Noushin Islam Student No. 201508438

Assignment 1 – COMP336

Part 1 Report

Assignment brief:

In this assignment, we are going to use PySpark to analyse the GPS trajectory dataset that was collected in the Geolife project (Microsoft Research Asia) over a period of five years by 100+ people. Each GPS trajectory in this dataset is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude and altitude.

Task 1

Brief:

Any data analysis starts with data cleaning (in fact this typically takes most of the time). In this case, we will convert all the collected dates/times from GMT to India time, where some of these trajectories were collected. This requires to move dates, times and timestamps by 5 and a half hours ahead. You should not create a new input file, but instead use Spark's map/withColumn transformation to change the RDD/DataFrame created from the original file.

Source Code:

```
# 1
print('\nTask 1')
# adding 5.5 hours to Timestamp, Date and Time in all data points

# first combine date and time to DateTime and convert it to timestamp format to add time
dt = df.withColumn('DateTime', concat_ws(' ', col('Date'), col('Time')))
task1 = dt.withColumn('DateTime', to_timestamp('DateTime') + expr('INTERVAL 5 HOURS 30 MINUTES'))
# add the same time to the column named Timestamp
# then extract data and time from DateTime and drop DateTime
task1 = task1.withColumn('Timestamp', col('Timestamp')+(5.5/24))\
        .withColumn('Date', to_timestamp('DateTime').cast('date'))\
        .withColumn('Time', date_format('DateTime', 'HH:mm:ss'))\
        .drop('DateTime')

# uncomment following line to show top 20 rows
# task1.show()
```

Output:

```
Task 1
+------+------------+------------+-------+---------------+-----------------+---------+--------+
|UserID|    Latitude|   Longitude|AllZero|       Altitude|        Timestamp|     Date|    Time|
+------+------------+------------+-------+---------------+-----------------+---------+--------+
|   100|39.974408918|116.303522101|      0|480.287355643045|40753.759861111066|2011-07-29|18:14:12|
|   100|39.974397078|116.303526932|      0|480.121151574803|40753.759872685165|2011-07-29|18:14:13|
|   100|39.973982524|116.303621837|      0|478.499455380577| 40753.75989583336|2011-07-29|18:14:15|
|   100|39.973943291|116.303632641|      0|479.176988188976|40753.759907407366|2011-07-29|18:14:16|
|   100|39.973937148|116.303639667|      0|479.129432414698|40753.759918981465|2011-07-29|18:14:17|
|   100|39.973916715| 116.30363848|      0|479.615278871391| 40753.75993055556|2011-07-29|18:14:18|
|   100|39.973892264|116.303644867|      0|480.506026902887| 40753.75994212966|2011-07-29|18:14:19|
|   100|39.973867401|116.303647142|      0| 481.38750984252|40753.759953703666|2011-07-29|18:14:20|
|   100|39.973836462| 116.30365019|      0|482.008727034121|40753.759965277764|2011-07-29|18:14:21|
|   100|39.973821199|116.303649412|      0|482.325816929134| 40753.75997685186|2011-07-29|18:14:22|
|   100|39.973807136|116.303642951|      0|482.289422572178| 40753.75998842596|2011-07-29|18:14:23|
|   100|39.973791514|116.303638069|      0|482.314536745541|40753.759999999966|2011-07-29|18:14:24|
|   100|39.973782219|116.303626231|      0|482.362713254593|40753.760011574064|2011-07-29|18:14:25|
|   100|39.973774037|116.303619373|      0|482.472345800525| 40753.76002314816|2011-07-29|18:14:26|
|   100|39.973764604|116.303612174|      0|482.716797900262| 40753.76003472226|2011-07-29|18:14:27|
|   100|39.973754251|116.303615089|      0|482.782529527559|40753.760046296266|2011-07-29|18:14:28|
|   100|39.973736535|116.303612592|      0|483.086404199475|40753.760057870364|2011-07-29|18:14:29|
|   100|39.973726284|116.303615942|      0|483.396486220472| 40753.76006944446|2011-07-29|18:14:30|
|   100|39.973717545|116.303614266|      0|483.624166666667| 40753.76008101856|2011-07-29|18:14:31|
|   100|39.973701448|116.303622536|      0|484.271414041995|40753.760092592565|2011-07-29|18:14:32|
+------+------------+------------+-------+---------------+-----------------+---------+--------+
only showing top 20 rows
```

Description:

To clean the data, in our case to correct the time, I first do so by combining "Date" and "Time" to "DateTime". The I convert the latter to timestamp datatype to add 5.5 hours using the expr() function. We also have a column called "Timestamp", to which the time difference is added by adding 5.5/24 as 1 represents 24 hours for this value. To have the correct columns "Date" and "Time" get extracted from "DateTime", after which the latter column is deleted by dropping it.

Task 2

Brief: Calculate for each person, on how many days were there at least two data points recorded for them (count any day with at least two data points). Output the top 10 user IDs according to this measure and its value (as mentioned above, in case of a tie, output the user with the larger ID).

Source Code:

```
# 2
print('\nTask 2')
# calculating for each UserID on how many days the data was recorded at least twice and displaying top 10

# group by UserID, Date and count the data points
task2 = task1.groupBy('UserID', 'Date').count().withColumnRenamed('count', 'data points')
# filter the results to drop any rows with less than than two data points
task2 = task2.filter(col('data points') >= 2).drop('count')
# group by UserID and count the days and sort data
task2 = task2.groupBy('UserID').count().withColumnRenamed('count', 'number of days')\
       .orderBy(col('number of days').desc(), col('UserID').desc())

# output of top 10 users
task2.show(10)
```

Output:

```
Task 2
+------+--------------+
|UserID|number of days|
+------+--------------+
|   128|           905|
|   126|           180|
|   104|           117|
|   115|           116|
|   112|           109|
|   125|            50|
|   101|            45|
|   119|            44|
|   111|            36|
|   122|            27|
+------+--------------+
only showing top 10 rows
```

Description:

To count the data points per user I first group the data by “UserID” and “Date” and then use the count() function. I filter the newly created column to get rid of any days where only one data point has been recorded. Once that is done, the data is grouped by “UserID” to find the number of days per user that have passed the condition. Lastly, the data frame is sorted to obtain the output desired.

Task 3

Brief: Calculate for each person, on how many days there were more than 150 data points recorded for them (count any day with at least 150 data points). Output all user IDs and the corresponding value of this measure.

Source Code:

```
# 3
print('\nTask 3')
# calculating for each UserID on how many days the data was recorded at least 150 times and displaying

# group by UserID, Date and count the data points
task3 = task1.groupBy('UserID','Date').count().withColumnRenamed('count', 'data points')
# filter the results to drop any rows with less than than 150 data points
task3 = task3.filter(col('data points') > 150).drop('count')
# group by UserID and count the days and sort data
task3 = task3.groupBy('UserID').count().withColumnRenamed("count", "number of days")\
        .orderBy(col('number of days').desc(), col('UserID').desc())

# output all users
task3.show(task3.count())
```

Output:

```
Task 3
+------+--------------+
|UserID|number of days|
+------+--------------+
|   128|           862|
|   126|           168|
|   112|            95|
|   115|            90|
|   104|            67|
|   125|            46|
|   119|            41|
|   122|            26|
|   101|            24|
|   130|            22|
|   103|            22|
|   113|            19|
|   111|            19|
|   102|            19|
|   114|            15|
|   110|            10|
|   127|             7|
|   121|             7|
|   105|             7|
|   124|             6|
|   100|             5|
|   123|             4|
|   116|             3|
|   106|             3|
|   129|             2|
|   120|             2|
|   118|             2|
|   109|             2|
|   108|             1|
+------+--------------+
```

Description:

To count the data points per user I first group the data by “UserID” and “Date” and then use the count() function. I filter the newly created column to get rid of any days where less than 150 data points have been recorded. Once that is done, the data is grouped by “UserID” to find the number of days per user that have passed the condition. Lastly, the data frame is sorted to obtain the output desired.

Task 4

Brief: Find for each person, the the northern most point that they reached. Output the top 10 user IDs according to this measure (the more northern the better), its value and the day that was achieved (in case of a tie, output the latest such a day).

Source Code:

```
# 4
print('\nTask 4')
# finding for each UserID the most northern part they reached and displaying top 10

# group by UserID and find max Latitude for each user
task4 = task1.groupBy('UserID').agg(max('Latitude').alias('Latitude'))
# find the corresponding date using .join
# group by UserID and Latitude and choose latest date and sort data
task4 = task4.join(task1, on=['UserID', 'Latitude'], how='inner')\
        .groupBy('UserID', 'Latitude').agg(max('Date').alias('Date'))\
        .orderBy(col('Latitude').desc(), col('Date').desc(), col('UserID').desc())

# output of top 10 users
task4.show(10)
```

Output:

```
Task 4
+------+---------------+----------+
|UserID|       Latitude|      Date|
+------+---------------+----------+
|   128|     58.7654916|2009-03-06|
|   118|        47.6693|2007-05-13|
|   120|     42.5513799|2009-09-19|
|   127|      42.533762|2008-09-30|
|   126|      41.974985|2008-05-02|
|   115|     41.9555633|2008-07-19|
|   111|41.6609333333333|2007-06-17|
|   122|      40.970421|2009-09-02|
|   108|40.6649833333333|2007-10-04|
|   103|     40.5504499|2008-08-30|
+------+---------------+----------+
only showing top 10 rows
```

Description:
Firstly, the data is grouped by “UserID” and the most northern “Latitude” value for every user is filtered. By using the join() function the corresponding date for the latter is found. Then, the data frame is sorted to obtain the output desired.

Task 5

Brief: Calculate for each person, the span of the altitude travelled, i.e., the difference between the highest altitude that he/she reached and the lowest one. Output the top 10 user IDs according to this measure and its value.

Source Code:

```
# 5
print('\nTask 5')
#calculating for each UserID the difference between highest and lowest altitude reached and displaying top 10

# group by UserID and find max and min Altitude
task5 = task1.groupBy('UserID').agg(max('Altitude').alias('maxAltitude'), min('Altitude').alias('minAltitude'))
# calculate the difference between the two values and sort data
task5 = task5.withColumn('Difference', col('maxAltitude') - col('minAltitude'))\
        .select(col('UserID'), col('Difference')).orderBy(col('Difference').desc(),col('UserID').desc())

# output of top 10 users
task5.show(10)
```

Output:

```
Task 5
+------+------------------+
|UserID|        Difference|
+------+------------------+
|   128|          119931.1|
|   115|  43038.0377952756|
|   126|           40646.3|
|   106|  36581.3648293963|
|   101|           35324.8|
|   112|26082.699999999997|
|   103|           25879.3|
|   125|           18429.0|
|   124|           17503.0|
|   111|  15121.3910761155|
+------+------------------+
only showing top 10 rows
```

Description:
Firstly, the data is grouped by “UserID” and the highest and lowest altitude reached by every user is found. After which we calculate the difference between the latter two values. Then, the data frame is sorted to obtain the output desired.

Task 6

Brief: Calculate for each person, the height that they climbed each day (i.e., the sum of all the differences in altititues between two consecutive data points that are positive). For each user output the (latest) day they climbed the most. Also, output the total height climbed by all users on all days.

Source Code:

```
# 6
print('\nTask 6')
#calculating for each UserID the sum of the daily and total heights climbed

# select column and sort them
task6 = task1.select(col('UserID'), col('Date'), col('Timestamp'), col('Altitude'))\
       .orderBy(col('UserID').desc(), col('Timestamp').asc())
# partition by UserID and sort by Timestamp
w = Window.partitionBy('UserID').orderBy(col('Timestamp').asc())
# lag the position and calculate the altitude difference
l = task6.withColumn('prevA', lag('Altitude').over(w))
dist = l.withColumn('height', col('Altitude') - col('prevA'))
# filter the result to get rid of non-positive values
# group by UserID and Date and sum the height for each user and sort data
task6 = dist.filter(col('height') > 0).groupBy('UserID', 'Date').sum('height')\
       .drop('TimeStamp', 'prevA').orderBy(col('sum(height)').desc(), col('Date').desc())
# partition by UserID and sort data
w_2 = Window.partitionBy('UserID').orderBy(col('sum(height)').desc(), col('Date').desc())
# get the latest day day each user climbed the most
task6_1 = task6.withColumn('row', row_number().over(w_2)).filter(col('row') == 1)\
        .drop('row').withColumnRenamed('sum(height)', 'max height')\
        .orderBy(col('max height').desc(), col('UserID').desc())
# sum all the height values to get the total value
task6_2 = task6.select(sum('sum(height)')).withColumn('Total height climbed', col('sum(sum(height))'))\
        .drop('sum(sum(height))')

# output all users
task6_1.show(task6_1.count())
print('\n')
# output all
task6_2.show(task6_2.count())
```

Output:

```
Task 6
+------+----------+------------------+
|UserID|      Date|        max height|
+------+----------+------------------+
|   128|2009-11-02|107768.90000000011|
|   124|2008-10-02|           59352.0|
|   106|2007-10-09| 56893.04461942263|
|   103|2008-09-12|42513.500000000044|
|   115|2008-09-13|34951.300000000076|
|   101|2008-03-28|30593.699999999997|
|   112|2008-05-24|25255.699999999924|
|   126|2008-06-01| 22545.79999999999|
|   125|2008-09-17|           21067.0|
|   127|2008-09-30|           20512.0|
|   105|2007-10-07|14612.860892388431|
|   119|2008-08-29|           12803.0|
|   130|2009-07-13|12579.072217847774|
|   123|2009-09-27|11829.078612204767|
|   122|2009-06-14|11326.875761154866|
|   129|2008-05-27|10570.115485564304|
|   118|2007-05-20|  9895.01312335956|
|   111|2007-06-16| 9517.716535433077|
|   121|2009-10-08| 4816.999934383195|
|   108|2007-10-04|4051.8372703411997|
|   113|2010-05-31|           2806.0|
|   107|2007-10-02|1801.1811023622045|
|   100|2011-08-09|1376.9904527559036|
|   114|2007-10-19| 1295.931758530187|
|   116|2011-08-03| 931.9613681102298|
|   117|2007-06-29| 636.4829396325456|
+------+----------+------------------+


+-------------------+
|Total height climbed|
+-------------------+
|   4821694.733064306|
+-------------------+
```

Description:
Firstly, the data selected is partitioned by “UserID” and sorted by time. To be able to calculate the difference in altitude between two consecutive data points we lag the position and calculate it. The result then gets filtered to get rid of any non-positive values. We then group the data by “UserID” and “Date” and sum the heights calculated for each user. After having sorted the data, we partition it once again by “UserID”. By using the row_number() function we filter the latest day each user has climbed the most. By summing all the summed height values of each user we get the total height climbed by all users.