Q1：(A) Show the top 30 most frequent occurring words and their average occurrences in a sentence.
(B) According to the result, what are the characteristics of these words?

(A)

在 Hadoop 的平台下，此題我以 master = local 的 Pyspark 設定進行運算：

```
from hdfs import Client
HDFSHOST = "http://192.168.83.130:9870"
FILENAME = "./Youvegottofindwhatyoulove.txt"
```

```
import findspark
findspark.init('/opt/spark')
```

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
spark = SparkSession.builder.master("local").getOrCreate()
sc = SparkContext.getOrCreate()
```

```
sc
```

SparkContext

Spark UI

Version
`v2.4.8`

Master
`local`

AppName
`pyspark-shell`

根據題目的要求，我首先將文字依據空格做斷字，並且計算各個單字出現的次數另外，我使用了 sortBy 函數，並且對字數負數進行排序，便能將個數由大至小排列：

```
Data = sc.textFile('./input/Youvegottofindwhatyoulove.txt')
data = Data.flatMap(lambda line: line.split())
wordcounts = data.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)
wordcounts = wordcounts.sortBy(lambda x: -x[1])
count = wordcounts.collect()
```

```
count
```

```
[(u'the', 90),
 (u'I', 86),
 (u'to', 71),
 (u'and', 49),
 (u'was', 47),
 (u'a', 46),
 (u'of', 40),
 (u'that', 38),
 (u'in', 33),
 (u'is', 28),
 (u'it', 27),
 (u'you', 27),
 (u'my', 25),
 (u'had', 22),
 (u'with', 18),
 (u'It', 18),
 (u'have', 17),
 (u'And', 17),
 (u'for', 17),
```

計算句子數量，我是將句號作為句子的結束點，因此藉由句號做”斷句”，計算出有幾個句子，在從中算出前三十名常見的單字在句子中平均出現幾次：

```
data1 = Data.flatMap(lambda line: line.split('.'))
N = data1.count()
```

```
average = list()
for word in words:
    n = data.filter(lambda line: line == word).count()
    average.append((word, float(n)/N))
```

```
average
```

```
[('the', 0.46632124352331605),
 ('I', 0.44559585492227977),
 ('to', 0.36787564766839376),
 ('and', 0.2538860103626943),
 ('was', 0.24352331606217617),
 ('a', 0.23834196891191708),
 ('of', 0.20725388601036268),
 ('that', 0.19689119170984457),
 ('in', 0.17098445595854922),
 ('is', 0.14507772020725387),
 ('it', 0.13989637305699482),
 ('you', 0.13989637305699482),
 ('my', 0.12953367875647667),
 ('had', 0.11398963730569948),
 ('with', 0.09326424870466321),
 ('It', 0.09326424870466321),
 ('have', 0.08808290155440414),
 ('And', 0.08808290155440414),
 ('for', 0.08808290155440414),
 ('all', 0.08290155440414508),
 ('what', 0.07253886010362694),
 ('out', 0.07253886010362694),
 ('your', 0.07253886010362694),
 ('as', 0.07253886010362694),
 ('be', 0.06735751295336788),
 ('from', 0.06735751295336788),
 ('me', 0.06217616580310881),
 ('on', 0.06217616580310881),
 ('so', 0.05699481865284974),
```

最終結果為：

```
[('the', 0.46632124352331605),
 ('I', 0.44559585492227977),
 ('to', 0.36787564766839376),
 ('and', 0.2538860103626943),
 ('was', 0.24352331606217617),
 ('a', 0.23834196891191708),
 ('of', 0.20725388601036268),
 ('that', 0.19689119170984457),
 ('in', 0.17098445595854922),
 ('is', 0.14507772020725387),
 ('it', 0.13989637305699482),
 ('you', 0.13989637305699482),
 ('my', 0.12953367875647667),
 ('had', 0.11398963730569948),
 ('with', 0.09326424870466321),
 ('It', 0.09326424870466321),
 ('have', 0.08808290155440414),
 ('And', 0.08808290155440414),
 ('for', 0.08808290155440414),
 ('all', 0.08290155440414508),
```

```
('what', 0.07253886010362694),
('out', 0.07253886010362694),
('your', 0.07253886010362694),
('as', 0.07253886010362694),
('be', 0.06735751295336788),
('from', 0.06735751295336788),
('me', 0.06217616580310881),
('on', 0.06217616580310881),
('so', 0.05699481865284974),
('at', 0.05699481865284974),
```

(B)

從此次實驗中，可以發現到最常出現的字詞幾乎為代名詞、介係詞、關係代名詞等的類型。也是各種句子中最會被頻常使用，作為連接語意的用詞。

Q2：In YARN cluster mode, implement a program to calculate the average amount in credit card trip and cash trip for different numbers of passengers. Explain also how you deal with the data loss issue.

我先設定在 YARN 叢集下的 Pyspark：

```
from pyspark import SparkConf
from pyspark.sql import SparkSession
from pyspark import SparkContext
conf = SparkConf()
conf.setMaster('yarn')
conf.setAppName('spark-yarn')
#conf.set('spark.yarn.dist.files','file:/path/to/pyspark.zip,file:/path/to/py4j-0.8.2.1-src.zip')
#conf.setExecutorEnv('PYTHONPATH','pyspark.zip:py4j-0.8.2.1-src.zip')
sc = SparkContext(conf = conf)
#sc = SparkContext('yarn', 'First App').getOrCreate()
```

```
sc
```

SparkContext

Spark UI

Version
v2.4.8

Master
yarn

AppName
spark-yarn

再藉由 SQL 語法讀取檔案：

```
from pyspark.sql import SQLContext
sql = SQLContext(sc)
df = sql.read.format('com.databricks.spark.csv')\
            .options(header='true')\
            .load('./input/yellow_tripdata_2018-10.csv')
```

```
df.show()
```

```
+--------+--------------------+--------------------+---------------+-------------+----------+------------------+-----------
-+------------+------------+-----------+-----+-------+----------+-----------+--------------------+------------+
|VendorID|tpep_pickup_datetime|tpep_dropoff_datetime|passenger_count|trip_distance|RatecodeID|store_and_fwd_flag|PULocationI
D|DOLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_surcharge|total_amount|
+--------+--------------------+--------------------+---------------+-------------+----------+------------------+-----------
-+------------+------------+-----------+-----+-------+----------+-----------+--------------------+------------+
|       1| 2018-10-01 00:23:34| 2018-10-01 00:44:50|              1|         6.20|         1|                 N|          6
8|           7|           2|       20.5|  0.5|    0.5|         0|          0|                 0.3|        21.8|
|       1| 2018-10-01 00:40:05| 2018-10-01 01:01:56|              1|        12.60|         1|                 N|         13
2|           9|           2|         35|  0.5|    0.5|         0|          0|                 0.3|        36.3|
|       1| 2018-10-01 00:05:35| 2018-10-01 00:19:38|              1|         6.10|         1|                 N|          5
0|         244|           1|         19|  0.5|    0.5|      5.05|          0|                 0.3|       25.35|
|       1| 2018-10-01 00:42:56| 2018-10-01 00:49:00|              1|         1.30|         1|                 N|         15
1|         239|           2|          7|  0.5|    0.5|         0|          0|                 0.3|         8.3|
|       1| 2018-10-01 00:19:14| 2018-10-01 00:31:54|              1|         2.60|         1|                 N|         23
3|         143|           1|         11|  0.5|    0.5|      2.45|          0|                 0.3|       14.75|
|       1| 2018-10-01 00:36:27| 2018-10-01 01:39:20|              1|        19.70|         1|                 N|         13
2|           7|           3|         54|  0.5|    0.5|         0|          0|                 0.3|        55.3|
|       1| 2018-10-01 00:26:25| 2018-10-01 00:34:58|              1|         2.30|         1|                 N|         26
```

根據題意所要求出的平均數，我使用 filter、groupby 和 agg 函數進行計算，結果如下圖：

```
df.filter(df.payment_type=='1').groupby('passenger_count').agg({'Total_amount':'mean'}).show()
```

```
+---------------+------------------+
|passenger_count| avg(Total_amount)|
+---------------+------------------+
|              7| 64.35026315789474|
|              3|18.397956039882867|
|              8|             74.32|
|              0|18.168499784354147|
|              5| 18.35350646849934|
|              6|18.137538831763177|
|              9| 78.46608695652174|
|              1| 18.09561629483899|
|              4|18.679751424181198|
|              2|18.823159226021037|
+---------------+------------------+
```

```
df.filter(df.payment_type=='2').groupby('passenger_count').agg({'Total_amount':'mean'}).show()
```

```
+---------------+------------------+
|passenger_count| avg(Total_amount)|
+---------------+------------------+
|              7| 53.98666666666666|
|              3|14.810038897804311|
|              8| 72.89999999999999|
|              0|13.700388333166083|
|              5|13.900753015657697|
|              6| 13.79258922650266|
|              9|52.521666666666675|
|              1|13.682682121839113|
|              4|15.506981138430348|
|              2| 14.72907295625041|
+---------------+------------------+
```

由這些函數的做法來看，對於缺失值部份是以忽略處理，也就是不將缺失值放入計算。而這筆資料量也約為 800 萬筆資料，因此做 DropNA 處理捨棄部份資訊，之後再取平均仍會遵守大數法則，使平均數趨近於母體平均數。

Q3：monitor HDFS and YARN metrics through HTTP API; collect MapReduce counters-related information through the web UI; provide screenshots and observations regarding the metrics in your report.

根據 https://docs.qubole.com/en/latest/user-guide/engines/hadoop/yarn/yarn-metrics.html#hdfs-metrics 網站上的定義，找出 Yarn Metrics 和 HDFS Metrics 的資訊如下表：

Metric	Description
yarn.QueueMetrics.AppsCompleted	It denotes the number of completed applications.
yarn.QueueMetrics.AppsPending	It denotes the number of pending applications.
yarn.QueueMetrics.AppsRunning	It denotes the number of running applications.
yarn.QueueMetrics.AppsFailed	It denotes the number of failed applications.
yarn.QueueMetrics.AppsKilled	It denotes the number of killed applications.
yarn.QueueMetrics.ReservedMB	It denotes the size of the reserved memory.
yarn.QueueMetrics.AvailableMB	It denotes the size of the available memory in Mebibytes.
yarn.QueueMetrics.AllocatedMB	It denotes the size of the allocated memory in Mebibytes.
yarn.QueueMetrics.ReservedVCores	It denotes the number of reserved virtual cores.
yarn.QueueMetrics.AvailableVCores	It denotes the number of available virtual cores.
yarn.QueueMetrics.AllocatedVCores	It denotes the number of allocated virtual cores.
yarn.NodeManagerMetrics.ContainersFailed	It denotes the number of containers that have failed.
yarn.NodeManagerMetrics.ContainersRunning	It denotes the number of running containers.
yarn.NodeManagerMetrics.ContainersKilled	It denotes the number of containers that are killed.
yarn.NodeManagerMetrics.ContainersCompleted	It denotes the number of containers that are completed.
yarn.QueueMetrics.AllocatedContainers	It denotes the number of allocated containers.
yarn.QueueMetrics.ReservedContainers	It denotes the number of reserved containers.
yarn.ClusterMetrics.NumActiveNMs	It denotes the number of active NodeManagers.
yarn.ClusterMetrics.NumDecommissionedNM	It denotes the number of decommissioned NodeManagers.
yarn.ClusterMetrics.NumDecommissioningNMs	It denotes the number of decommissioning NodeManagers.
yarn.ClusterMetrics.NumLostNMs	It denotes the number of NodeManagers that are lost.
yarn.ClusterMetrics.NumRebootedNMs	It denotes the number of rebooted NodeManagers.
yarn.ClusterMetrics.NumUnhealthyNMs	It denotes the number of unhealthy NodeManagers.

dfs.FSNamesystem.CapacityTotal	It denotes the total disk capacity in bytes.
dfs.FSNamesystem.CapacityUsed	It denotes the disk usage in bytes.
dfs.FSNamesystem.CapacityRemaining	It denotes the remaining disk space left in bytes.
dfs.FSNamesystem.CapacityUsedGB	It denotes the disk usage in Gigabytes.
dfs.FSNamesystem.CapacityTotalGB	It denotes the total disk capacity in Gigabytes.
dfs.FSNamesystem.TotalLoad	It denotes the total load on the file system.
dfs.FSNamesystem.BlocksTotal	It denotes the total number of blocks.
dfs.FSNamesystem.FilesTotal	It denotes the total number of files.
dfs.FSNamesystem.MissingBlocks	It denotes the number of missing blocks.
dfs.FSNamesystem.CorruptBlocks	It denotes the number of corrupt blocks.
dfs.FSNamesystem.PendingReplicationBlocks	It denotes the number of blocks pending replication.
dfs.FSNamesystem.UnderReplicatedBlocks	It denotes the number of under replicated blocks.
dfs.FSNamesystem.ScheduledReplicationBlocks	It denotes the number of blocks scheduled for replication.
dfs.FSNamesystem.PendingDeletionBlocks	It denotes the number of pending deletion blocks.

因此我盡可能地從 9870、8088、中找出符合的 metrics。

首先，為了建立起 Yarn Metrics 部份，我先建造了 Map code 和 Reduce Code：

【Map Code】

```python
#!/usr/bin/env python
from sys import stdin

flag = 0
# payment_type=="1"
# payment_type=="2"

cols = ['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
        'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag', 'PULocationID',
        'DOLocationID', 'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
        'improvement_surcharge', 'Total_amount']
check = cols.index('payment_type')
I = cols.index('passenger_count')
II = cols.index('Total_amount')

trying = 0
for obj in stdin:
        data = obj.strip().split(',')
        PMT = data[check]
        PC = data[I]
        TA = data[II]
        if PC == cols[check]:
            continue
        print("{0},{1},{2}".format(PMT, PC, TA))
```

【Reduce Code】

```python
#!/usr/bin/env python
from sys import stdin
from operator import itemgetter


K1 = [i for i in range(10)]
K2 = [i for i in range(10)]
N1 = [0 for i in range(10)]
N2 = [0 for i in range(10)]
for obj in stdin:
    OBJ = obj.strip().split(',')
    if OBJ[0] == '1':
        K1[int(OBJ[1])] += float(OBJ[2])
        N1[int(OBJ[1])] += 1
    if OBJ[0] == '2':
        K2[int(OBJ[1])] += float(OBJ[2])
        N2[int(OBJ[1])] += 1

for i in range(1, 3):
    print("Payment_Type:{0}".format(i))
    if i == 1:
        for j in range(10):
            print("Passenger Count {0} : {1}".format(j, K1[j]/N1[j]))
    if i == 2:
        for j in range(10):
            print("Passenger Count {0} : {1}".format(j, K2[j]/N2[j]))
```

之後，再寫 run.sh 和 run2.sh（分別為 hadoop 跟 yarn 的指令）：

【run.sh】

```
hadoop jar "/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.2.jar" \
-mapper "python mapper.py" \
-file /home/hadoop/mapper.py \
-reducer "python reducer.py" \
-file /home/hadoop/reducer.py \
-input "./input/yellow2.txt" \
-output "./output"
```

【run2.sh】

```
yarn jar "/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.2.jar" \
-mapper "python mapper.py" \
-file /home/hadoop/mapper.py \
-reducer "python reducer.py" \
-file /home/hadoop/reducer.py \
-input "./input/yellow2.txt" \
-output "./output"
```

之後再進入 8088 取出此 Yarn Metrics 和 HDFS Metrics：如下圖

【Hadoop】

Application Overview

User:	hadoop
Name:	streamjob1708647051053381459.jar
Application Type:	MAPREDUCE
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	Sat Apr 23 18:54:06 +0000 2022
Launched:	Sat Apr 23 18:54:06 +0000 2022
Finished:	Sat Apr 23 18:55:03 +0000 2022
Elapsed:	56sec
Tracking URL:	History
Log Aggregation Status:	DISABLED
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>

Application Metrics

Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	345602 MB-seconds, 270 vcore-seconds
Aggregate Preempted Resource Allocation:	0 MB-seconds, 0 vcore-seconds

Application Attempt Overview

Application Attempt State:	FINISHED
Started:	Sat Apr 23 18:54:06 +0000 2022
Elapsed:	56sec
AM Container:	container_1650736749029_0010_01_000001
Node:	node:44285
Tracking URL:	History
Diagnostics Info:	
Nodes blacklisted by the application:	-
Nodes blacklisted by the system:	-

Total Allocated Containers: 8

Each table cell represents the number of NodeLocal/RackLocal/OffSwitch containers satisfied by NodeLocal/RackLocal/OffSwitch resource requests.

	Node Local Request	Rack Local Request	Off Switch Request
Num Node Local Containers (satisfied by)	6		
Num Rack Local Containers (satisfied by)	0	0	
Num Off Switch Containers (satisfied by)	0	0	2

【Yarn】

Application Overview

User:	hadoop
Name:	streamjob1414633981842118013.jar
Application Type:	MAPREDUCE
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	Sat Apr 23 18:51:52 +0000 2022
Launched:	Sat Apr 23 18:51:52 +0000 2022
Finished:	Sat Apr 23 18:52:48 +0000 2022
Elapsed:	55sec
Tracking URL:	History
Log Aggregation Status:	DISABLED
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>

Application Metrics

Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	334335 MB-seconds, 261 vcore-seconds
Aggregate Preempted Resource Allocation:	0 MB-seconds, 0 vcore-seconds

Application Attempt Overview

Application Attempt State:	FINISHED
Started:	Sat Apr 23 18:51:52 +0000 2022
Elapsed:	55sec
AM Container:	container_1650736749029_0009_01_000001
Node:	node:34551
Tracking URL:	History
Diagnostics Info:	
Nodes blacklisted by the application:	-
Nodes blacklisted by the system:	-

Total Allocated Containers: 9

Each table cell represents the number of NodeLocal/RackLocal/OffSwitch containers satisfied by NodeLocal/RackLocal/OffSwitch resource requests.

	Node Local Request	Rack Local Request	Off Switch Request
Num Node Local Containers (satisfied by)	7		
Num Rack Local Containers (satisfied by)	0	0	
Num Off Switch Containers (satisfied by)	0	0	2

從中可以看出這些程式所使用的時間記憶體、CPU 以及程式上的執行狀況（如 MapReduce）。另外從 8088 首頁亦可以監控 application 的執行狀況：

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus
application_1650736749029_0010	hadoop	streamjob1708647051053381459.jar	MAPREDUCE		default	0	Sun Apr 24 02:54:06 +0800 2022	Sun Apr 24 02:54:06 +0800 2022	Sun Apr 24 02:55:03 +0800 2022	FINISHED	SUCCEEDED
application_1650736749029_0009	hadoop	streamjob1414633981842118013.jar	MAPREDUCE		default	0	Sun Apr 24 02:51:52 +0800 2022	Sun Apr 24 02:51:52 +0800 2022	Sun Apr 24 02:52:48 +0800 2022	FINISHED	SUCCEEDED

我在 MapReduce 上做的程式測試大致上亦可以從 8088 網頁上監控得知：

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources
10	0	0	10	0	<memory:0 B, vCores:0>	<memory:8 GB, vCores:8>

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unheal
1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

另外從 9870 中，可以找到 Node 的狀況、capacity 的數據等：

Configured Capacity:	19.56 GB
Configured Remote Capacity:	0 B
DFS Used:	3.82 GB (19.53%)
Non DFS Used:	11.62 GB
DFS Remaining:	3.1 GB (15.87%)
Block Pool Used:	3.82 GB (19.53%)
DataNodes usages% (Min/Median/Max/stdDev):	19.53% / 19.53% / 19.53% / 0.00%
Live Nodes	1 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	27
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Sat Apr 23 22:58:42 +0800 2022
Last Checkpoint Time	Sun Apr 24 03:51:54 +0800 2022
Enabled Erasure Coding Policies	RS-6-3-1024k

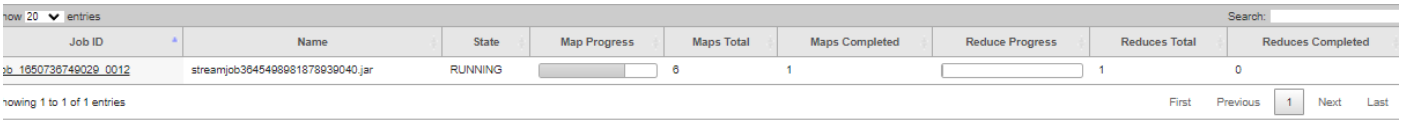
因此，藉由這些網頁的 API 和 UI 介面，我們可以隨時監控 MapReduce 的執行狀況跟結果，最終再用以下程式碼輸出結果：

```
hdfs dfs -cat ./output/part-00000
```

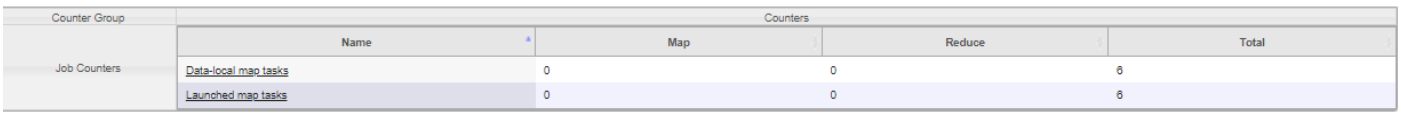
```
Payment_Type:1
Passenger Count 0 : 18.1684997844
Passenger Count 1 : 18.0956165172
Passenger Count 2 : 18.8231614926
Passenger Count 3 : 18.3979681345
Passenger Count 4 : 18.6797903097
Passenger Count 5 : 18.3535253491
Passenger Count 6 : 18.1375763353
Passenger Count 7 : 64.5344736842
Passenger Count 8 : 74.5780645161
Passenger Count 9 : 78.8573913043
```

```
Payment_Type:2
Passenger Count 0 : 13.7003883332
Passenger Count 1 : 13.6826826887
Passenger Count 2 : 14.7290780308
Passenger Count 3 : 14.810064658
Passenger Count 4 : 15.5070485012
Passenger Count 5 : 13.900799162
Passenger Count 6 : 13.7926796588
Passenger Count 7 : 55.1533333333
Passenger Count 8 : 75.5666666667
Passenger Count 9 : 54.0216666667
```

最後對於 MapReduce Counters 部份，由於在我的電腦上無法做儲存 log 的動作，因此我試驗一次並擷取過程的 MapReduce 作為監控結果：



Job ID	Name	State	Map Progress	Maps Total	Maps Completed	Reduce Progress	Reduces Total	Reduces Completed
job_1650736749029_0012	streamjob3645498981878939040.jar	RUNNING		6	1		1	0



Counter Group	Name	Map	Reduce	Total
Job Counters	Data-local map tasks	0	0	6
	Launched map tasks	0	0	6

此次的執行結果亦是 SUCCEED：

application_1650736749029_0012	hadoop	streamjob3645498981878939040.jar	MAPREDUCE	default	0	Sun Apr 24 04:56:27 +0800 2022	Sun Apr 24 04:56:27 +0800 2022	Sun Apr 24 04:57:21 +0800 2022	FINISHED	SUCCEEDED