Big Data Programming
Project 2 – Coding Assessment
Sai Srujan

Question 1: Bash and Data Management
1.1

1. To remove the erroneous string ”#]” from the name column of the dataset the following bash command was used:

sed 's/#]//g' bashdm.csv > output1.csv

The sed command is used to remove the erroneous string “#]” occurrences throughout the dataset. s/ search from beginning of the line of the dataset for “#]” string and then replace it with an empty value which basically refers to removing of the string. The /g is specified to ensure that all the occurrences of the string is searched and replaced in the dataset throughout. The final output after removing the string is saved into the output1.csv file.

A sample output of the removal of the erroneous string from the data set:

2. To convert the file delineated by a ”-” character to a comma delineated file the following command was used:

sed -e ':a;s/^\(\("[^"]*"\|[^",]*\)*\)-/\1,/;ta' output1.csv > output2.csv

The sed command was used to replace all the occurrences of the “-“ character to a comma. The -e option is used to specify the expression in the script and since the “-“ character should not be replaced in the names we use :a to specify the address location for branch (loop), s/ for searching from beginning of line for '[^",]*,' or '"...",' then replace “-“ by comma, ta branch to a if previous s/ have been matched. So all the “-“ characters will be replaced with comma except for the ones inside the double quotes in the data for names. The output is then saved in the output2.csv file.

A sample output of the comma separated data set:

Command Prompt - docker exec -it dbBigData bash

```
INDEX,Name,Age,Country,Height,Hair_Colour,YLA,CONF
0,Jeanne Wallace,86,CODE:MY,157,White,1,YES
1,Anthony Gentry,49,CODE:QA,162,Black,1,YES
2,Marcia Jones,75,CODE:IE,186,Dyed,1,YES
3,"James Patch-Walker-Willis",18,CODE:KI,166,Ginger,1,YES
4,Vickie White,42,CODE:CO,160,Blonde,1,YES
5,Dwayne Peterson,70,CODE:UZ,142,Dyed,1,YES
6,Rose Rubottom,79,CODE:PL,141,White,1,YES
7,Thomas Salas,30,CODE:ML,156,Black,1,YES
8,Xiao Uong,41,CODE:AD,159,Blonde,1,YES
9,Lakisha Stewart,48,CODE:CF,182,Blonde,1,YES
10,Claire Nunes,43,CODE:SN,171,Black,1,YES
```

After the dataset is comma separated, the sed command is used to remove the double quotes to make it consistent and save in output3.csv:

sed 's/"//g' output2.csv > output3.csv

Sample output:

```
3,James Patch-Walker-Willis,18,CODE:KI,166,Ginger,1,YES
```

3. To remove the columns in dataset with non-useful values we check for the columns having the same values for all the rows. The following script was used:

```
#!/bin/bash
for i in {1..8}
do
  val=$(cut -d, -f$i output3.csv| sort | uniq | wc -l)
  if [ $val -le 2 ]
      then
           col="${col}${i},"
 fi
done
colNo=`echo $col | sed 's/\(.*\),/\1 /'`
cut --complement -d, -f$colNo output3.csv> output4.csv
```

The cut command is used to separate each of the column values in the for loop. In the loop for each column, the unique values are extracted with the uniq and the count of number of unique values is calculated with wc-l. All the columns with 2 unique values(column name, value in columns) are noted in a comma separated string. The trailing comma is then removed and all the column numbers (comma separated) is provided to the cut command to remove those columns and save the output in output4.csv. **The columns YLA, CONF are removed.**

Sample output:

Command Prompt - docker exec -it dbBigData bash

```
INDEX,Name,Age,Country,Height,Hair_Colour
0,Jeanne Wallace,86,CODE:MY,157,White
1,Anthony Gentry,49,CODE:QA,162,Black
2,Marcia Jones,75,CODE:IE,186,Dyed
3,James Patch-Walker-Willis,18,CODE:KI,166,Ginger
4,Vickie White,42,CODE:CO,160,Blonde
5,Dwayne Peterson,70,CODE:UZ,142,Dyed
```

4. To update the country-code with the country-name in the dataset the **update-country.sh** bash script is used which is attached in the folder.

Each line of the dataset is read and the country column data is extracted from each line. With the country code from that line, grep is used to find the country details in the dictionary provided. From the details returned from the grep command, the cut command is used to extract the country name from the dictionary and then sed is used with -i option to replace the country code with the name, so that the replacement happens in place. The updated list will be saved in ouput4.csv itself.

Sample output:

Command Prompt - docker exec -it dbBigData bash

```
INDEX,Name,Age,Country,Height,Hair_Colour
0,Jeanne Wallace,86,Malaysia,157,White
1,Anthony Gentry,49,Qatar,162,Black
2,Marcia Jones,75,Ireland,186,Dyed
3,James Patch-Walker-Willis,18,Kiribati,166,Ginger
4,Vickie White,42,Colombia,160,Blonde
5,Dwayne Peterson,70,Uzbekistan,142,Dyed
6,Rose Rubottom,79,Poland,141,White
7,Thomas Salas,30,Mali,156,Black
8,Xiao Uong,41,Andorra,159,Blonde
9,Lakisha Stewart,48,Central_African_Rep,182,Blonde
10,Claire Nunes,43,Senegal,171,Black
```

1.2

1. The two bash scripts to insert the data into SQL and mongoDB are attached in the folder. The script **sql-insert.sh** inserts the data into the SQL tables and **mongo-insert.sh** inserts into mongoDB. The table for SQL is created in the script first with the name person_data and then cut is used to extract the various values from the file and insert those values into the table.
For mongoDB each line of the dataset is looped over, the values are extracted from each column and is then inserted to the personData collection in mongoDB.
Sample output:

Command Prompt - docker exec -it dbBigData bash

```
Database changed
mysql> select * from person_data;
+-----+--------------------------------+------+----------------------------+--------+------------+
| id  | name                           | age  | country                    | height | hairColour |
+-----+--------------------------------+------+----------------------------+--------+------------+
|   0 | Jeanne Wallace                 |   86 | Malaysia                   |    157 | White      |
|   1 | Anthony Gentry                 |   49 | Qatar                      |    162 | Black      |
|   2 | Marcia Jones                   |   75 | Ireland                    |    186 | Dyed       |
|   3 | James Patch-Walker-Willis      |   18 | Kiribati                   |    166 | Ginger     |
|   4 | Vickie White                   |   42 | Colombia                   |    160 | Blonde     |
|   5 | Dwayne Peterson                |   70 | Uzbekistan                 |    142 | Dyed       |
|   6 | Rose Rubottom                  |   79 | Poland                     |    141 | White      |
|   7 | Thomas Salas                   |   30 | Mali                       |    156 | Black      |
|   8 | Xiao Uong                      |   41 | Andorra                    |    159 | Blonde     |
|   9 | Lakisha Stewart                |   48 | Central_African_Rep        |    182 | Blonde     |
|  10 | Claire Nunes                   |   43 | Senegal                    |    171 | Black      |
```

Command Prompt - docker exec -it dbBigData bash

```
2021-11-27T20:44:25.518+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2021-11-27T20:44:25.518+0000 I CONTROL  [initandlisten]
> use projectDB
switched to db projectDB
> db.personData.find()
{ "_id" : ObjectId("61a2bf965e705f0652cfcf24"), "id" : "0", "name" : "Jeanne Wallace", "age" : "86", "country" : "Malaysia", "height" : "157", "hairColour" : "White" }
{ "_id" : ObjectId("61a2bf9622f41a83b2f98671"), "id" : "1", "name" : "Anthony Gentry", "age" : "49", "country" : "Qatar", "height" : "162", "hairColour" : "Black" }
{ "_id" : ObjectId("61a2bf967658eebbd0a7804e"), "id" : "2", "name" : "Marcia Jones", "age" : "75", "country" : "Ireland", "height" : "186", "hairColour" : "Dyed" }
{ "_id" : ObjectId("61a2bf96b1e2df68bc137ebb"), "id" : "3", "name" : "James Patch-Walker-Willis", "age" : "18", "country" : "Kiribati", "height" : "166", "hairColour" :
 "Ginger" }
{ "_id" : ObjectId("61a2bf97ed50e33950bfb3ce"), "id" : "4", "name" : "Vickie White", "age" : "42", "country" : "Colombia", "height" : "160", "hairColour" : "Blonde" }
{ "_id" : ObjectId("61a2bf9750bea3a7c80f9b09"), "id" : "5", "name" : "Dwayne Peterson", "age" : "70", "country" : "Uzbekistan", "height" : "142", "hairColour" : "Dyed"
}
{ "_id" : ObjectId("61a2bf9772675842024b67ef"), "id" : "6", "name" : "Rose Rubottom", "age" : "79", "country" : "Poland", "height" : "141", "hairColour" : "White" }
{ "_id" : ObjectId("61a2bf97ff621e9e6428fd43"), "id" : "7", "name" : "Thomas Salas", "age" : "30", "country" : "Mali", "height" : "156", "hairColour" : "Black" }
{ "_id" : ObjectId("61a2bf9716f4cede279b3de6"), "id" : "8", "name" : "Xiao Uong", "age" : "41", "country" : "Andorra", "height" : "159", "hairColour" : "Blonde" }
{ "_id" : ObjectId("61a2bf971dc0a7c5f1b6d476"), "id" : "9", "name" : "Lakisha Stewart", "age" : "48", "country" : "Central_African_Rep", "height" : "182", "hairColour"
: "Blonde" }
{ "_id" : ObjectId("61a2bf97a67a5f4b3b31d86e"), "id" : "10", "name" : "Claire Nunes", "age" : "43", "country" : "Senegal", "height" : "171", "hairColour" : "Black" }
{ "_id" : ObjectId("61a2bf974dccf273b45b9327"), "id" : "11", "name" : "Cheryl Person", "age" : "73", "country" : "Cuba", "height" : "155", "hairColour" : "Blonde" }
{ "_id" : ObjectId("61a2bf97671cf7e458e596b9"), "id" : "12", "name" : "Tina Mesiti", "age" : "27", "country" : "China", "height" : "147", "hairColour" : "Black" }
{ "_id" : ObjectId("61a2bf97d28b3b2bb47a7dcf"), "id" : "13", "name" : "James Santos", "age" : "33", "country" : "Estonia", "height" : "143", "hairColour" : "White" }
{ "_id" : ObjectId("61a2bf983fe485100610cecf"), "id" : "14", "name" : "Chester Bradshaw", "age" : "28", "country" : "Eritrea", "height" : "182", "hairColour" : "Blonde"
}
{ "_id" : ObjectId("61a2bf98af1531c2b206e18c"), "id" : "15", "name" : "Dale Mitchell", "age" : "78", "country" : "Viet_Nam", "height" : "162", "hairColour" : "Ginger" }
{ "_id" : ObjectId("61a2bf98f32f888ac65237c5"), "id" : "16", "name" : "Rodney Chesney", "age" : "56", "country" : "Bhutan", "height" : "153", "hairColour" : "Ginger" }
{ "_id" : ObjectId("61a2bf98d66b103a7d187121"), "id" : "17", "name" : "Margaret Anderson", "age" : "48", "country" : "Viet_Nam", "height" : "194", "hairColour" : "Blond
e" }
{ "_id" : ObjectId("61a2bf986e8e21ccb5f00780"), "id" : "18", "name" : "Richard Chung", "age" : "21", "country" : "Comoros", "height" : "188", "hairColour" : "Black" }
{ "_id" : ObjectId("61a2bf98a1bdbf593c7ad28c"), "id" : "19", "name" : "Kathleen Mcreynolds", "age" : "26", "country" : "Samoa", "height" : "193", "hairColour" : "Blonde
" }
Type "it" for more
```

2. The following query is used to find the average height per country:
select country, avg(height) from person_data group by country;

Sample output:

Command Prompt - docker exec -it dbBigData bash

```
mysql> select country, avg(height) from person_d
+-----------------------------+-------------+
| country                     | avg(height) |
+-----------------------------+-------------+
| Malaysia                    |    161.2500 |
| Qatar                       |    165.1667 |
| Ireland                     |    172.8571 |
| Kiribati                    |    167.2500 |
| Colombia                    |    165.6667 |
| Uzbekistan                  |    168.3333 |
| Poland                      |    171.3333 |
| Mali                        |    168.5000 |
```

3. The following query is used to find the maximum height per hair colour:
select hairColour, max(height) from person_data group by hairColour;

```
mysql> select hairColour, max(height) from person_data group by hairColour;
+------------+-------------+
| hairColour | max(height) |
+------------+-------------+
| White      |         200 |
| Black      |         200 |
| Dyed       |         199 |
| Ginger     |         199 |
| Blonde     |         200 |
| Brown      |         200 |
+------------+-------------+
6 rows in set (0.01 sec)
```

4. The following script is used to add a characteristic “UID”-unique ID to each person:

var i=0; db.personData.find().forEach(function(doc){db.personData.update(doc, {$set : {"UID": i++}},false,true);});

```
> db.personData.find().pretty()
{
        "_id" : ObjectId("61a2bf965e705f0652cfcf24"),
        "id" : "0",
        "name" : "Jeanne Wallace",
        "age" : "86",
        "country" : "Malaysia",
        "height" : "157",
        "hairColour" : "White",
        "UID" : 0
}
{
        "_id" : ObjectId("61a2bf9622f41a83b2f98671"),
        "id" : "1",
        "name" : "Anthony Gentry",
        "age" : "49",
        "country" : "Qatar",
        "height" : "162",
        "hairColour" : "Black",
        "UID" : 1
}
```

5. The person with the lowest value for height is **Ryan Hall.**
To find the person with lowest value for height:
db.personData.find().sort({height:1}).limit(1)

```
> db.personData.find().sort({height:1}).limit(1)
{ "_id" : ObjectId("61a2bfdd808a6f798138e2d3"), "id" : "803", "name" : "Ryan Hall", "age" : "63",
"country" : "Ethiopia", "height" : "139", "hairColour" : "Blonde" }
```

Question 2: Simple Hadoop Graph Processing

1. To create a list of number of routes that connect each harbour the **HarbourRoutes.java** script is used which is attached in the folder. The map phase creates key value pairs of (harbour_name, 1) for each of the harbours in the list. The reduce phase adds up the count of routes for each harbour and gives the final count for each of them.
Sample Output:

```
Aliceblue-Delta 1
Aliceblue-Iota  1
Aliceblue-Kappa 2
Antiquewhite-Alpha      1
Antiquewhite-Epsilon    1
Antiquewhite-Mu 1
Antiquewhite-Nu 1
Antiquewhite-Omicron    3
Antiquewhite-Pi 1
Antiquewhite-Rho        1
Antiquewhite-Sigma      1
Antiquewhite-Xi 2
Aqua-Alpha      1
```

2. To get the harbour associated with the route ”Wolfsbane_Nine” the **HarbourRoutes2.java** script is used which is attached in the folder. To get the harbour name the map phase was changed to produce key value pairs of (route_name, harbour_name). The reduce phase then combines all the harbours associated with the routes and produces route_name with the harbour_names separated by a space. In the reduce phase the results were filtered to only return the value with the “Wolfsbane_Nine” route.
The harbour associated with the route ”Wolfsbane_Nine” is **Mintcream-Tau**.

```
root@59c5367927a3:~# hdfs dfs -cat /output/part-r-00000
2021-12-03 15:12:32,828 INFO sasl.SaslDataTransferClient: SASL encrypt
Wolfsbane_Nine    Mintcream-Tau
```

3. To get the harbours connected by route “Carnation_Sixty-seven**“** the **HarbourRoutes3.java** script is used which is attached in the folder. The script is very similar to the one used for the previous one but the only change is that in the reduce phase the filtering condition is changed to return the value with the “Carnation_Sixty-seven” route.
The harbours associated with the route ” Carnation_Sixty-seven” are **Lightcoral-Pi and Seashell-Nu**

```
root@59c5367927a3:~# hdfs dfs -cat /output/part-r-00000
2021-12-03 15:25:28,890 INFO sasl.SaslDataTransferClient: SASL encryp
Carnation_Sixty-seven     Lightcoral-Pi Seashell-Nu
```

4. To get the harbours that fielded emergency routes the **HarbourRoutes4.java** script is used which is attached in the folder. The map phase produces key value pairs of (route_number, harbour_name). The reduce phase then combines all the harbours associated with that route number with the harbour_names separated by a space. In the reduce phase we only return the values with route numbers starting with ‘911’.

The routes are 911, 9110538 and 9112064.

```
root@59c5367927a3:~# hdfs dfs -cat /output/part-r-00000
2021-12-03 16:46:37,613 INFO sasl.SaslDataTransferClient: SASL encryption trust check: lo
911       Midnightblue-Rho
9110538   Mediumvioletred-Eta Darkkhaki-Zeta Goldenrod-Sigma
9112064   Bisque-Mu Ghostwhite-Omicron Burlywood-Epsilon Seashell-Zeta Sandybrown-Iota
```

5. To find the harbours the **HarbourRoutes5.java** script is used which is attached in the folder. Two map reduce operations are used, the first map reduce gives the output of harbour and route associated with “Midnightblue-Epsilon” and the second map reduce gives key value pairs of routes and the associated harbour of those routes. We then use the cut command to get the route from the first output file as it has only one output with the harbour and route. We then use grep to find the route from the second output which gives all the harbours associated with it.

”Midnightblue-Epsilon” is connected to two other harbours by a route **Chrysanthemum_Four_hundred_and_sixty-five** and the harbours connected to this are **Orangered-Beta** and **Teal-Beta.**

```
root@59c5367927a3:/# hdfs dfs -copyToLocal /output/part-r-00000 output1.txt
2021-12-10 15:35:46,796 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHos
ostTrusted = false
root@59c5367927a3:/# hdfs dfs -copyToLocal /output1/part-r-00000 output2.txt
2021-12-10 15:35:55,037 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHos
ostTrusted = false
root@59c5367927a3:/# grep `cut -f2 output1.txt` output2.txt
Chrysanthemum_Four_hundred_and_sixty-five        Midnightblue-Epsilon Teal-Beta Orangered-Beta
```

Question 3: Spark

The given dataset is loaded into the dataframe in Spark to run some operations based on it and then get the required results. (The full script with all commands is attached in folder with the name spark.txt)

val projectData = spark.read.format("com.databricks.spark.csv").option("header","true").option("inferSchema","true").load("./spark.csv")

```
scala> val projectData = spark.read.format("com.databricks.spark.csv").option("header","true").option("inferSchema","true").load("/spark.csv")
projectData: org.apache.spark.sql.DataFrame = [INDEX: int, Restaurant: string ... 3 more fields]

scala> projectData.printSchema
root
 |-- INDEX: integer (nullable = true)
 |-- Restaurant: string (nullable = true)
 |-- Region: string (nullable = true)
 |-- No.Reviews: integer (nullable = true)
 |-- Reviewtext: string (nullable = true)
```

To execute the SQL commands, we create a table using the command **projectData.registerTempTable("projectData")** and then run the SQL commands.

1. The number of records the dataset has in total is **1000.**

projectData.count()

```
scala> projectData.count()
res2: Long = 1000
```

2. The restaurant with the highest number of reviews is **Roasted Shallot** with 1500 reviews.
projectData.where(projectData("`No.Reviews`") === projectData.agg(max("`No.Reviews`")).first()(0)).show()

```
+-----+---------------+------+----------+--------------------+
|INDEX|     Restaurant|Region|No.Reviews|          Reviewtext|
+-----+---------------+------+----------+--------------------+
|  420|Roasted Shallot|    AK|      1500|As for the servic...|
+-----+---------------+------+----------+--------------------+
```

3. The restaurant with the longest name is **Extraordinary Vegetable Soup Emporium Place.**

val longestName = spark.sql("select * from projectData where length(Restaurant) = (select max(length(Restaurant)) from projectData)");

```
scala> val longestName = spark.sql("select * from projectData where length(Restaurant) = (select max(length(Restaurant)) from projectData)");
longestName: org.apache.spark.sql.DataFrame = [INDEX: int, Restaurant: string ... 3 more fields]

scala> longestName.collect.foreach(println)
[885,Extraordinary Vegetable Soup Emporium Place,SC,1433,Good prices]
```

4. To find the number of reviews for each region the reviews from each region are grouped and then the sum of all the values is calculated.

projectData.groupBy("Region").agg(sum(projectData("`No.Reviews`"))).show()

```
scala> projectData.groupBy("Region").agg(sum(projectData("`No.Reviews`"))).show()
+------+--------------+
|Region|sum(No.Reviews)|
+------+--------------+
|    SC|         21688|
|    AZ|         13161|
|    LA|         25089|
|    MN|         12180|
|    NJ|         20231|
|    OR|         15749|
|    VA|         14528|
|    RI|         14807|
|    KY|          8886|
|    WY|          9279|
|    NH|         10573|
|    MI|         17750|
|    NV|         16894|
|    WI|         19642|
|    ID|          9510|
|    CA|          7011|
|    CT|         15448|
|    NE|         13957|
|    MT|         14854|
|    NC|         20676|
+------+--------------+
only showing top 20 rows
```

5. The most frequently occurring term in the review column is ‘good’ if we consider only the reviews not containing the stop words The’, ‘A’,’of’. We use RDD as it is easier to process the large amount of data: **val projectDataRDD = sc.textFile("./spark.csv")**

projectDataRDD.map(line => line.split(",")(4)).filter(line => !(line contains("The")) && !(line contains("of")) && !(line contains("a"))).flatMap(line => line.split(" ")).map(word=>(word,1)).reduceByKey(_+_).sortBy(T=>T._2, false).take(1)

```
scala> projectDataRDD.map(line => line.split(",")(4)).filter(line => !(line contains("The")) && !(
line contains("of")) && !(line contains("a"))).flatMap(line => line.split(" ")).map(word=>(word,1)
).reduceByKey(_+_).sortBy(T=>T._2, false).take(1)
res13: Array[(String, Int)] = Array((good,9))
```

If we consider all the reviews and then take the count of the words other than the stop words 'The', 'A','of' we get 'the' as the most frequent term. But if we ignore all stop terms we get 'food' as the most frequent term used.

projectDataRDD.map(line => line.split(",")(4)).flatMap(line => line.split(" ")).filter(line => !(line contains("The")) && !(line contains("of")) && !(line contains("a"))).map(word=>(word,1)).reduceByKey(_+_).sortBy(T=>T._2, false).collect()

```
scala> projectDataRDD.map(line => line.split(",")(4)).flatMap(line => line.split(" ")).filter(line
 => !(line contains("The")) && !(line contains("of")) && !(line contains("a"))).map(word=>(word,1)
).reduceByKey(_+_).sortBy(T=>T._2, false).collect()
res14: Array[(String, Int)] = Array((the,405), (I,295), (to,216), (is,171), (food,112), (for,105),
 (not,104), (it,103), (in,101), (this,96), (good,89), (with,72), (service,68), (be,66), (were,65),
```

Question 4: GraphX

1. To create the graph the following steps are performed: (The full script with all commands is attached in folder with the name graphx.txt)

- **case class Harbour(index:Int, route:String, from: String, to:String, trip_no:Long)**
- **def parseHarbour(str:String):Harbour={val line=str.split(",");Harbour(line(0).toInt, line(1), line(2), line(3), line(4).toLong)}**
- **var textRDD = sc.textFile("./hadoop_mirrored.csv")**
- **val header=textRDD.first()**
- **textRDD = textRDD.filter(row=>row!=header)**
- **val harbourRDD = textRDD.map(parseHarbour).cache()**
- **var portNames = harbourRDD.flatMap(port=>Seq(port.from,port.to)).distinct –** All the distinct harbour ports in the dataset is returned
- **val portMap = portNames.zipWithIndex.map{ case (port,i) => (port -> i)}.collect.toMap –** A map is created by providing an id to each of the harbour port
- **var ports = harbourRDD.flatMap(port=>Seq((portMap(port.from), port.from),(portMap(port.to),port.to))).distinct –** Vertices are created with the id of the port and the name of the port
- **val routes = harbourRDD.map(port=>((portMap(port.from),portMap(port.to)),port.route)).distinct –** The routes are created with the ids of the source and destination port with the route name as the property of the edge
- **val edges=routes.map{case((from,to),route)=>Edge(from,to,route)} –** The edges are created from the routes variable with source and destination port id and route name
- **val HarbourMap = portNames.zipWithIndex.map{ case (port,i) => (i -> port)}.collect.toMap –** A reverse map is created to get the name of the harbour ports with the ids of the ports.
- **val nowhere="nowhere"**
- **val graph=Graph(ports,edges,nowhere)**

```
scala> val graph=Graph(ports,edges,nowhere)
graph: org.apache.spark.graphx.Graph[String,String] = org.apache.spark.graphx.impl.GraphImpl@53935785

scala> graph.vertices.take(2)
res9: Array[(org.apache.spark.graphx.VertexId, String)] = Array((68,Lightblue-Xi), (386,Orange-Zeta))

scala> graph.edges.take(2)
res10: Array[org.apache.spark.graphx.Edge[String]] = Array(Edge(1,109,Solidago_Three_hundred_and_thirty), Edge(4,265,Ranunculus_One_hundred_and_seventy-one))
```

2. To get the array of all the routes connected to each harbour the following code is used:

val routeMap = graph.collectEdges(EdgeDirection Out)

val harbourRoutes = routeMap.map{case (x,y) => (HarbourMap(x),y.map(e => e.attr).distinct.size,y.map(e => e.attr).distinct.mkString(","))}

harbourRoutes.foreach(println)

The collectEdges method is used to get all the edges connected to the harbours. Then each harbour is mapped with the distinct routes associated with it. All the routes associated with each harbour will then be returned.

Sample output is given below:

```
scala> harbourRoutes.foreach(println)
(Lightblue-Xi,1,Throatwort_Eight)
(Yellow-Rho,3,Nerine_Two,Achillea_Three_hundred_and_sixty-four,Myrtle_Three_hundred_and_nine)
(Cornflowerblue-Theta,1,Nerine_Two)
(Orange-Zeta,1,Throatwort_Three_hundred_and_five)
(Mediumseagreen-Delta,1,Broom_Four_hundred_and_fourteen)
(Mediumpurple-Gamma,3,Carnation_Four_hundred_and_forty,Liatris_One_hundred_and_ninety,Love-lies-bleeding_Ninety-five)
(Mediumorchid-Epsilon,2,Myrsine_Two_hundred,Porium_Two_hundred_and_sixty-six)
(Wheat-Sigma,1,Buddleia_Three_hundred_and_sixteen)
(Saddlebrown-Lamda,1,Peony_One_hundred_and_ninety-nine)
(Darkviolet-Iota,2,Nigella_One_hundred_and_seventy-one,Cornflower_Four_hundred_and_two)
(Darkorange-Pi,3,Narcissus_Forty-five,Convallaria_One_hundred_and_sixty-one,Rose_Two_hundred_and_thirteen)
(Maroon-Tau,3,Gyp_Two_hundred_and_ninety-four,Ranunculus_One_hundred_and_eighteen,Godetia_Two_hundred_and_eighty-six)
(Peru-Mu,1,Lily_One_hundred_and_fourteen)
(Mediumorchid-Gamma,1,Bellflower_One_hundred_and_sixty-two)
(Mintcream-Beta,1,Throatwort_Three_hundred_and_thirty-four)
(Darkslategrey-Mu,2,Pittosporum_Two_hundred_and_eighteen,Nephrolepis_Three_hundred_and_seventy-three)
(Yellowgreen-Omicron,1,Lisianthus_One_hundred_and_ten)
(Yellow-Eta,1,Gerbera_Twelve)
(Firebrick-Pi,1,Gardenia_Two_hundred_and_ninety-eight)
(Mediumvioletred-Upsilon,5,Waxflower_Two_hundred_and_twenty-five,Ginger_Two_hundred_and_sixteen,Knapweed_Two_hundred_and
_ninety-seven,Anthurium_Three_hundred_and_eighteen,Lisianthus_One_hundred_and_ten)
```

3. The harbour(s) is/are served by route ”Porium Thirty-one” is **Yellowgreen-Eta**

graph.edges.filter{case (Edge(from,to,route))=>route.equals("Porium_Thirty-one")}.take(3)

We filter the edge with the required value and get the harbour name by mapping it.

```
scala> graph.edges.filter{case (Edge(from,to,route))=>route.equals("Porium_Thirty-one")}.take(3)
res20: Array[org.apache.spark.graphx.Edge[String]] = Array(Edge(82,275,Porium_Thirty-one))

scala> HarbourMap(82)
res21: String = Yellowgreen-Eta

scala> HarbourMap(275)
res22: String = nowhere
```

4. The harbour which has the most routes associated with it is **Darksalmon-Zeta.**

val routeMap = graph.collectEdges(EdgeDirection Out)

val harbourRoutes = routeMap.map{case (x,y) => (HarbourMap(x),y.map(e => e.attr).distinct.size,y.map(e => e.attr).distinct.mkString(","))}

harbourRoutes.sortBy(x => x._2,ascending=false).take(2)

The collectEdges method is used to get all the edges connected to the harbours. Each harbour, the number of distinct routes and the distinct routes associated with it (comma separated) are mapped together. It is then sorted by the number of routes associated with each harbour.

(Considering the distinct route names from each harbour and not taking the number of routes as they can have same route names from each harbour)

```
scala> harbourRoutes.sortBy(x => x._2,ascending=false).take(2)
res58: Array[(String, Int, String)] = Array((Darksalmon-Zeta,6,Hippeastrum_Seventy-two,Rattlesnake_Four_hundred_and_twen
ty-nine,Speedwell_Twelve,Cosmos_Three_hundred_and_sixty-one,Gardenia_Three_hundred_and_sixty-three,Snapdragon_Two_hundre
d_and_fifty-nine), (Mediumvioletred-Upsilon,5,Waxflower_Two_hundred_and_twenty-five,Ginger_Two_hundred_and_sixteen,Knapw
eed_Two_hundred_and_ninety-seven,Anthurium_Three_hundred_and_eighteen,Lisianthus_One_hundred_and_ten))
```

5. If we consider the harbour directly connected to other harbours then we get **Oldlace-Omicron.** collectNeigborIDS method is used to get all neighbours and is mapped with the associated number of neighbours. It is then sorted on size to get the most connected.

val neighbors = graph.collectNeighborIds(EdgeDirection Out)
val harbourConnections = neighbors.map{case (x,y) => (HarbourMap(x),y.distinct.size,y.map(e => HarbourMap(e)).distinct.mkString(","))}
harbourConnections.sortBy(x => x._2,ascending=false).take(2)

```
scala> harbourConnections.sortBy(x => x._2,ascending=false).take(2)
res11: Array[(String, Int, String)] = Array((Oldlace-Omicron,9,Crimson-Zeta,Lightsteelblue-Theta,Red-Pi,Red-Lamda,Ghostw
hite-Omega,nowhere,Cyan-Psi,Powderblue-Lamda,Darksalmon-Beta), (Ghostwhite-Omicron,7,Seashell-Zeta,Sandybrown-Iota,Deeps
kyblue-Sigma,Bisque-Mu,Burlywood-Epsilon,nowhere,Green-Nu))
```

If we consider the number of Harbours connected indirectly to all the harbours then we get **Orangered-Epsilon**

val connectedComponents = graph.connectedComponents.vertices.map(_.swap).groupByKey (Using Connected Components algorithm from GraphX)

```
(0,CompactBuffer(68, 386, 574, 454, 534, 324, 180, 340, 320, 130, 582, 580, 408, 586, 428, 66, 138, 170, 464, 346, 14, 4
80, 518, 530, 24, 466, 302, 146, 140, 624, 204, 390, 514, 226, 318, 12, 628, 228, 94, 452, 192, 450, 160, 330, 482, 492,
 100, 332, 420, 366, 184, 632, 300, 460, 22, 544, 548, 350, 194, 126, 528, 40, 186, 558, 434, 118, 590, 58, 168, 636, 38
8, 296, 338, 524, 254, 556, 82, 216, 476, 28, 292, 98, 222, 30, 246, 92, 266, 344, 336, 214, 472, 322, 252, 414, 106, 72
, 110, 546, 248, 218, 144, 274, 178, 84, 592, 122, 404, 56, 76, 470, 326, 566, 182, 394, 208, 124, 164, 438, 382, 206, 8
, 174, 88, 372, 618, 244, 536, 602, 34, 440, 4, 526, 474, 412, 134, 600, 502, 622, 80, 458, 352, 510, 104, 368, 152, 272
, 270, 202, 278, 236, 552, 90, 74, 20, 18, 400, 342, 612, 268, 462, 256, 406, 334, 162, 224, 568, 504, 614, 200, 616, 37
0, 166, 578, 418, 142, 584, 606, 538, 220, 572, 172, 62, 250, 594, 264, 284, 190, 304, 6, 444, 442, 620, 496, 86, 60, 11
4, 306, 308, 328, 282, 516, 2, 398, 26, 230, 52, 554, 176, 570, 298, 16, 158, 156, 638, 456, 376, 426, 262, 294, 188, 64
, 50, 416, 358, 362, 132, 634, 242, 70, 432, 364, 596, 422, 500, 10, 148, 540, 564, 280, 276, 232, 374, 116, 380, 48, 24
0, 128, 260, 490, 0, 446, 560, 42, 102, 210, 314, 468, 448, 604, 598, 392, 488, 478, 484, 550, 234, 384, 522, 348, 315,
89, 19, 609, 155, 451, 297, 129, 71, 373, 369, 529, 635, 307, 171, 311, 473, 11, 235, 303, 37, 185, 215, 195, 1, 63, 543
, 201, 17, 443, 535, 605, 309, 211, 447, 299, 217, 539, 21, 321, 509, 331, 577, 77, 403, 367, 355, 283, 429, 59, 247, 34
7, 255, 637, 293, 93, 489, 139, 621, 113, 565, 515, 453, 525, 599, 547, 31, 363, 381, 137, 81, 419, 387, 231, 345, 375,
29, 79, 633, 551, 603, 191, 591, 161, 275, 541, 57, 149, 505, 581, 463, 313, 83, 251, 9, 49, 435, 109, 423, 167, 519, 18
7, 467, 397, 15, 553, 337, 441, 401, 243, 379, 585, 329, 351, 589, 95, 371, 469, 507, 501, 571, 181, 227, 479, 237, 317,
 259, 141, 143, 595, 579, 7, 223, 461, 389, 575, 499, 457, 615, 439, 221, 151, 101, 267, 491, 327, 593, 193, 153, 319, 5
13, 265, 437, 105, 359, 65, 157, 35, 383, 639, 111, 611, 527, 415, 175, 449, 459, 279, 495, 89, 341, 597, 241, 497, 339,
 103, 481, 199, 269, 145, 99, 583, 409, 277, 253, 169, 395, 261, 25, 333, 545, 343, 613, 617, 601, 627, 393, 213, 503, 3
3, 561, 23, 3, 135, 425, 405, 85, 91, 431, 391, 287, 197, 239, 5, 225, 189, 455, 511, 271, 13, 475, 325, 465, 323, 407,
477, 55, 557, 87, 177, 107, 121, 349, 607, 549, 531, 75, 51, 179, 219, 563, 567, 445, 45, 385, 305, 285, 421, 569, 173,
273, 183, 523, 629, 205, 133, 43, 555, 159, 61, 47, 301, 485, 587, 97, 281, 207, 117, 27, 521, 233, 245, 537, 559, 295,
335, 361, 163, 377, 493, 357, 573, 487, 517, 365, 433, 147))
```

HarbourMap(0) -> res27: String = Orangered-Epsilon