TDT4225 Assignment 2 MySQL

Group 45
Group member(s): Simen Refsland

October 5, 2023

NOTE! In addition to the libraries provided in requirements.txt, I've also used NumPy to speed up some vectorized calculations in task 8 part 2. According to a TA in Piazza, Pandas should be fine, so I assume it's fine to use NumPy.

Introduction

In this assignment, I was tasked with cleaning raw data and then inserting structured data from the Geolife dataset. This involved combining and extracting info present in the raw .plt files in such a way that it was compatible with the User, Activity and TrackPoint tables specified. In the next part, several different queries were written to answer questions about the data, sometimes involving a combination of SQL queries and manual processing in Python.

I have not used the virtual machines to run MySQL, the testing database is local. Since I was the only one in the group, I have worked locally, as opposed to using Git.

Results

Part 1

NOTE! When I discard files that are longer than 2506 lines (including header lines), it should be noted that the lines are counted where there is actual text, meaning I don't count the last line which is empty. This *may* slightly affect the number of activities that are inserted. I also assume that we discard activities that have more than 2500 lines, as there would be no need to check if the .plt file exceeds 2500 lines (you could just take the first 2506 lines otherwise). Some transportation modes are also discarded if they don't fit the start and end date of an activity.

```
User table:
  id    has_labels
----  ------------
 000             0
 001             0
 002             0
 003             0
 004             0
 005             0
 006             0
 007             0
 008             0
 009             0
```

Figure 1: First 10 users

Activity table:

id	user_id	transportation_mode	start_date_time	end_date_time
1	000		2008-10-23 02:53:04	2008-10-23 11:11:12
2	000		2008-10-24 02:09:59	2008-10-24 02:47:06
3	000		2008-10-26 13:44:07	2008-10-26 15:04:07
4	000		2008-10-27 11:54:49	2008-10-27 12:05:54
5	000		2008-10-28 00:38:26	2008-10-28 05:03:42
6	000		2008-10-29 09:21:38	2008-10-29 09:30:28
7	000		2008-10-29 09:30:38	2008-10-29 09:46:43
8	000		2008-11-03 10:13:36	2008-11-03 10:16:01
9	000		2008-11-03 23:21:53	2008-11-04 03:31:08
10	000		2008-11-10 01:36:37	2008-11-10 03:46:12

Figure 2: First 10 activities

Track point table:

id	activity_id	lat	lon	altitude	date_days	date_time
1	1	39.9847	116.318	492	39744.1	2008-10-23 02:53:04
2	1	39.9847	116.318	492	39744.1	2008-10-23 02:53:10
3	1	39.9847	116.318	492	39744.1	2008-10-23 02:53:15
4	1	39.9847	116.318	492	39744.1	2008-10-23 02:53:20
5	1	39.9847	116.318	492	39744.1	2008-10-23 02:53:25
6	1	39.9846	116.318	493	39744.1	2008-10-23 02:53:30
7	1	39.9846	116.318	493	39744.1	2008-10-23 02:53:35
8	1	39.9846	116.318	496	39744.1	2008-10-23 02:53:40
9	1	39.9845	116.317	500	39744.1	2008-10-23 02:53:45
10	1	39.9846	116.317	505	39744.1	2008-10-23 02:53:50

Figure 3: First 10 trackpoints

Part 2

Task 1

Counts the number of rows of each table user, activity and track_point.

Task 1: Number of users, activities and trackpoints in the database:

user_count	activity_count	trackpoint_count
182	16048	9681756

Figure 4: Task 1 result

Task 2

First counts the trackpoints per user, and then uses that table to average, min and max the number of trackpoints.

Task 2: Average, minimum and maximum number of trackpoints logged by the users:

average_count	minimum_count	maximum_count
55963.9	17	1010325

Figure 5: Task 2 result

Task 3

Simple counting of activites grouped by user_id and ordered from highest to lowest.

Task 3: Top 15 users with the most activities logged:

user_id	activity_count
128	2102
153	1793
025	715
163	704
062	691
144	563
041	399
085	364
004	346
140	345
167	320
068	280
017	265
003	261
014	236

Figure 6: Task 3 result

Task 4

Fetches the distinct user_id's that have logged taking the bus as transportation_mode in at least one of the activities.

Task 4: Users that have logged taking the bus:

user_id
010
052
062
073
081
084
085
091
092
112
125
128
175

Figure 7: Task 4 result

Task 5

Very similar to task 3, but we count the distinct transportation modes instead.

Task 5: Top 10 users with most types of different transportation modes:

user_id	transportation_count
128	9
062	7
085	4
084	3
058	3
163	3
078	3
081	3
112	3
065	2

Figure 8: Task 5 result

Task 6

No results found, but returns one result if I add an activity with the same user id, transportation mode and start and end date.

```
Task 6: Activities that are logged twice:
user_id    transportation_mode    start_date_time    end_date_time
---------  ---------------------  -----------------  ---------------

(0.0000 seconds)
```

Figure 9: Task 6 result

Task 7

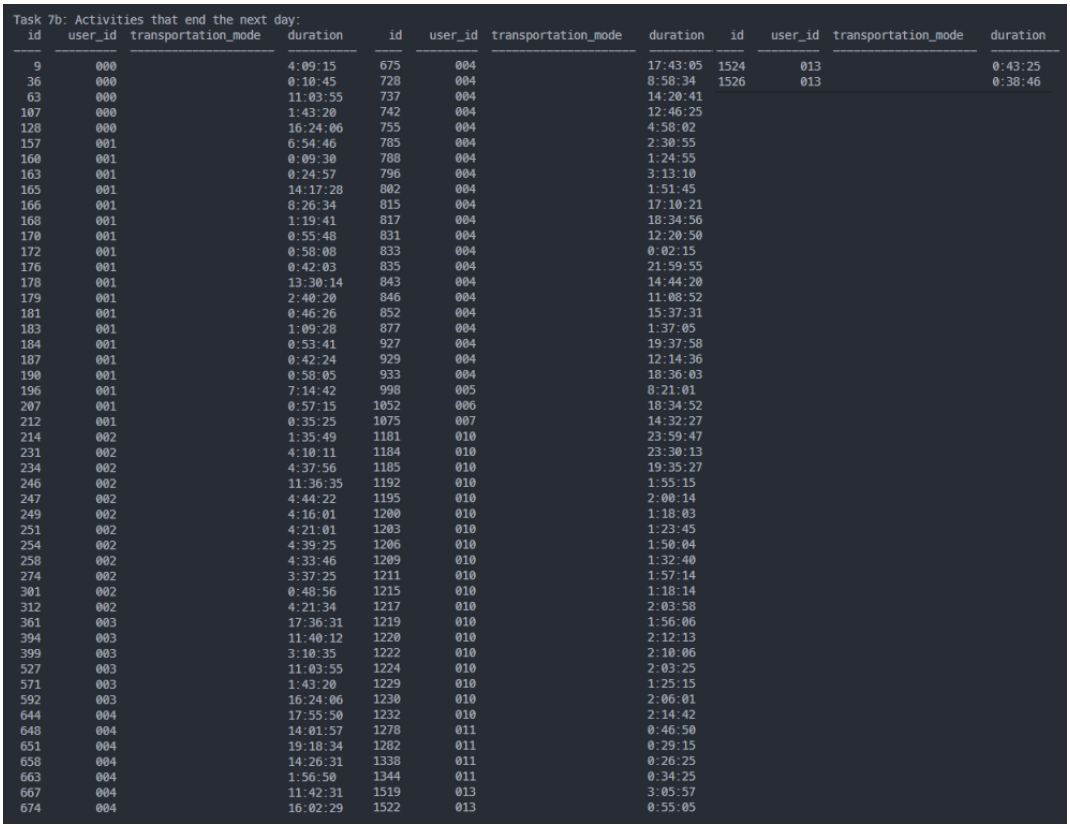
Assumption here is that the difference in dates (by days) between start_date_time and end_date_time is 1, meaning that if an activity for example starts 2007-08-09 23:59:35 and ends 2007-08-10 00:35:34, it counts as ending the next day.

```
Task 7a: Number of users with activities that end the next day:
  COUNT(DISTINCT user_id)
-------------------------
                      98

(0.0000 seconds)
```

Figure 10: Task 7a result

In 7b), I get 1011 results, which would be very cumbersome to insert as results here due to the way the results are printed, so I limit the results to the first 100 results.



Task 7b: Activities that end the next day:

id	user_id	transportation_mode	duration	id	user_id	transportation_mode	duration	id	user_id	transportation_mode	duration
9	000		4:09:15	675	004		17:43:05	1524	013		0:43:25
36	000		0:10:45	728	004		8:58:34	1526	013		0:38:46
63	000		11:03:55	737	004		14:20:41				
107	000		1:43:20	742	004		12:46:25				
128	000		16:24:06	755	004		4:58:02				
157	001		6:54:46	785	004		2:30:55				
160	001		0:09:30	788	004		1:24:55				
163	001		0:24:57	796	004		3:13:10				
165	001		14:17:28	802	004		1:51:45				
166	001		8:26:34	815	004		17:10:21				
168	001		1:19:41	817	004		18:34:56				
170	001		0:55:48	831	004		12:20:50				
172	001		0:58:08	833	004		0:02:15				
176	001		0:42:03	835	004		21:59:55				
178	001		13:30:14	843	004		14:44:20				
179	001		2:40:20	846	004		11:08:52				
181	001		0:46:26	852	004		15:37:31				
183	001		1:09:28	877	004		1:37:05				
184	001		0:53:41	927	004		19:37:58				
187	001		0:42:24	929	004		12:14:36				
190	001		0:58:05	933	004		18:36:03				
196	001		7:14:42	998	005		8:21:01				
207	001		0:57:15	1052	006		18:34:52				
212	001		0:35:25	1075	007		14:32:27				
214	002		1:35:49	1181	010		23:59:47				
231	002		4:10:11	1184	010		23:30:13				
234	002		4:37:56	1185	010		19:35:27				
246	002		11:36:35	1192	010		1:55:15				
247	002		4:44:22	1195	010		2:00:14				
249	002		4:16:01	1200	010		1:18:03				
251	002		4:21:01	1203	010		1:23:45				
254	002		4:39:25	1206	010		1:50:04				
258	002		4:33:46	1209	010		1:32:40				
274	002		3:37:25	1211	010		1:57:14				
301	002		0:48:56	1215	010		1:18:14				
312	002		4:21:34	1217	010		2:03:58				
361	003		17:36:31	1219	010		1:56:06				
394	003		11:40:12	1220	010		2:12:13				
399	003		3:10:35	1222	010		2:10:06				
527	003		11:03:55	1224	010		2:03:25				
571	003		1:43:20	1229	010		1:25:15				
592	003		16:24:06	1230	010		2:06:01				
644	004		17:55:50	1232	010		2:14:42				
648	004		14:01:57	1278	011		0:46:50				
651	004		19:18:34	1282	011		0:29:15				
658	004		14:26:31	1338	011		0:26:25				
663	004		1:56:50	1344	011		0:34:25				
667	004		11:42:31	1519	013		3:05:57				
674	004		16:02:29	1522	013		0:55:05				

Figure 11: Task 7b result

Task 8

To reduce the massive number of potential matches of trackpoints, activities of user x and y are first filtered out if they overlap in start_date_time and end_date_time, and if their bounding boxes overlap (as defined as min and max latitude and longitude). The result is pairs of activities that overlap, and if any of their respective trackpoints fit the criteria (within 50 m and 30 sec), the users have been close in time and space. Despite this filtering, the program ran for $\approx$ 2 hours, so there's probably something that I'm missing.

```
Task 8: Number of users that have been close to each other: 121
```

Figure 12: Task 8 result

Task 9

Row numbering (window function ROW_NUMBER() partitioned by activity id) is used to consecutively number the trackpoints in an activity after invalid entries are removed. Previous attempts used the id, but this skips linking the current valid altitude to the next. For example, if trackpoint with id 7 is removed, the gap must be bridged between id 6 and id 8. I've noticed that there are several dubious recorded altitudes (such as -1000 feet), but I've based my answer on the assumption that only -777 is an invalid altitude, since it is possible to be below sea level, where a filtering of altitude < 0 would be logical if not. All results are in meters by converting feet to meter by a constant.

```
Task 9: Top 15 users with the highest total altitude gained:
  user_id    total_gained_altitude
---------  -----------------------
      128                  650887
      153                  554969
      004                  332036
      041                  240758
      003                  233664
      085                  217642
      163                  205264
      062                  181692
      144                  179457
      030                  175680
      039                  146704
      084                  131161
      000                  121505
      002                  115063
      167                  112973
```

Figure 13: Task 9 result

Task 10

I assume that ”in one day”, that refers to within a single date, not within 24 hours of an activity has started. Users are filtered out if they have has_label set to 1, and trackpoints are selected if the activity transportation mode is not null. Distances are calculated if the current trackpoint and previous have the same date, transportation mode and activity id. Activity id is important if there are for example two taxi rides in a day for a user, with different starting locations, so that the distance between them are not added. The distances accumulate based on user, date and transportation mode.

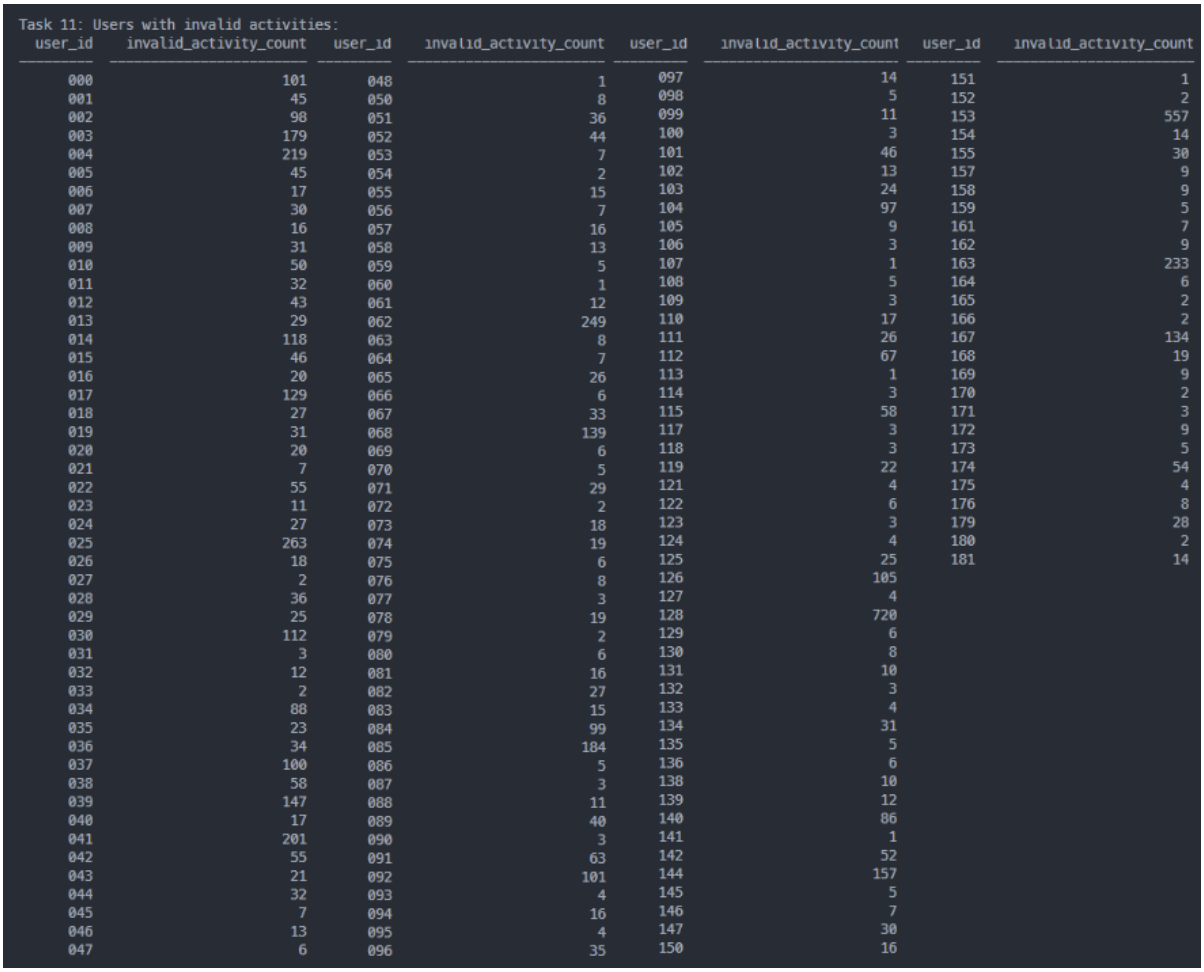
```
Task 10: Users with the longest distance traveled per transportation mode:
Transportation mode bus: user: 128, distance: 207.41 km
Transportation mode taxi: user: 128, distance: 40.22 km
Transportation mode walk: user: 108, distance: 22.81 km
Transportation mode bike: user: 128, distance: 63.11 km
Transportation mode car: user: 128, distance: 398.17 km
Transportation mode run: user: 062, distance: 0.03 km
Transportation mode train: user: 062, distance: 277.26 km
Transportation mode subway: user: 128, distance: 33.94 km
Transportation mode airplane: user: 128, distance: 2527.12 km
Transportation mode boat: user: 128, distance: 65.55 km

----------------------------------------
```

Figure 14: Task 10 result

Task 11

I’ve assumed that an activity is invalid if ANY consecutive trackpoints deviate at least 5 min, not all of them.



Task 11: Users with invalid activities:

user_id	invalid_activity_count	user_id	invalid_activity_count	user_id	invalid_activity_count	user_id	invalid_activity_count
000	101	048	1	097	14	151	1
001	45	050	8	098	5	152	2
002	98	051	36	099	11	153	557
003	179	052	44	100	3	154	14
004	219	053	7	101	46	155	30
005	45	054	2	102	13	157	9
006	17	055	15	103	24	158	9
007	30	056	7	104	97	159	5
008	16	057	16	105	9	161	7
009	31	058	13	106	3	162	9
010	50	059	5	107	1	163	233
011	32	060	1	108	5	164	6
012	43	061	12	109	3	165	2
013	29	062	249	110	17	166	2
014	118	063	8	111	26	167	134
015	46	064	7	112	67	168	19
016	20	065	26	113	1	169	9
017	129	066	6	114	3	170	2
018	27	067	33	115	58	171	3
019	31	068	139	117	3	172	9
020	20	069	6	118	3	173	5
021	7	070	5	119	22	174	54
022	55	071	29	121	4	175	4
023	11	072	2	122	6	176	8
024	27	073	18	123	3	179	28
025	263	074	19	124	4	180	2
026	18	075	6	125	25	181	14
027	2	076	8	126	105		
028	36	077	3	127	4		
029	25	078	19	128	720		
030	112	079	2	129	6		
031	3	080	6	130	8		
032	12	081	16	131	10		
033	2	082	27	132	3		
034	88	083	15	133	4		
035	23	084	99	134	31		
036	34	085	184	135	5		
037	100	086	5	136	6		
038	58	087	3	138	10		
039	147	088	11	139	12		
040	17	089	40	140	86		
041	201	090	3	141	1		
042	55	091	63	142	52		
043	21	092	101	144	157		
044	32	093	4	145	5		
045	7	094	16	146	7		
046	13	095	4	147	30		
047	6	096	35	150	16		

Figure 15: Task 11 result

Task 12

To get one transportation mode, I've used row numbers for ranking the transportation modes, so if there's a tie, one of them will be assigned row number 2 instead of 1. I guess I could have concatenated them together alternatively.

```
Task 12: Most used transportation mode per user:
  user_id  transportation_mode
---------  ----------------------
      010  taxi
      020  bike
      021  walk
      052  bus
      056  bike
      058  taxi
      060  walk
      062  walk
      064  bike
      065  bike
      067  walk
      069  bike
      073  walk
      075  walk
      076  car
      078  walk
      080  taxi
      081  bike
      082  walk
      084  walk
      085  walk
      086  car
      087  walk
      089  car
      091  bus
      092  bus
      097  bike
      098  taxi
      101  car
      102  bike
      107  walk
      108  walk
      111  taxi
      112  walk
      115  car
      117  walk
      125  bike
      126  bike
      128  car
      136  walk
      138  bike
      139  bike
      144  walk
      153  walk
      161  walk
      163  bike
      167  bike
      175  bus
```

Figure 16: Task 12 result

Discussion

I believe I did most tasks as explained, I did not use SQL variables however. In many of the simpler SQL tasks, a pure SQL query was sufficient, but tasks necessitated the use of manual calculations in Python (especially haversine distance measures).

Task 8 was by far the most difficult, and I could only get a result within a reasonable runtime when doing a coarse filtering with overlapping bounding boxes and durations before doing trackpoint comparisons. Task 9 and 10 were also quite difficult. Task 9 could probably been more easily done if I calculated the altitude gains in Python instead of forming a SQL query that did everything.

I feel like I've learned a lot from this assignment, as I beforehand have dealt with by and large pretty simple SQL queries, and to solve the tasks, I had to brush up on a lot of basic SQL as well as attaining more intermediate knowledge. I've also gained some experience in how to insert large amounts of data into SQL in turn depending on foreign key constraints (User→Activity→TrackPoint), as well as some organizing and cleaning of data.

Feedback

The assignment was fun and educational, but some of the questions may be a bit ambiguous, e.g., what is defined as an activity logged twice, same user and start and end date, or the same trackpoints? Most questions were clarified on Piazza, but if a variant of this assignment would be assigned again, I would consider being as explicit as possible.