarbours = graph.degrees.collect.sortWith(_._2 >_._2).map(x => (reverseMap(x.
_1), x._2)).take(3)
```

Description:

We use graph.degrees here to get the to get the highest-degree vertex in a graph. This will allow us to find the most connected harbour from our graph. Please see below for the output of running the code and the answer to which harbour has the most routes associated with it. As we can see "nowhere" appears first but given this is not a harbour we will move to the next highest which is "Oldlace-Omicron".

```
connectedHarbours: Array[(String, Int)] = Array((nowhere,373), (Oldlace-Omicron,17), (Powderblue-Lamda,14))
```