Introduction and Tutorial Guide on SUSS-Microsoft Analytics Hackathon 2018

Kevin Kam Fung Yuen

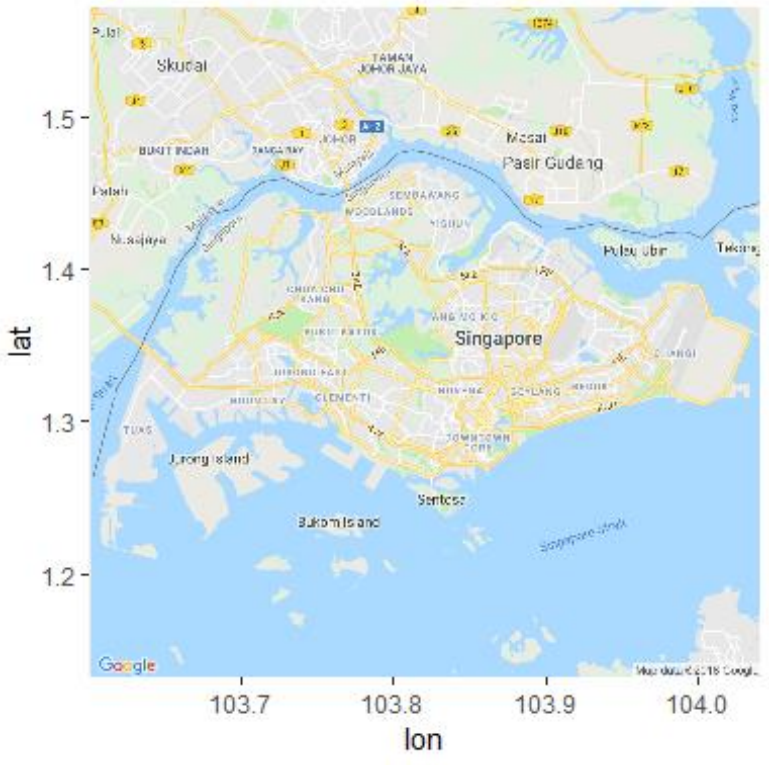
PhD, Senior Lecturer, School of Business, Singapore University of Social Sciences

kfyuen@suss.edu.sg, kevinkf.yuen@gmail.com

1 September, 2018

1.Background

EAT DELIVERY, a company specialising in food order delivery services, would like to improve their delivery order assignment process. Therefore, they are planning to develop an automated system to optimize the delivery order assignment and route schedule for their riders. For this hackathon, we need to support the company to design the system.



2. Challenges and Datasets

The Hackathon challenges can be referred to two datasets provided: *riders.csv* and *orders.csv*.

2.1 Orders

All the orders from customers in Singapore are presented in the data file, *orders.csv*. A snapshot for the file is presented as below.

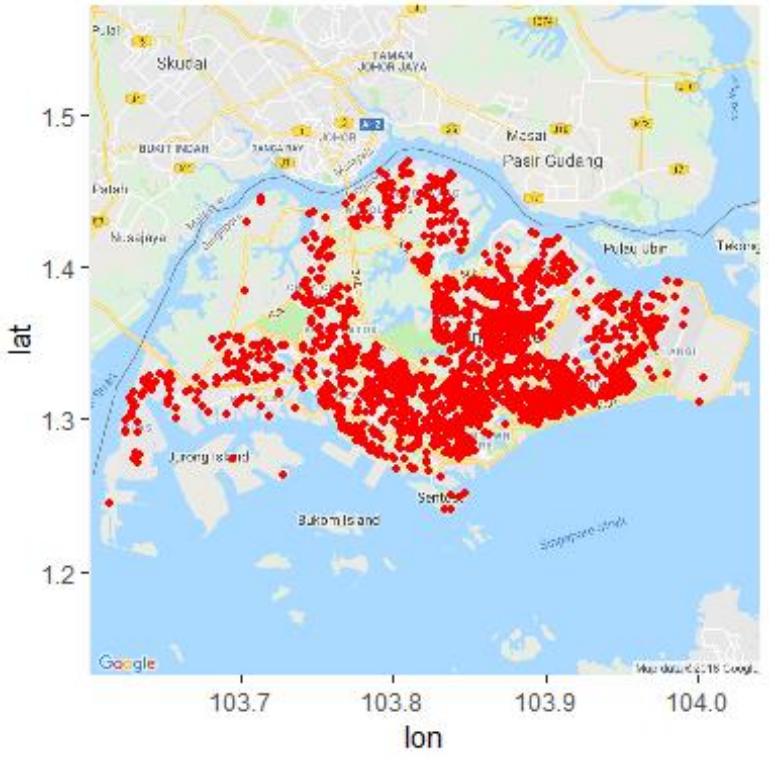
```
orders <- read.csv("orders.csv",row.names=1)
head(orders)
```

```
##   Postal_Code Longitude Latitude
## 1      100082  103.8079   1.2739
## 2      100108  103.8103   1.2810
## 3      101103  103.8134   1.2807
## 4      109344  103.8048   1.2805
## 5      109691  103.8030   1.2822
## 6      109918  103.8112   1.2682
```

```
#The size of the dataset
dim(orders)
```

```
## [1] 2965    3
```

The distribution of the orders is shown in Red on the map.



2.2 Riders

The data file, riders.csv, includes location information of riders who are available to take up new deliveries. A snapshot is presented as below.

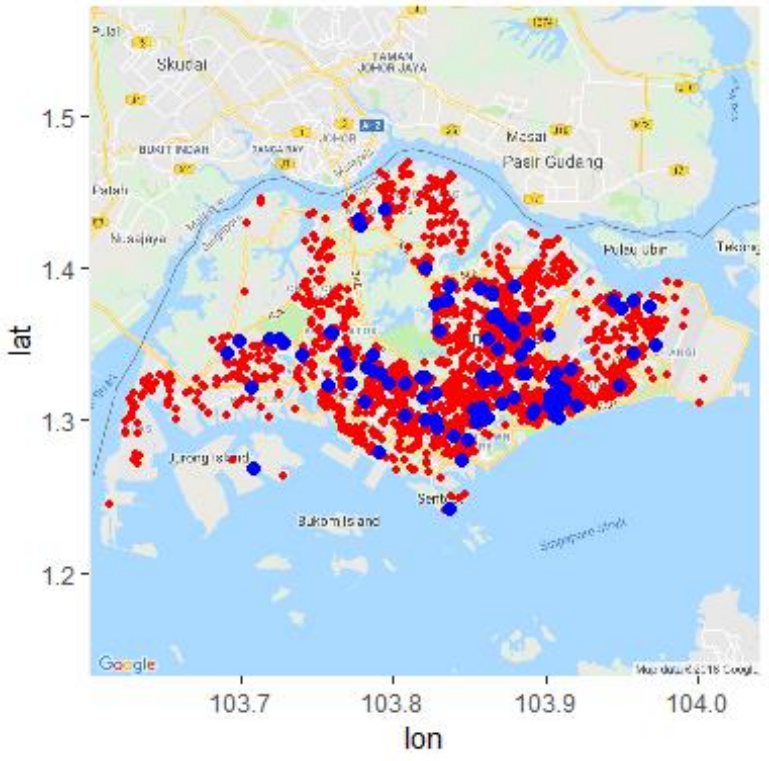
```
riders <- read.csv("riders.csv",row.names=1)
head(riders)
```

```
##      Postal_Code Longitude Latitude
## 5001      118109  103.7907   1.2795
## 5002      120117  103.7712   1.3245
## 5003      129168  103.7564   1.3229
## 5004      188971  103.8536   1.2992
## 5005      189862  103.8585   1.3011
## 5006      199572  103.8617   1.3016
```

```
#The size of the dataset
dim(riders)
```

```
## [1] 100   3
```

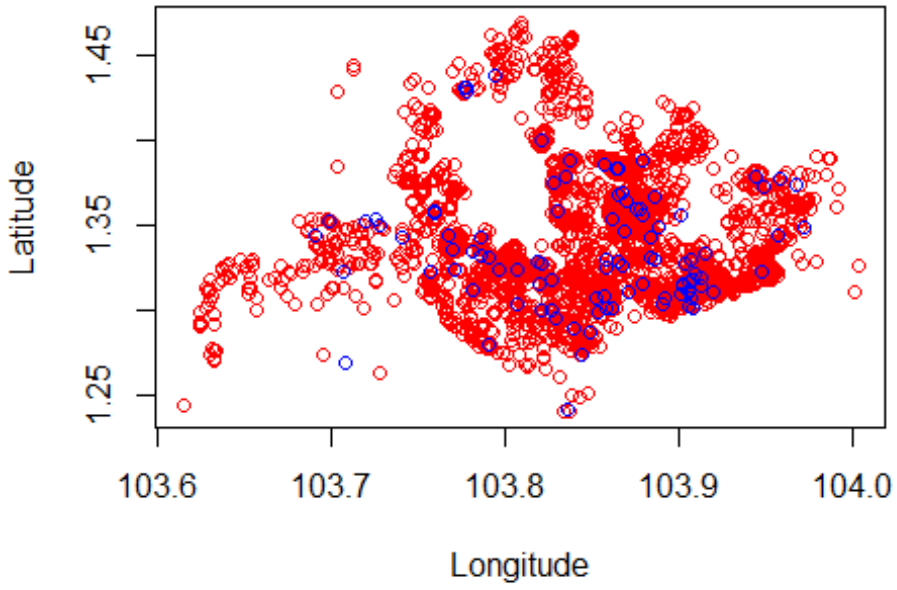
The distribution of the riders is visualized in blue on the map.



2.3 Display by simple plot

To simplify the development, we remove the less important information, and could just use the simple *plot* and *points* functions to present the spatial data. You may study the details in *Help*.

```
plot(orders[,2:3], col="red")
points(riders[,2:3], col="blue")
```



3 Goals and Deliverables

Your goal is to come up with an automated system providing EAT DELIVERY with the case assignments for each rider. The automated case assignment system will address the challenges as below:

1. How many orders will be assigned to each rider?
2. Which orders will be assigned for each rider?
3. What are the best routes for all riders to fulfill the orders?
4. How many percentages of customer orders are delivered on time, providing that the service promise of food order deliver is no more than one hour?

4. Assumptions

The case is provided by a local anonymous company and is further revised to fit for the purpose of this hackathon. To reduce the complexity of of the problem, we have made several assumptions.

Assumption 1:

Each rider is at his/her current location, where the restaurants are located and have the unlimited food supply to fulfil all customer orders. Therefore, you do not have to consider the case of the riders having to visit the restaurants to pick up food once the orders are assigned. After the riders have delivered all orders assigned, they do not need to return to the original starting point.

Assumption 2:

The actual travel paths along roads are not considered. The distance of the path between any two location points is only calculated by Havensine Distance Function presented as below.

Havensine Distance Function (in meter)

- long: Ending Longitude
- lat: Ending Latitude
- slong: Starting Longitude
- slat: Starting Latitude

```
Havensine <-function(slong,slat,long,lat){
  a=6378137*acos(

cos((pi/180)*lat)*cos((pi/180)*long)*cos((pi/180)*slat)*cos((pi/180)*slong)
    +
cos((pi/180)*lat)*sin((pi/180)*long)*cos((pi/180)*slat)*sin((pi/180)*slong)+
      sin((pi/180)*lat)*sin((pi/180)*slat));
  round(a,3)
}

#Test the function
Havensine(103.7911,1.2797,103.7712,1.3243)

## [1] 5436.411
```

Alternatively, the *geosphere::distHaversine()* function may be used to calculate the Havensine distance. Make sure that your package has been installed by executing *install.packages("geosphere")* before it is used.

```
library("geosphere")
distHaversine(c(103.7911,1.2797),c(103.7712,1.3243))

## [1] 5436.411
```

Assumption 3:

Each order takes **1.5 minutes** to reach the customer when the rider arrives at the customer's building. The riders' travelling speed is **48km/hr** between two locations.

Traffic lights and traffic congestion are not considered. The company's promise to the customers is to deliver the orders within one hour.

Others

You may make more assumptions for the system if it is needed.

5. A Possible Solution Guide

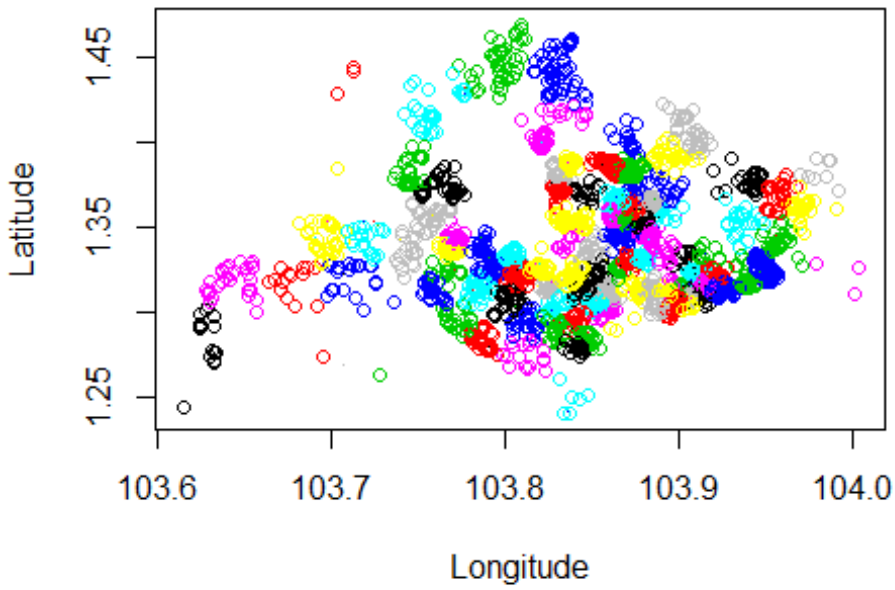
There are a number of methods to solve the problems. The following demonstration illustrates one of the methods to find the solution with the purpose for you to understand the challenges better. There should be the other better solutions for you to explore.

5.1. Divide the orders data into different subsets

We may develop or apply a method to divide the orders dataset into different subsets assigned to the dedicated riders.

For example, we may use some clustering methods such as k-means which produces the clustering result as below.

```
plot(orderPoints, col=groupIndices+1)
points(riders[,2:3], col=1:100,pch = ".")
```



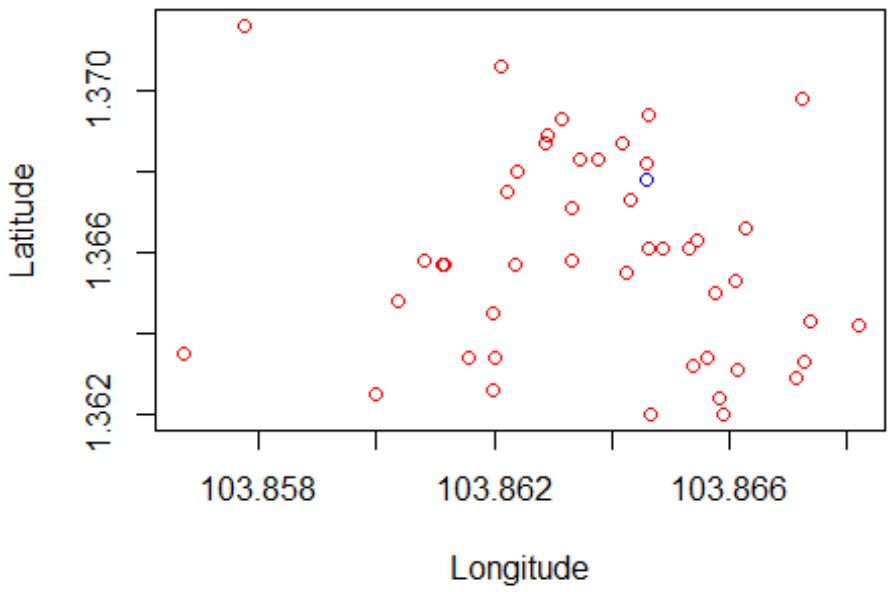
The Rider 5068 is taken as an example. The orders assigned to him/her may be like in this way.

```
ci = 68; # cluster index is 68 with respect to rider 5068
(clusterMembers=orderPoints[groupIndices==ci,])
```

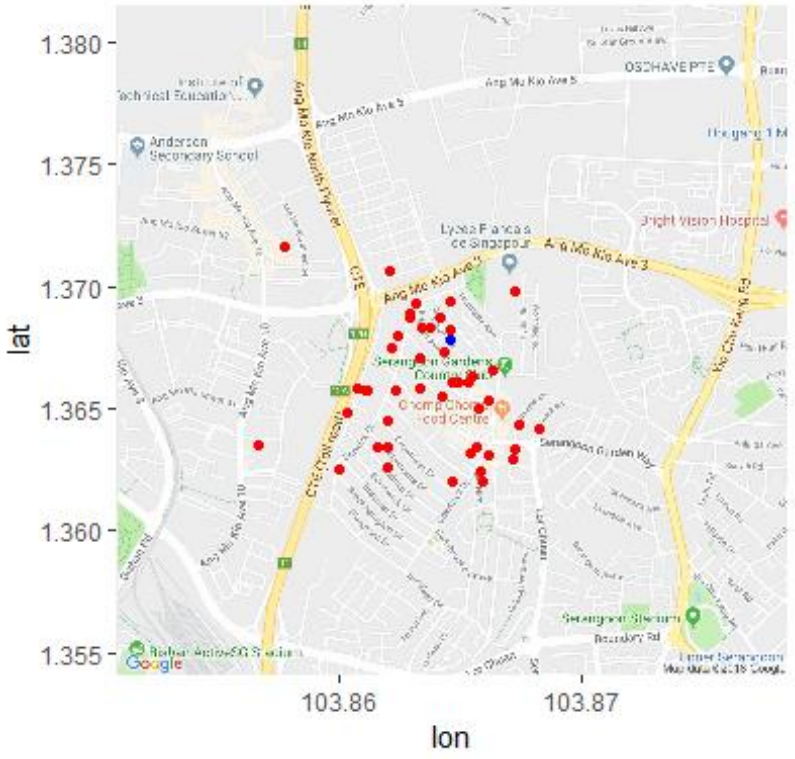
```
##      Longitude Latitude
## 1849  103.8604   1.3648
## 1850  103.8612   1.3657
## 1851  103.8611   1.3657
## 1852  103.8608   1.3658
## 1858  103.8658   1.3624
## 1859  103.8659   1.3620
## 1865  103.8674   1.3643
## 1867  103.8622   1.3675
## 1868  103.8624   1.3680
## 1869  103.8629   1.3687
## 1870  103.8629   1.3689
## 1871  103.8632   1.3693
## 1887  103.8673   1.3633
## 1888  103.8656   1.3634
## 1889  103.8623   1.3657
## 1896  103.8646   1.3661
## 1897  103.8633   1.3671
## 1899  103.8654   1.3632
## 1901  103.8658   1.3650
## 1912  103.8661   1.3653
## 1913  103.8663   1.3666
## 1915  103.8672   1.3698
## 1916  103.8646   1.3682
## 1917  103.8646   1.3694
## 1918  103.8642   1.3687
## 1921  103.8642   1.3655
## 1922  103.8633   1.3658
## 1923  103.8647   1.3620
## 1924  103.8620   1.3645
## 1925  103.8655   1.3663
## 1926  103.8638   1.3683
## 1932  103.8671   1.3629
## 1936  103.8661   1.3631
## 1952  103.8600   1.3625
## 1956  103.8653   1.3661
## 1957  103.8649   1.3661
## 1958  103.8643   1.3673
## 1959  103.8634   1.3683
## 1962  103.8620   1.3626
## 1963  103.8620   1.3634
## 1964  103.8616   1.3634
## 1969  103.8682   1.3642
## 1974  103.8567   1.3635
## 1978  103.8577   1.3716
## 2009  103.8621   1.3706
```

We visualize the assignment on the map by the basic *plot* and *points* functions.

```
plot(clusterMembers, col="red") # The location of delivery orders assigned to
the rider
points(ridersPoints["5068",], col="blue") # the location of the Rider 5068
```



Alternatively, the cluster is visualized on the map.



Is it easy for us to recommend the tour for this single case? How can we design a solution to recommend the optimal tour for Rider 5068 to reach all the order points with the least effort?

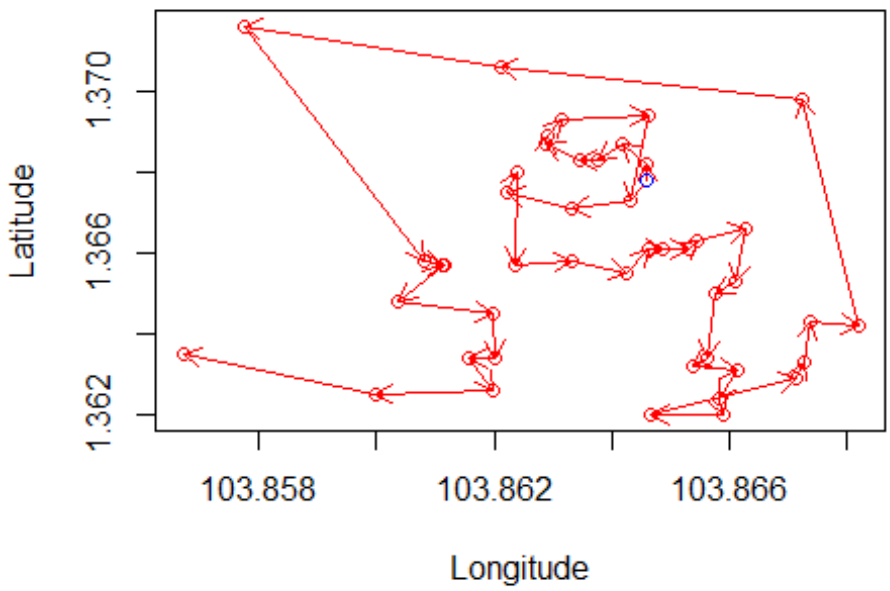
5.2. Best routes

We may refer to *TSP package* to provide the solution as below by applying Nearest Neighbor method.

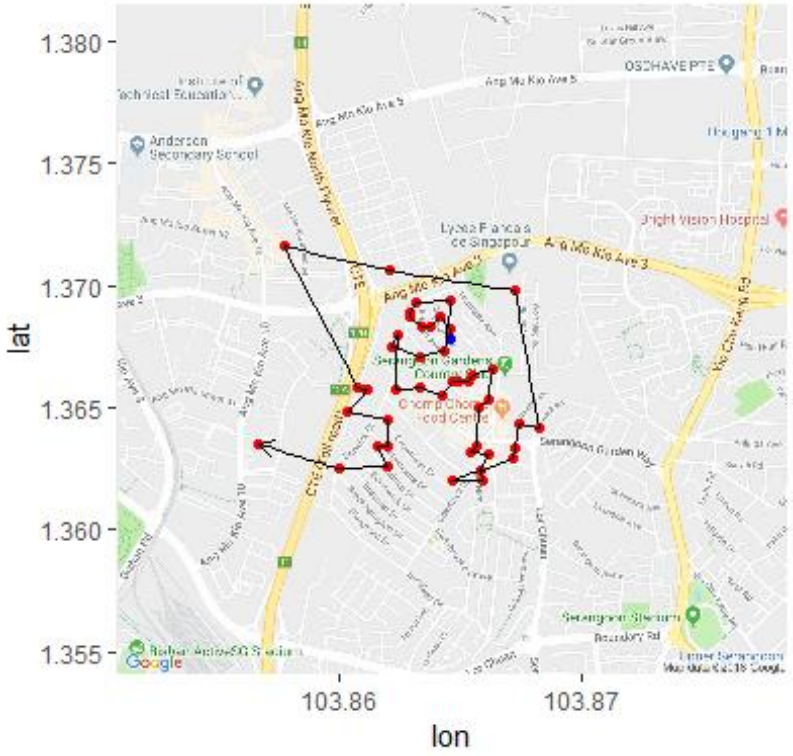
```
##  [1] "5068" "1916" "1918" "1926" "1959" "1869" "1870" "1871" "1917" "1958"
## [11] "1897" "1867" "1868" "1889" "1922" "1921" "1896" "1957" "1956" "1925"
## [21] "1913" "1912" "1901" "1888" "1899" "1936" "1858" "1859" "1923" "1932"
## [31] "1887" "1865" "1969" "1915" "2009" "1978" "1852" "1851" "1850" "1849"
## [41] "1924" "1963" "1964" "1962" "1952" "1974"
```

The route can be visualized on the graph.

```
plot(resultPath, col="red")
s <- seq(nrow(resultPath)-1)  # one shorter than data
x=resultPath[,1]
y=resultPath[,2]
arrows(x[s], y[s], x[s+1], y[s+1], col = "red",length = 0.1)
points(ridersPoints["5068",], col="blue")
```



The route is visualized on the map in this way.



However, there should be the other better method to solve this. You may use TSP or find the other package and configure the settings to provide the better solution. Or more credits will be given if you develop and use your own new algorithms.

5.3. The Solution Formats

We perform the similar calculation for the rest riders. A table for the solution the Delivery Orders Assignment and Routing Information for all Riders is generated.

Once you have the results in the data table, You need to generate two files. Providing that you define resultTable as the data.frame variable for the result, you may use the R functions below to generate the files which will be submitted.

```
#save(resultTable,file="resultTable.Rda") # remove "#"" when the function is
needed
#write.csv(resultTable,file="resultTable.csv",row.names = FALSE)
```

The details of the structure of the solution data format are prestented as below.

```
dim(resultTable)

## [1] 100   5

str(resultTable)

## 'data.frame':    100 obs. of  5 variables:
##  $ RiderID       : Factor w/ 100 levels "5001","5002",..: 1 2 3 4 5 6 7 8
9 10 ...
##  $ OrderSetID    : chr
"49,20,21,47,48,53,52,22,23,17,16,46,19,38,39,40,41,11,32,8,18,12,33,34,15,25
,24,28,26,27,29,42,31"
"2227,2228,2229,2182,2177,2178,2174,2179,2175,2183,2184,55,2181,2180,2176,215
7,98,97,56"
"96,101,100,99,81,63,80,78,65,73,74,77,76,70,69,57,79,88,62,87,86,85,91,93,83
,94,92,60,61,84,90,89,95,75,72,2245"
"170,162,216,238,239,240,241,205,204,249,203,222,225,224,248,247,242,243,255,
256,282,283,268,269,201,202,212,245"| __truncated__ ...
##  $ Total_Distance: num  7530 5682 10255 6133 4888 ...
##  $ No_Of_Orders  : int  33 19 36 45 37 3 45 25 20 27 ...
##  $ Total_time    : num  58.9 35.6 66.8 75.2 61.6 ...
```

A snapshot of some riders' solutions is demonstrated as below.

```
kable(resultTable[c(6,45,89,100),],row.names = FALSE)
```

RiderID	OrderSetID	Total_Distance	No_Of_Orders	Total_time
5006	859,861,1118	3047.146	3	8.308933
5045	961,960,894,882,883,1112,886,895	1693.842	8	14.117303
5089	2534,2533,2532	9573.965	3	16.467457
5100	2958,2957,2961,2960,2964,2959	4867.267	6	15.084084

5.4. Customer Service Promise

On the basis of the results in the table, we can calculate how many late delivery orders (i.e. *more than one hour*) to the customers. A example of the code below is used to find how many **riders** who could not complete the orders on time.

```
sum(resultTable$Total_time>60)

## [1] 36
```

You need to take close looks into the order assignments and routes to make further improvement in order to minimize the number of late delivery orders and delivery time to the customers.

6. Evaluation

6.1 Your Solution

In addition to the four criteria stated in Section 3, you may suggest additional measurement criteria to evaluate your approach and outputs. Extra credits will be given if the team can provide better solution by changing or making some assumptions to make the solution more realistic and practical.

Whilst R is encouraged, you may use your own programming language. Please make sure that your programme can at least produce a csv file with the required format presented in Table 3.

6.2 Other Criteria

The evaluation criteria also include the quality of your code, report and your possible presentation on the Finals. Each should be clear and concise.

7. Submission

You are encouraged to submit the solution by solving the problem as much as you can. You need to submit a *report* and your codes. The *Rda* and *csv* files with the required solution format are needed to be submitted. You can define your own report format. The report should not be more than *8 pages*. Please submit all codes, any files generated, and a report to sbiz_anl@suss.edu.sg . The timeline is as below.

- 10 September 2018 (12:00pm): Submission deadline. Any late submission will not be considered.
- 12 September 2018: Shortlisted teams will be notified.
- 15 September 2018(9.00am – 1.00pm):Finals. Presentation for shortlisted team and prize giving ceremony.