GTFS Data Analysis

ENGG1001 Assignment 2
Semester 1, 2021
Due date: 23rd April, 16:00

1 Getting started

To start, download *gtfs.zip* from Blackboard and extract the contents. The *gtfs.zip* folder contains all the necessary files to start this assignment. You will be required to implement your assignment in a file called *a2.py* This is the only file you should upload.

Note: The functionality of your assignment will be marked by automated tests. This means that the output of each function, as well as the output of your overall program, must be exact to the specifications outlined below in the **Implementation section**.

2 Data description

The General Transit Feed Specification (GTFS) defines a common format for public transportation schedules and associated geographic information. GTFS feeds let public transport agencies publish their transit data and developers write applications that consume that data in an interoperable way.

This assignment will require the following files and the corresponding specific attributes from the GTFS;

- `stops.txt`: Stops where vehicles pick up or drop off riders. Also defines stations and station entrances.
  - `stop_id`: Identifies a stop.
  - `stop_name`: Name of the stop.
  - `stop_lat`: Latitude of the stop location.
  - `stop_lon`: Longitude of the stop location.
  - `parent_station`: It contains the name of the parent station. For example, UQ Lakes has several platforms (from A to E) where each is identified as a specific stop, the `parent_station` enclosing all is 'place_UQLAKE'.
- `stop_times.txt`: Times that a vehicle arrives at and departs from stops for each trip.
  - `trip_id`: Identifies a trip.

  - `arrival_time`: Arrival time at a specific stop for a specific trip on a route. Often, arrival and departure times at a stop will be the same. For times occurring after midnight on the service day, the values are greater than 24:00:00 in HH:MM:SS local time for the day on which the trip schedule begins.
  - `departure_time`: Departure time from a specific stop for a specific trip on a route.
  - `stop_id`: Identifies a stop.

- `trips.txt`: Trips for each route. A trip is a sequence of two or more stops that occur during a specific time period.
  - `service_id`: Uniquely identifies a set of dates when service is available for one or more routes. ID referencing `calendar.service_id`.
  - `trip_id`: Identifies a trip.

- `calendar.txt`: Service dates specified using a weekly schedule with start and end dates.
  - `service_id`: Uniquely identifies a set of dates when service is available for one or more routes.
  - `monday`: Indicates whether the service operates on all Mondays in the date range specified by the `start_date` and `end_date` fields. 1 - Service is available for all Mondays in the date range, 0 - Service is not available for Mondays in the date range.
  - `tuesday` to `sunday`: Functions in the same way as monday except applies to other days from `tuesday` to `sunday`.
  - `start_date`: Start service day for the service interval.
  - `end_date`: End service day for the service interval.

3 Implementation

Within this section, the following function will be developed;

- `arriving_buses(stop_name, start_time, end_time, date, interval_length)`: This function returns a list containing the number of buses arriving at `stop_name` between `start_time` and `end_time` on `date` with aggregation intervals of `interval_length`. Denote t_a the arrival time of a bus at `stop_name`, the first value in the returned list should be the number of buses satisfying the following conditions; `start_time` $\leq t_a <$ `start_time`+ `interval_length`, the second value in the list should be the number of buses where `start_time`+`interval_length` $\leq t_a <$ `start_time`+2*`interval_length`, etc.
  - `stop_name: str`
  - `start_time: str` in the HH:MM format

  - `end_time`: `str` in the HH:MM format
  - `interval_length`: `int` in minutes
  - `date`: `str` date in the format 'YYYYmmdd'. You can assume the provided `date` exists in the available date range in `calendar.txt`. If not, the result should be an empty list.

You must write the following functions as part of your implementation. You are encouraged to add your own additional functions if they are beneficial to your solution.

- `read_data() -> df, df, df, df`: This function reads the csv files and returns four data frames named `trips, stops, stop_times, calendar` in accordance with the file names. Date and time columns in the relevant data sets should be converted to `datetime` objects. Note that `arrival_time` and `departure_time` in `stop_times` can be greater than 24:00:00 (night time within a service day can take values between 24:00:00 and 29:59:59); rows with this type of entries should be removed.

  Example:

  ```
  >>> trips, stops, stop_times, calendar = read_data()
  trips.size, stops.size, stop_times.size, calendar.size
  (694288, 140250, 18963581, 1550)
  ```

- `find_service(calendar, date) -> list`: This functions returns a list containing the `service_ids` operating on the `date`. The returned `service_ids` should satisfy two conditions; (i) `date` must be between `start_date` and `end_date`, (ii) service must operate on the day of the week for the given `date`.

- `create_subsets(stops, stop_times, trips, calendar, stop_name, date) -> df, df`: As a first step, this function (i) finds the `stop_id`(s) corresponding to the `stop_name` in the data frame `stops`, (ii) creates a subset, named `stoptime_subset`, of the data frame `stop_times` consisting of the rows which contain the previously chosen `stop_id`(s), and (iii) creates a subset, named `trip_subset`, of the data frame `trips` consisting of the rows which contain the same `trip_ids` as `stoptime_subset`. At this stage, `stoptime_subset` and `trip_subset` would include all the observations related to `stop_name` without taking into account the operation date.

  Second, this function should identify the `service_id`(s) operating on the `date` through the function `find_service` described above, and create further subsets of `trip_subset` and `stoptime_subset` (with the same names) accounting for the available `service_id`(s) on the `date` and the corresponding `trip_id`(s), respectively. Note that if `stop_name` does not exist in `stops['stop_name']`, `stops['parent_station']` must be searched. *Hint:* `empty` command from Pandas can be useful to determine whether the resulting series or data frame is empty.

  Example:

  ```
  >>> stoptime_subset, trip_subset = create_subsets(stops, stop_times, trips,
  calendar, 'place_UQLAKE', '20201116')
  stoptime_subset.size
  4893
  ```

- `calculate_counts(start_time, end_time, interval, stoptime_subset) -> list`: This functions considers the data frame `stoptime_subset`, and returns a list containing the number of vehicles between `start_time` and `end_time` with aggregation intervals of `interval`. Mathematically speaking, denote t_a the arrival time of a bus at `stop_name`, the first value in the returned list should be the number of buses satisfying the following conditions; `start_time` $\leq t_a <$ `start_time`+ `interval_length`, the second value in the list should be the number of buses where `start_time`+`interval_length` $\leq t_a <$ `start_time`+2*`interval_length`, etc.

  Example:

  ```
  >>> calculate_counts('06:00', '09:00', 15, stoptime_subset)
  [3, 3, 4, 7, 8, 12, 12, 13, 13, 16, 13, 13]
  ```

4 Example Output

```
>>> start_time = '06:00'
end_time = '09:00'
stop_name = 'place_UQLAKE'
day = '20201116'
interval = 30
counts = arriving_buses(stop_name, start_time, end_time, day, interval)
counts
6, 11, 20, 25, 29, 26
```

5 Marking Criteria

5.1 Functionality Assessment

The functionality will be marked out of 12. Your assignment will be put through a series of tests and your functionality mark will be proportional to the number of tests you pass. If, say, there are 25 functionality tests and you pass 20 of them, then your functionality mark will be 20/25 * 12. You will be given the functionality tests before the due date for the assignment so that you can gain a good idea of the correctness of your assignment yourself before submitting. You should, however, make sure that your program meets all the specifications given in the assignment. That will ensure that your code passes all the tests. Note: Functionality tests are automated and so string outputs need to exactly match what is expected.

5.2 Code Style Assessment

The style of your assignment will be assessed by one of the tutors, and you will be marked according to the style rubric provided with the assignment. The style mark will be out of 3.

6 Assignment Submission

You must submit your completed assignment electronically through Blackboard. The only file you submit should be a single Python file called a2.py (use this name – all lower case). This should be uploaded to Blackboard>Assessment>Assignment 2. You may submit your assignment multiple times before the deadline – only the last submission will be marked.

Late submission of the assignment will not be accepted. In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on time, you may submit a request for an extension. See the course profile for details of how to apply for an extension.

Requests for extensions must be made no later than 48 hours prior to the submission deadline. The application and supporting documentation (e.g. medical certificate) must be submitted to the ITEE Coursework Studies office (78-425) or by email to enquiries@itee.uq.edu.au. If submitted electronically, you must retain the original documentation for a minimum period of six months to provide as verification should you be requested to do so.