Motivation and Objective

Data Used

Project Meeting Reports

# [2022-01-19] Project Meeting 00

* Set up Google Folders for the projects

(<https://drive.google.com/drive/folders/1EGvZJIzOUNa9PDYcl7ZE0UJvOkh1ZED7?usp=sharing>)

* Weekly meeting formats
* Progress report (QA)
* Task discussion

Tasks:

* ~~Upload and organize the data you find into the folder:~~

<https://drive.google.com/drive/folders/1l5UIM2H-9j_xTJ2WvoogaUpix8P1qGM3?usp=sharing>

* [GTFS](https://drive.google.com/drive/folders/1CTg1zKGXidH9AhExV9bDFKGYF8qn5g7h?usp=sharing) has the same time period as the ALine data (Oct. 1-8, 2016)
* Road network data (attributes)

(OSM-GIS; [Road centerline](https://gisdata.mn.gov/dataset/us-mn-state-metrogis-trans-road-centerlines-gac)-DOT)

* Create Jupyter project and Read AVL data into panda
* Fields?
* Field types?
* What fields are potentially useful? (QA)
* (optional) create a test dataset with smaller size

**1/22**

**Reading fields on A-Line Messages table**

* Transmitted\_Message\_ID
* Calendar\_ID: bigint, format 1YYYYMMDD, ie 120161001
  + Datatype: bigint
* Message\_Type\_ID:
* Latitude: latitude in long integer, ie (lat\*10,000,000)
  + Datatype: int
  + Importance: gives message/bus location
* Longitude: longitude in long integer, ie (long\*10,000,000)
  + Datatype: int
  + Importance: gives message/bus location
* Adherence: difference between scheduled and actual, minutes
  + negative values are behind schedule
  + Datatype: smallint
  + Importance: indicates delay, paired with odometer
* Odometer: integer \* 100, given in the 100th/mile. Restarts daily
  + Datatype: int
  + Importance: total distance travelled so far in the day
* Validity:
* Source\_Class
* Source\_Host: vehicle transmitter ID
  + Importance: gives the vehicle id

##### **~~Destination\_Class~~**

##### **~~Destination Host~~**

* Route\_Version
* Messages\_Version
* Route \_Offset
* Direction: 1 is Southbound, 4 is Northbound
* TIME\_POINT\_OFFSET
* STOP\_OFFSET
* FLAG32
* EFFECTIVE\_SERVICE
* MDT\_TIMESTAMP
* MESSAGE\_TIMESTAMP
* PASSENGER\_COUNT\_ON: passenger boarding count
* PASSENGER\_COUNT\_OFF: passenger disembark count
* ST\_MDT\_VERSION
* ~~SYSPARAM\_FLAG~~
* MSG\_GROUP
* CAT\_1
* CAT\_2
* CAT\_3
* CAT\_4
* CAT\_5
* CAT\_6
* CAT\_7
* CAT\_8
* CAT\_9
* CAT\_10
* LOWER32
* UPPER32
* CURRENT\_DRIVER
* FREE\_TEXT\_MSG
* LOCAL\_TIMESTAMP
* MDT\_BLOCK\_ID
* SIGNAL\_STRENGTH
* BLOCK\_ABBR
* ROUTE\_ABBR
* PROPERTY\_TAG
* SERVICE\_ABBR: service type (Weekday, Sun, Sat)
* NODE\_ABBR: 4-letter code standing for
* FOM: GPS accuracy, 2 is high
* dGPS
* ValidOdo: Boolean
* ValidAdh: Boolean
* ValidLoc: Boolean
* MESSAGE\_TYPE\_TEXT: ‘Vehicle Location’ or ‘Verbose Pass Count’

# [2022-01-24] Project Meeting 01

* Questions: What does ‘overloaded’ mean? These rows have TPC/APC value filled
* Questions: Fields 17, 18, 42, 46, 49, 55

(create a key value for future “join” back to the original data)

(<https://drive.google.com/drive/folders/1CTg1zKGXidH9AhExV9bDFKGYF8qn5g7h?usp=sharing>) Old Scripts for reference about fields

Tasks

* ~~Two options:~~
* ~~OSM; AVL data - GPS (x, y, t) <check python packages that can match GPS data to OSM networks> <OSM 2016 Oct.> -> GTFS routes~~

* ~~GTFS routes -> shapefile/geometry (network geometries); AVL data-GPS (x, y, t) <linear reference to the routes>~~
* Check Oct 1, Oct 4 (routes; schedules) compare two files
* Discuss if there are any differences between two file sets

No differences; using one GTFS on Oct. 1, 2016

* ~~How to~~ **~~reconstruct the geometries of routes (trips)~~**
* Python packages (GTFS - trips)
  + **gtfs-kit:** [**https://pypi.org/project/gtfs-kit/**](https://pypi.org/project/gtfs-kit/)
  + gtfs\_functions: <https://pypi.org/project/gtfs-functions/>
  + grfsk: <https://mrcagney.github.io/gtfstk_docs/>
  + py\_gtfs: <https://pygtfs.readthedocs.io/en/latest/>
  + Peartree: <https://pypi.org/project/peartree/>
  + partridge: <https://pypi.org/project/partridge/>
  + gtflib: <https://pypi.org/project/gtfslib/>
* Are the GTFS files the same for multiple dates? Compare using script
* Oct1, Oct4 ,Oct5 = same
* Oct5 =/= Oct6
* We have decided to just use Oct 1st for all days

# [2022-01-31] Project Meeting 02

**Task:** create route shapefiles from GTFS data, which will be used as the base for linear referencing.

1. Considering the stability of Python packages in the long run, ArcPy may be a “saver” choice: <https://medium.com/analytics-vidhya/the-hitchhikers-guide-to-gtfs-with-python-e9790090952a>

* Check the ArcPy functions

GenerateShapesFeaturesFromGTFS\_conversion

??? to be direct links connecting two points? Or the actual routes along roads?

If the above Option 1 does not work, looking at the resources regarding importing GTFS into the OSM and then export the shapes as Option 2

* ~~Check how to integrate GTFS with OSM~~

<https://wiki.openstreetmap.org/wiki/General_Transit_Feed_Specification>

**Check ArcPy functions**

**Way 1:** GTFS w/o shapes.txt >>  GenerateShapesFeaturesFromGTFS >>FeaturesToGTFSShapes >> new shapes.txt file

**GenerateShapesFeaturesFromGTFS\_conversion** - if no existing shapes.txt file in GTFS dataset, and you want to estimate shapes based on the stop, route, and schedule information from an existing GTFS dataset, creating new shapes.txt file with FeaturesToGTFSShapes tool

* This tool also requires spatial data for a road network

You seem to be able to generate straight lines OR along a network with this tool (although without the network there are some straight and bent). Also creates transit stop layout

**FeaturesToGTFSShapes\_conversion** - Creates a shapes.txt file for a GTFS public transit dataset based on route line representations created by the Generate Shapes Features From GTFS tool. This also creates a new stop\_times.txt file

**Way 2:** GTFS w/ existing shapes.txt >> GTFSShapesToFeatures >>feature class

**GTFSShapesToFeatures\_conversion** - if GTFS dataset contains existing shapes.txt and you want to convert shapes to feature class

This seems to create a network of existing shapes

HOT export tool

# [2022-02-07] Project Meeting 03

Task: Use Python scripts to generate transit route and stop shapefiles

* ~~Create supporting queries for future use, which allows select by route\_short\_name; by date and time ranges;  {shape\_id <-> trip\_id}~~
* ~~Use ArcPy function to generate transit route shapefiles (real shapes of routes instead of straight line); document what the network used as inputs to generate routes.~~
* ~~Use log/lat to points or ArcPy function to generate stop location shapefiles~~
* Consider using one or a few shape\_id for testing the generation of routes
* Better to use geopandas to visualize the shape and use OSM as base layer as reference
* **Make sure the IDs in the generated shapefiles are one-to-one corresponding to the original IDs in the .csv files.**

Next step: Linear reference the AVL data (x, y, t, tid…) to the route

* Break down AVL data into trips

Continue some tasks from previous week:

* ~~Query~~
  + ~~(step 0) select all trips and shapes for a given route (A-Line) <south; north>~~
  + ~~(step 1) select all stops and stops\_times~~
* ~~Break down AVL data into trips~~
  + ~~(step 2) use the south\_bound (4) and north\_bound (1) to generate trips from AVL data; add new sequence number (ID) for each trip~~

* (step 3) generate a new table with the route ID, sequence number, first GPS time and location, last GPS time and location;
* AVL (OID, datetime, log, lat, direction, trip\_SEQ, device\_id?)

AVL (OID, datetime, log, lat, direction, date\_seq, trip\_seq, device\_id?)

Summary Table of AVL

(date\_seq, trip\_seq, device\_id?, direction, first\_datetime. first\_log, first\_lat, last\_datetime, last\_log, last\_lat)

* GTFS (date, route, trip, <vehicle?>); {stop\_time} for each trip; shapes

Summary Table of GTFS

(date,  trip\_seq,vehicle\_id?, direction, first\_stop\_time, first\_st\_log, first\_st\_lat, last\_stop\_time, last\_st\_log, last\_st\_lat)

* (step 4)  basic ideas: use start T, end T, direction (north/south) of a trip (GTFS); first GPS time, last GPS time, and direction (AVL data)=> get the shape of the trip for a given set of GPS data (route ID, sequence number #)

(optional solution #1) join based on two sets of keys (blue and yellow)

* Final output: joint table (trip ID for GTFS; route\_ID + sequence for AVL)
* Calculate the distance between the first/last GPS location and the first/last stop for the matched trip (same bound, similar duration)
* Calculate the time difference between the first/last GPS location and the first/last stop\_time along the matched trip

(optional solution #2) join based on the (log, lat, time, direction, vehicle?) of AVL and GTFS summary

Final result is a linked/joint table

<(date\_seq, trip\_seq, device\_id), (date, trip\_seq, vehicle\_id?)>

Join process brainstorming: Queries

|  |  |  |
| --- | --- | --- |
|  | *Only want specific routes* | *Only want specific times/days* |
| 1 | Query **routes.txt** by route\_short\_name to give route ID | Query calendar\_dates.txt by date to give service ID |
| 2 | Join **trips.txt** on **route.txt(tgt)** by route\_ID to get shape\_ID | Join **trips.txt** on **calendar\_dates.txt** by service\_ID to give shape\_ID |
| 3 | Join **shapes.txt** on **trips\_query.txt** by shape\_ID to get revised shapes | Join **shapes.txt** on **trips\_query.txt** by shape\_ID to get revised shapes |
| 4 | Put **shapes.txt** into **GTFSShapesToFeatures** | Put **shapes.txt** into **GTFSShapesToFeatures** |

Join trips.new on calendar\_new.txt

Remove Join

Join trips\_new.txt on calendar.txt

|  |  |  |
| --- | --- | --- |
| From route | From date |  |
|  | * Pare calendar\_dates.txt down by date |  |
| * Pare routes.txt down by route ID * Select and save as routes\_new.txt * Join trips.txt on routes\_new by routeID * **Select as trips\_new.txt** * Remove Join | * Join calendar.txt on calendar\_dates.txt * **Select as calendar\_new.txt** * Remove Join * Join trips.txt on calendar\_new by date\_ID * **Select and save as trips\_new.txt** |  |
|  | * Join stop\_times.txt on trips\_new * Select and save as stop\_times\_new.txt * Remove Join |  |
|  |  |  |
| * Join shapes.txt on trips\_new * Select and save as shapes\_new.txt | * Join shapes.txt on trips\_new * Select and save as shapes\_new.txt |  |

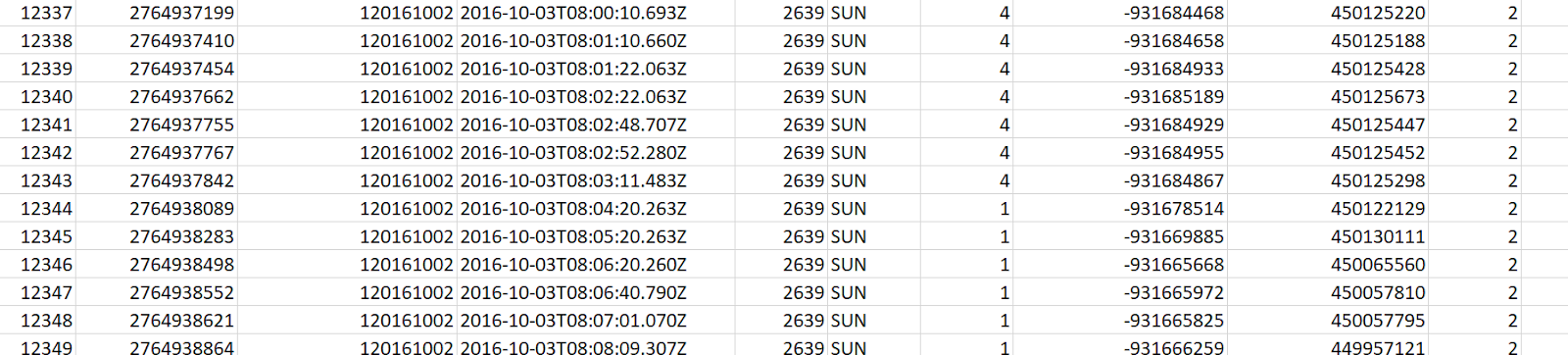
# [2022-02-14] Project Meeting 04

Continue some tasks from previous week:

* ~~Query~~
  + ~~(step 0) select all trips and shapes for a given route (A-Line) <south; north>~~
  + ~~(step 1) select all stops and stops\_times “ ”~~
* ~~Break down AVL data into trips~~
  + ~~(step 2) use the south\_bound (4) and north\_bound (1) to generate trips from AVL data; add new sequence number (ID) for each trip~~

**Sequence each or NB and SB? Or just one?**

* Cut down .txt files based on query  (GTFS Joining.py)
* Put new shapes, stops, stop times files, etc in new folder (Folder makes up GTFS dataset)
* Generate feature classes of shapes and stops (GTFS Shapes to Features)
  + BUT, will not work if format of new files isn’t the same..
* Geopandas will not work in the arcpro python environment (and arcpy won’t in conda)



**Why no match?**

# [2022-02-21] Project Meeting 05

* **Generate New Tables** with sequence number, first GPS time and location, last GPS time and location;
  + **~~AVL~~**
    - ~~Create an intermediate AVL table (OID, datetime, log, lat, direction, date\_seq, trip\_seq, device\_id?)~~
* ~~Trip sequence starts over each day, which might end at midnight, might not; D1-T1, D1-T2, D2-T1, etc.~~
* ~~Create summary Table of AVL:~~
  + ~~(date\_seq, trip\_seq, device\_id?,direction, first\_datetime, first\_long, first\_lat, last\_datetime, last\_long, last\_lat)~~
* **GTFS**
* Create an intermediate GTFS table (date, route, trip, <vehicle?>); {stop\_time} for each trip; shapes
  + Probably will need some joins here
* Create summary Table of GTFS
  + (date, trip\_seq, vehicle\_id?, direction, first\_stop\_time, first\_st\_log, first\_st\_lat, last\_stop\_time, last\_st\_log, last\_st\_lat)
* **Matching GTFS to AVL:** use start T, end T, direction (north/south) of a trip (GTFS); first GPS time, last GPS time, and direction (AVL data)=> get the shape of the trip for a given set of GPS data (route ID, sequence number #)

**(optional solution #1)** join based on two sets of keys (blue and orange)

* Calculate the distance between the first/last GPS location and the first/last stop for the matched trip (same bound, similar duration)
* Calculate the time difference between the first/last GPS location and the first/last stop\_time along the matched trip
* Final output: joint table (trip ID for GTFS; route\_ID + sequence for AVL)

**(optional solution #2)** join based on the (log, lat, time, direction, vehicle?) of AVL and GTFS summary

* Final result is a linked/joint table by these keys (blue and orange)

<(date\_seq, trip\_seq, device\_id), (date, trip\_seq, vehicle\_id?)>

126,000 messages

* \*message timestamp: time in UTC (sometimes doesn't align with calendar)
* \*local timestamp: time in local time (many of the same time?
* Calendar ID indicates the schedule day, not the time day
* There are 664 rows with no direction value - these seem to be when the bus is parked at a garage (its odometer isn't going up)
* The seq codes for the device ID should probably be the actual device ID # and not a sequential value so the same number is always the same bus
* Better to use day-bus-trip-time or bus-day-trip-time?
  + SEQ1 = Day#
  + SEQ2 = Day #'s Bus#
  + SEQ3 = Bus#'s Trip#
  + SEQ4 = Trip#'s Message#
* Does the GTFS table need trip seq or tripID?
* Calendar.txt & trips.txt table > do I only want the trips with service ID’s running through Oct 1-8?

**Step1:**

if: row's day# <> previous row's day#,

bus# resets to 1 and tripcount# resets to 1

else (day# is same):

if: row bus# <> previous row's bus#,

 row's tripcount# reset to one

else (bus# is same):

if: row dir# <> previous row's dir# ,

row's tripcount# = previous row's tripcount# + 1

else(no change in dir#):

If: tripcount# <> previous row tripcount#,

row’s ping resets to 1

Else (trip# is same):

row’s ping = previous row’s ping+1

What about the pings with NO direction? NaN != NaN, so it is labeling each NaN as its own trip, creating lots of “1st” pings in a row: This messes the trip count up A LOT

# [2022-02-28] Project Meeting 06

**Cole’s Questions for 2/28 (I didn’t get far this week-stuck)**

* Use message timestamp: time in UTC (sometimes doesn't align with calendar) vs. local timestamp: time in local time (many of the same time)
* 664 rows with no direction value - these seem to be when the bus is parked at a garage (its odometer isn't going up): keep them in the summary table, but not for future linear reference. Because these are very likely when vehicle start to get signals (GPS starts to locate vehicle, no direction)

* The seq codes for the device ID should probably be the actual device ID # and not a sequential value so the same number is always the same bus?

Not necessarily. And for our project, the sequence in a service day will be what we need to know. Buses could change order/seq of services, but it does not affect our analysis later.

* Is it better to use day-bus-trip-time or bus-day-trip-time? **Day-bus-trip-time**
* **~~AVL summary table~~**
  + ~~Summarize by (Day#, Bus#) (try the groupby function)~~
* For each service day, what are the service periods for each bus
* Interested in knowing (a) total range, start and end time; (b) time gaps??
* We can use this as a reference for future discussion (a) order of service for each bus, (b) across different days?; (c) help with next step of joining
* Summarize by (Day#)
* How many buses serving that day <required> service\_id
* Set of bus# are the same across different days
* **~~GTFS summary table~~**
* GTFS (date, route, trip, <vehicle?>); {stop\_time} for each trip; shapes
* Final Summary Table of GTFS

(date,  trip\_seq,vehicle\_id?, direction, first\_stop\_time, first\_st\_log, first\_st\_lat, last\_stop\_time, last\_st\_log, last\_st\_lat)

* Does the GTFS table need trip seq or tripID?

Keep in mind that GTFS will provide the **SHAPE for each trip; and schedule time at each stop along the trip.**

**Day-bus-trip-time**

* **AVL summary table**
  + ~~Summarize by (Day#, Bus#) (try the groupby function)~~
* For each service data, what are the service periods for each bus
* Interested in knowing (a) total range, start and end time; (b) time gaps??
* We can use this as a reference for future discussion (a) order of service for each bus, (b) across different days?; (c) help with next step of joining
* Summarize by (Day#)
* How many buses serving that day <required> service\_id
* Set of bus# are the same across different days
* **GTFS summary table**
* GTFS (date, route, trip, <vehicle?>); {stop\_time} for each trip; shapes

Summary Table of GTFS

(date,  trip\_seq,vehicle\_id?, direction, first\_stop\_time, first\_st\_log, first\_st\_lat, last\_stop\_time, last\_st\_log, last\_st\_lat)

* Does the GTFS table need trip seq or tripID?

Keep in mind that GTFS will provide the **SHAPE for each trip; and schedule time at each stop along the trip.**

And we need to map each trip to the corresponding AVL trip.

* Calendar.txt & trips.txt table > do I only want the trips IDs with service a ID running through Oct 1-8?
* **~~AVL summary table~~**
  + Summarize by (Day#, Bus#) (try the groupby function)
* For each service day, what are the service periods for each bus
  + Service period meaning the whole day or by trip?
* Interested in knowing (a) total range, start and end time; (b) time gaps??
  + Total range?
* We can use this as a reference for future discussion (a) order of service for each bus, (b) across different days?; (c) help with next step of joining

**DayBusSortAVL.csv:** calendar\_id, trans message id, message\_time(UTC), source\_host, odometer, route\_abbr, direction, longitude, latitude, day\_id (1-8), source\_id (1-?)....then ordered by time

* Summarize by (Day#) (is just the precursor to above?
* How many buses serving that day <required> service\_id
* Set of bus# are the same across different days
* **~~GTFS summary table~~**
* GTFS (date, route, trip, <vehicle?>); {stop\_time} for each trip; shapes**:**
  + **all\_trip\_shapes.csv:** route\_id, trip\_id, service\_id, trip\_headsign, shape\_id, shape\_pt\_lat, shape\_pt\_lon, shape\_pt\_sequence, direction
  + **all\_trip\_stop\_times.csv:** trip\_id, service\_id, trip\_headsign, shape\_id, arrival\_time, departure\_time, stop\_sequence, stop\_id, stop\_name, stop\_desc, stop\_lat, stop\_lon, direction\_id
  + **sumtableGTFS:** trip\_id, service\_id, trip\_headsign, arrival\_time, departure\_time, stop\_seq (1 and 20 only), stop\_id, stop\_name, stop\_desc, stop\_lat, stop\_lon
* **finalresultGTFS:** trip\_id, service\_id, trip\_headsign,

start[arrival\_time, departure\_time, stop\_seq, stop\_id, stop\_name, stop\_desc, stop\_lat, stop\_lon]

stop[arrival\_time, departure\_time, stop\_seq, stop\_id, stop\_name, stop\_desc, stop\_lat, stop\_lon]

* Summary Table of GTFS

(date,  trip\_seq,vehicle\_id?, direction, first\_stop\_time, first\_st\_log, first\_st\_lat, last\_stop\_time, last\_st\_log, last\_st\_lat)

**\***GTFS will provide the **SHAPE for each trip; and schedule time at each stop along the trip.** And we need to map each trip to the corresponding AVL trip.

**Days: will each weekday get the whole set of trips for the weekday service\_id?**

# [2022-03-07] Project Meeting 07

**Days:** will each AVL weekday calendar\_id ( ie, 10/3, 10/4) **get the same whole set** of GTFS trips with the single weekday service\_id code?

GTFS tables created

**all\_trip\_shapes.csv: route\_id, trip\_id,** service\_id, trip\_headsign, shape\_id, shape\_pt\_lat, shape\_pt\_lon, shape\_pt\_sequence, direction,

a**ll\_trip\_stop\_times.csv:** trip\_id, service\_id, trip\_headsign, shape\_id, arrival\_time, departure\_time, stop\_sequence, stop\_id, stop\_name, stop\_desc, stop\_lat, stop\_lon, direction\_id

**sumtableGTFS: trip\_id,** service\_id, trip\_headsign,

stop\_seq (1), stop\_lat, stop\_lon; arrival\_time, departure\_time, stop\_id, stop\_name, stop\_desc,

stop\_seq (-1), stop\_lat, stop\_lon; arrival\_time, departure\_time, stop\_id, stop\_name, stop\_desc,

**finalresultGTFS: trip\_id, service\_id, trip\_headsign,**

start[arrival\_time, departure\_time, stop\_seq, stop\_id, stop\_name, stop\_desc, stop\_lat, stop\_lon]

stop[arrival\_time, departure\_time, stop\_seq, stop\_id, stop\_name, stop\_desc, stop\_lat, stop\_lon]

**AVL tables created**

**DayBusSortAVL.csv:** calendar\_id, trans message id, message\_time(UTC), source\_host, odometer, route\_abbr, direction, longitude, latitude, day\_id (1-8), source\_id (1-?)....then ordered by time

**Summarize by** (Day#): **(is just the precursor to above?)**

* How many buses serving that day <required> service\_id
* Set of bus# are the same across different days
* ~~AVL data: for each bus/device, sort by messenger time, break it into trop, and create a new tripID for next steps~~

df[‘TRIPID’] = df[‘SOURCE\_HOST’]

Sortdf (sortby SOURCE\_HOST, messenger\_time)

curHost = none

curDir = none

curSeq = 0 # sequence of trips for that vehile

For row, index in sortdf.iterrows():

     rowDir = row[‘DIRECTION’],

     rowHost = row[‘Source\_HOST’]

     if curHost == none:

          if rowDir:

           curHost = rowHost

           curDir = rowDir

              curSeq = 0

              Sortdf.at[index, ‘TripID’] = str(curHost) + ‘\_’ + str(curSeq)

     elif curHost != rowHost:

          If rowDir:

              curHost = rowHost

              curDir = rowDir

              curSeq = 0

     else:

          If curDir != rowDir:

curSeq + 1

….

* ~~Summarize AVL trip information~~

New Numpy.array

#summary table

              for group in (Groupby\_TripID):

    (sort by messenger time)

                    (first row) First messenger time; first long lat

                    (last row) First

                    (route\_abb) ‘unique’  # ‘good’

                    (direction) ‘unique’    # ‘’

                    (total number of GPS data)

append(‘TripID, **start\_time**, start\_log, start\_lat, end\_time, end\_log, end\_lat,

route\_abb, direction’, **‘startWSS’**, ‘endWSS’,

‘cntPts’,  ‘Label’)

**GTFS**

* **sumtableGTFS: trip\_id,** service\_id, trip\_headsign,

stop\_seq (first), stop\_lat, stop\_lon; arrival\_time, departure\_time, stop\_id, stop\_name, stop\_desc,

stop\_seq (last), stop\_lat, stop\_lon; arrival\_time, departure\_time, stop\_id, stop\_name, stop\_desc,

**AVL**

* For each bus/device, sort by messenger time, break it into trop, and create a new tripID for next steps

df[‘TRIPID’] = df[‘SOURCE\_HOST’]

Sortdf (sortby SOURCE\_HOST, messenger\_time)

curHost = none

curDir = none

curSeq = 0 # sequence of trips for that vehicle

For row, index in sortdf.iterrows():

     rowDir = row[‘DIRECTION’],

     rowHost = row[‘Source\_HOST’]

     if curHost == none:

          if rowDir:

           curHost = rowHost

           curDir = rowDir

              curSeq = 0

              Sortdf.at[index, ‘TripID’] = str(curHost) + ‘\_’ + str(curSeq)

     elif curHost != rowHost:

          If rowDir:

              curHost = rowHost

              curDir = rowDir

              curSeq = 0

     else:

          If curDir != rowDir:

curSeq + 1

….

* Summarize AVL trip information

New Numpy.array

#summary table

              for group in (Groupby\_TripID):

    (sort by messenger time)

                    (first row) First messenger time; first long lat

                    (last row) First

                    (route\_abb) ‘unique’  # ‘good’

                    (direction) ‘unique’    # ‘’

                    (total number of GPS data)

append(‘TripID, **start\_time**, start\_log, start\_lat, end\_time, end\_log, end\_lat,

route\_abb, direction’, **‘startWSS’**, ‘endWSS’,

‘countPts’,  ‘Label’)

WSS = weekday, saturday, sunday

# [2022-03-14] Project Meeting 08

* ~~(AVL\_Summary) Double-check the scripts and data~~

(count of AVL records 150? 200+? / why the last few records had inconsistent times? Whether the “service date” make sense)

* ~~Convert UTC datetime to local times for future use~~
* ~~Double check the date~~
* via the total number of records
* ~~Create DOW (weekday/saturday/sunday)~~
* Use this field to join to <service ID>
* ~~(GTFS\_Summary) Encode results~~
* ~~Serice\_id\_start:~~
* (Date Range, via join calendar table) (weekday/saturday/sunday)
* ~~Trip\_headsign:~~
* from “north/south” to “1/4” <consider all directions in the tech document>
* **stop\_longitude & stop\_latitude**
* **check scripts for NaN values**

**sumtableGTFS: trip\_id,** service\_id, trip\_headsign,

stop\_seq (1), stop\_lat, stop\_lon; arrival\_time, departure\_time, stop\_id, stop\_name, stop\_desc,

stop\_seq (-1), stop\_lat, stop\_lon; arrival\_time, departure\_time, stop\_id, stop\_name, stop\_desc,

**SumtabelAVL**: ‘**TripID,** start\_time, start\_log, start\_lat, end\_time, end\_log, end\_lat,

route\_abb, direction’, ‘startWSS’, ‘endWSS’, ‘cntPts’,  ‘Label’)

* Summarize by Service Date
* SumtableGTFS:

- arrival\_time == departure\_time?????

Based on departure\_time

For each [**service id, direction**], get

(first departure\_time, last departure\_time, from the first stop)

(first arrival\_time, last\_arrival\_time, from the last stop)

Total number of trips per serviceID, direction

* Make sure to check the original data so that the last record of service is 26:xx:xx not in the middle of the day (duplicated record???, reorder by departure time??)

* For AVL data

Because buses running over midnight for a service ID

Based on **calendar\_date**, get the corresponding **service id** based on *wss* and the *date range*

For each [calendar\_date, **direction**, **service\_id**], left-join the [**service\_id, direction**], get difference between

First departure\_time diff (AVL-GTFS), last departure\_time diff (AVL-GTFS)

First arrival\_time diff(AVL-GTFS),

**Total number of trips per Day**

**(good news)** they match: sequence to link GTFS tripID to AVL tripSeq

Join the previous two based on Weekday/Sun/Sat Direction merge(AVL data summary, GTFS summary, left)

* **GTFS summary table**
* Check if the original GTFS data has the start and end stop information for north bound? If not, set start as the end of south bound and set the end as the start of the south boun
* **AVL data**
* Sorted by “service date”

(+ weekend/sat/sun) Don’t use 12am as the breakout time point, instead using the actual service date

* Then, sorted by “direction” and “trip start time”

sorted\_values([‘service\_date’, ‘direction’, ‘trip start time’]) so that it matches the GTFS results

(See parts that highlighted in blue above for next steps)

# [2022-03-21] Project Meeting 09

* ~~(AVL\_Summary) Double-check the scripts and data~~

(count of AVL records 150? 200+? / why the last few records had inconsistent times? Whether the “service date” make sense)

* ~~Convert UTC datetime to local times for future use~~
* ~~Double check the date~~
* via the total number of records
* ~~Create DOW (weekday/saturday/sunday)~~
* Use this field to join to <service ID>
* ~~(GTFS\_Summary) Encode results~~
* ~~Service\_id\_start:~~
* (Date Range, via join calendar table) (weekday/saturday/sunday)
* ~~Trip\_headsign:~~
* from “north/south” to “1/4” <consider all directions in the tech document>
* ~~stop\_longitude & stop\_latitude~~
* check scripts for NaN values

**sumtableGTFS: trip\_id,** service\_id, trip\_headsign,

stop\_seq (1), stop\_lat, stop\_lon; arrival\_time, departure\_time, stop\_id, stop\_name, stop\_desc,

stop\_seq (-1), stop\_lat, stop\_lon; arrival\_time, departure\_time, stop\_id, stop\_name, stop\_desc,

**SumtabelAVL**: ‘**TripID,** start\_time, start\_log, start\_lat, end\_time, end\_log, end\_lat,

route\_abb, direction’, ‘startWSS’, ‘endWSS’, ‘cntPts’,  ‘Label’)

**PT2**

* Summarize by Service Date
* SumtableGTFS:

- arrival\_time == departure\_time?????

Based on departure\_time

For each [**service id, direction**], get

(first departure\_time, last departure\_time, from the first stop)

(first arrival\_time, last\_arrival\_time, from the last stop)

Total number of trips per serviceID, direction

* Make sure to check the original data so that the last record of service is 26:xx:xx not in the middle of the day (duplicated record???, reorder by departure time??)
* For AVL data

Because buses running over midnight for a service ID

Based on **calendar\_date**, get the corresponding **service id** based on *wss* and the *date range*

For each [calendar\_date, **direction**, **service\_id**], left-join the [**service\_id, direction**], get difference between

First departure\_time diff (AVL-GTFS), last departure\_time diff (AVL-GTFS)

First arrival\_time diff(AVL-GTFS),

**Total number of trips per Day**

**(good news)** they match: sequence to link GTFS tripID to AVL tripSeq

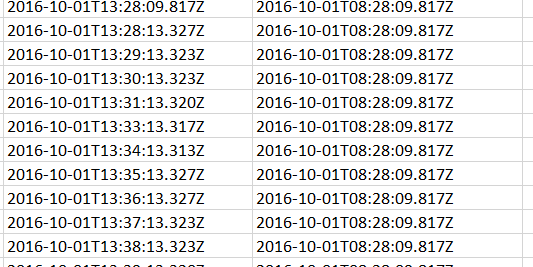
# [2022-03-28] Project Meeting 10

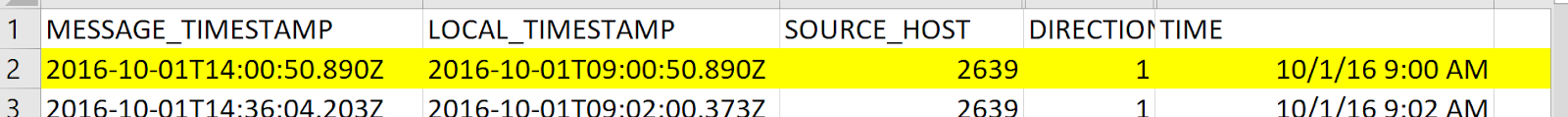
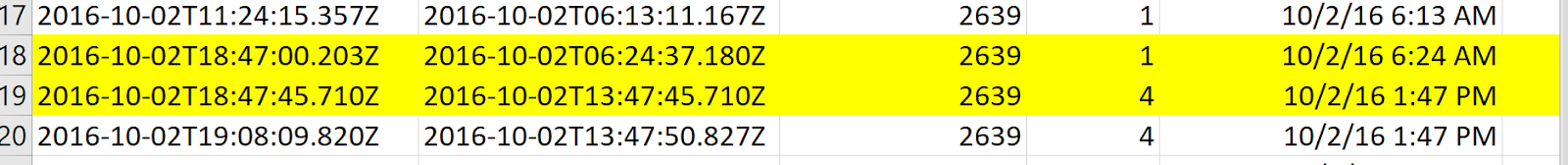
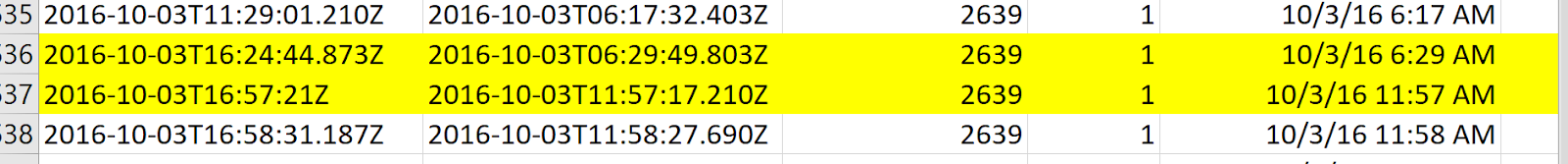
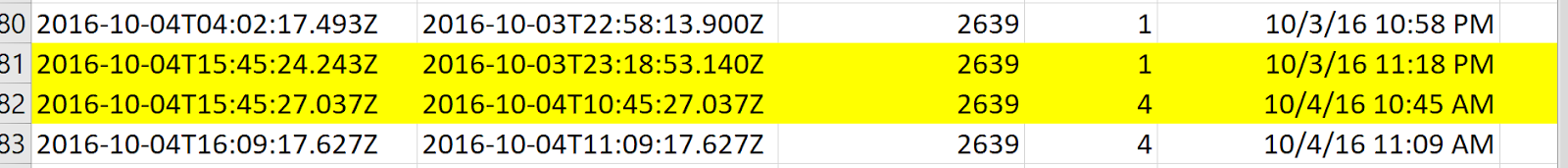
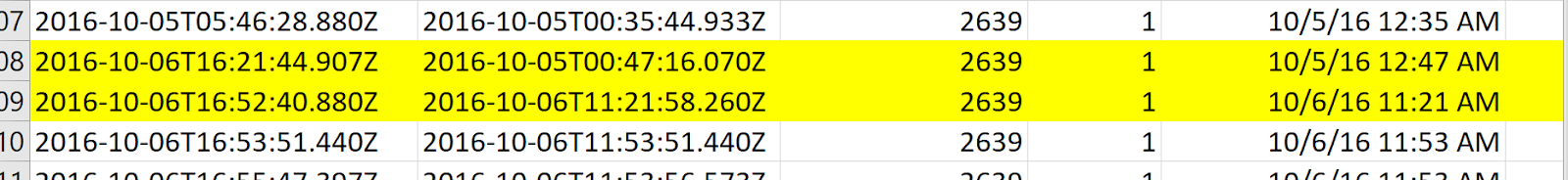
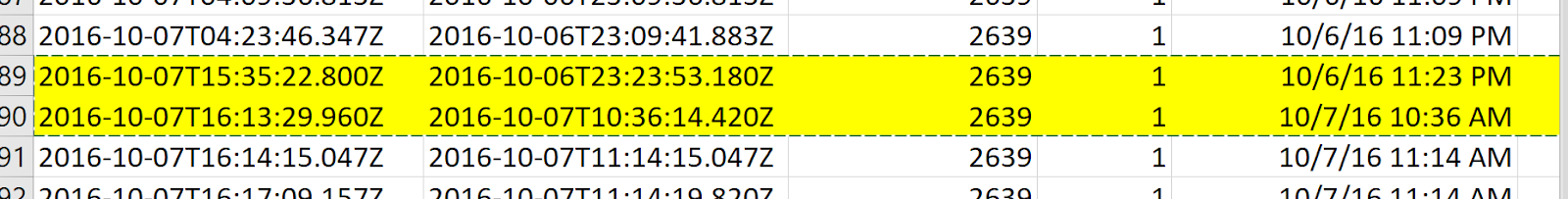
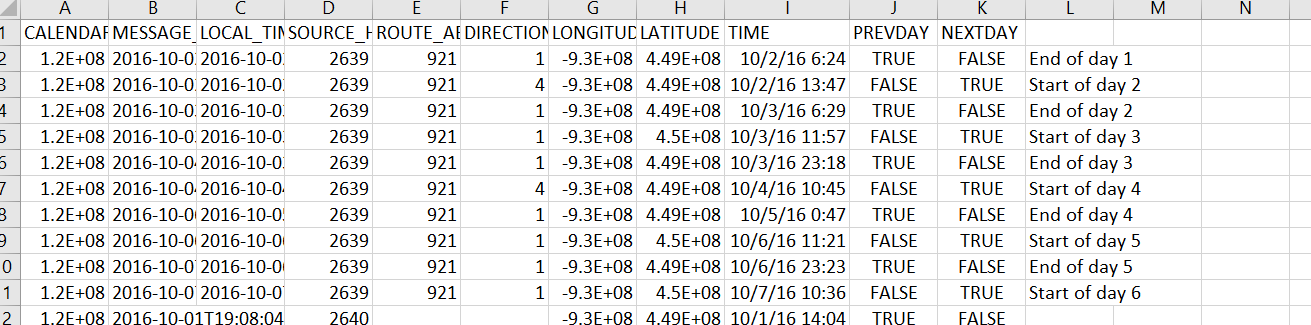
\*there are two saturdays in the dataset

* ~~Summarize by Service Date~~
* ~~SumtableGTFS:~~
  + ~~Based on departure\_time~~
  + ~~For each [~~**~~service id~~** ~~(as in like the BUS-WK-01),~~ **~~direction~~**~~], get:~~
    - ~~(first departure\_time, last departure\_time, from the first stop)~~
    - ~~(first arrival\_time, last\_arrival\_time, from the last stop)~~
    - ~~Total number of trips per serviceID, direction~~
* ~~Make sure to check the original data so that the last record of service is 26:xx:xx not in the middle of the day (duplicated record???, reorder by departure time??)~~
* ~~Check if the original GTFS data has the start and end stop information for north bound? If not, set start as the end of south bound and set the end as the start of the south bound~~
* For AVL data
  + ~~Because buses running over midnight for a service ID~~
* ~~Based on~~ **~~calendar\_date~~**~~, (ie, service date, not literal) get the corresponding~~ **~~service id~~** ~~(ie, the BUS-WK-01 code from GTFS)~~
* ~~Then sort by direction, then trip start time~~
* **~~Total number of trips per Day~~**
* ~~sorted\_values([‘service\_date’, ‘direction’, ‘trip start time’]) so that it matches the GTFS results~~
* I think the message counts are correct. They add to the message total?
* For each [calendar\_date/service date, **direction**, **service\_id**], left-join the [**service\_id, direction**], get difference between
  + Service id is just the BUS-WK code
* First departure\_time diff (AVL-GTFS), last departure\_time diff (AVL-GTFS)
* First arrival\_time diff(AVL-GTFS),
* **(good news)** they match: sequence to link GTFS tripID to AVL tripSeq
* Join the previous two based on Weekday/Sun/Sat Direction merge(AVL data summary, GTFS summary, left)

# [2022-04-04] Project Meeting 11

groupby: transmitterID sortby local timestamp

* Local timestamp is not as specific as message timestamp; several messages in a row are given the same time. (See Below)
* But say we remove any duplicates so there is just one of each local timestamp time for simplicity
* Here are the breaks for Bus #2639:

* AVL: Grouped by:
  + source\_host
  + then time
  + #still need to add ‘service\_type’ marker
  + then takes first and last time for each service day (for that one bus)
* GTFS: Sorted by:
  + service day
  + then direction
  + assign trip numbers (+1 when direction change, reset at day change)
  + then start/stop each trip
* Then get all AVL messages sorted (service\_type,  direction, trip start time)
  + something like give value until >3 hr time gap, then add one?

# [2022-04-18] Project Meeting 12

* **Use Python to realize the process**
* Create route\_shape using GTFS toolset (trip\_shape), output will be the (trip\_shapefile)
* Use the projected coordinate system for inputs (UTM 15N or StatePlane: meters)
* Input of the function (GPS\_series, trip\_shapefile)
* Output (LRDis\_along\_routes; log, lat\_along\_routes; timestamps; sequence)
* Further deal with cases when LRDis is not ascending

# [2022-04-25] Project Meeting 13

Take 1 trip and line up to GTFS shape

Took S and N route from GTFS shapes file, took 1 S and 1 N trip worth of AVL messages. Divide up into N and S. Create N and S routes layer. Then use located features along route for each

# [2022-05-02] Project Meeting 14

**Brief script walkthrough**

**Disussion/Conclusion**