

# DataEng: Project Assignment 2

Validate, Transform, Enhance and Store

Team Name : Data Miners

Team Members: Neha Agrawal, Geetha Chittuluri, Sukanya Kothapally, Nishna Reddy Aleti

[GITHUB LINK](#)

## A. Review the Data

Have a look at a few of the data records and see if you can determine the meaning of each record attribute. Also, have a look at [the only documentation that we have for the data](#) (provided to us by C-Tran). The documentation is not very good, so we are counting on you to give better descriptions of each field. In your submission document, provide an improved description of each data field.

**Note: Below section was completed before version2 was posted. You might find some of these fields no longer applicable to the new schema.**

OPD\_DATE + Vehicle\_ID + Act\_Time = composite key

The below table explains the data columns description

Column Name	Type	Primary Key	Min and Max value	Description
EVENT_NO_TRIP (trip_id)	Integer (32 bit)	-		Event Trip is a single run of a bus over a bus route. So, when the bus trip starts for the day, a <b>"TRIP ID"</b> gets generated. So all bread crumb readings sharing the same EVENT_NO_TRIP should be for the same vehicle, and they all correspond to that vehicle servicing a specific route.
EVENT_NO_STOP	Integer (32 bit)	-		Along the trip as the bus travels, it records the stops throughout the route. But for

				the project part 2 we don't think it has any value.
OPD_DATE (tstamp)	Date	PK	In dd:mm:yyy format	Operation day of tracking the speed of vehicles on a particular bus route
VEHICLE_ID	Integer (32 bit)	PK	Min-1776 Max-1298380	The bus(Vehicle) identification number.
METERS	Integer (32 bit)			Distance traveled by each vehicle Don't know when it's zeroed. The first breadcrumb record of a vehicle starts at
ACT_TIME (tstamp)	Integer (32 bit)	PK	hh::mm::ss format (00:00:00) To (23:59:59)	Time displayed in seconds after midnight (From midnight - midnight). The data for a particular vehicle(vehicle id) at GPS coordinates (GPS_LATITUDE, GPS_LONGITUDE) was recorded at ACT_TIME seconds after midnight at its operational date (OPD_DATE)
VELOCITY (speed)	Float	-	Min-0 max-165.533	Speed calculated from meters and actual time columns in <b>meters/second</b>
DIRECTION	Integer (32 bit)	-	Value should be between 0-359 both inclusive	The route in which the bus travels 0 - north
RADIO_QUALITY	Integer (32 bit)	-		The signal that tracks the vehicle position
GPS_LATITUDE	Float	-	Min- 45.49438 Max- 45.866858	The accurate latitude position of the bus on the trip.
GPS_LONGITUDE	Float	-	Min -122.708248 Max - 122.299477	The accurate longitude position of the bus on the trip.
GPS_SATELLITES	Float	-	Value should be between 0 - 12	Number of satellites that were used to get the position

				of the vehicle at that point of time.
GPS_HDOP	Float	-		Horizontal Dilution of Precision 1 - ideal Anything less than 1 is not considered good. 1-2 is great
SCHEDULE_DEVIATION	Integer (32 bit)	-		Current Schedule Deviation

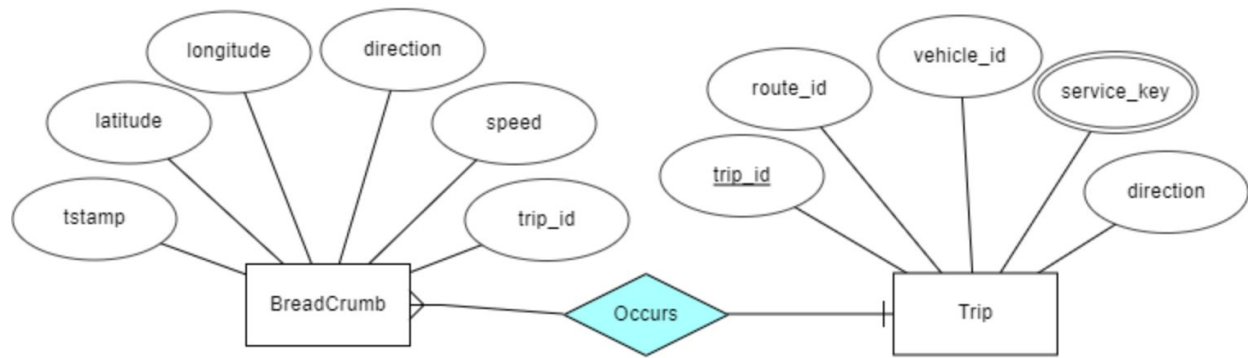
#### Version2 Changes:

1. **EVENT\_NO\_TRIP** is changed to **trip\_Id** which acts like an identifier (P.K ) of the trip table. It is also a foreign key in Breadcrumb table
2. **OPD\_DATE**( date ) and **ACT\_TIME** (time in seconds) fields have been combined to form a single field named **tstamp** which is in datetime format.
3. **VELOCITY** (originally in m/s) is changed to **speed** and it is measured in miles per hour.
4. **GPS\_LATITUDE** and **GPS\_LONGITUDE** are changed to **latitude** and **longitude** respectively.
5. **EVENT\_NO\_STOP** ,**METERS**, **RADIO\_QUALITY**, **GPS\_SATELLITES**, **GPS\_HDOP**, **SCHEDULE\_DEVIATION** are dropped from the table.

## B. Planning the Schema

As of now, the breadcrumb data consists of vehicle cyclic events for a specific route where the vehicle position updates every 5 seconds.

The ERD diagram for the table is shown below. To construct this ERD diagram, we used the ERDPlus application which is an excellent data modeling tool. (Slightly changed ER Diagram)



### Observations:

- **trip\_id:** This column is a foreign key referencing trip\_id in the Trip table. It is a unique integer identifier for the trip.
- **tstamp:** The moment at which the event occurred.(timestamp should be **yyyy-mm-dd hh:mm:ss** format)
- **direction:**Its value is in the range 0 - 359(inclusive) indicating the forward facing direction of the bus ( Direction is measured in degrees, 0 equals north. )
- **speed:** the speed of the bus at a given timestamp. Speed is measured in miles per hour.
- **Vehicle\_id:** The vehicle that serviced the trip. A trip is serviced by only one vehicle but a vehicle services potentially many trips.

### DDL Statements:

- CREATE database ctran; // **creating database**
- drop table if exists BreadCrumb;
- drop table if exists Trip;
- drop type if exists service\_type;
- drop type if exists tripdir\_type;
- CREATE Table commands:
  1. create type service\_type as enum ('Weekday', 'Saturday', 'Sunday');
  2. create type tripdir\_type as enum ('Out', 'Back');
  3. create table Trip (
    - trip\_id integer,
    - route\_id integer,
    - vehicle\_id integer,
    - service\_key service\_type,

```

        direction tripdir_type,
        PRIMARY KEY (trip_id)
    );

4. create table BreadCrumb (
    tstamp timestamp,
    latitude float,
    longitude float,
    direction integer,
    speed float,
    trip_id integer,
    FOREIGN KEY (trip_id) REFERENCES Trip
);

```

## C. Data Validation -

**Create 20 distinct data validation assertions for the bread crumb data. These should be in English (not python). Include them in your submission document (see below).**

**Then implement at least 10 of the assertions in your Kafka consumer code. Implement a variety of different types of assertions so that you can experience with each of the major types of data validation assertions: existence, limit, intra-record check, inter-record check, summary, referential integrity, and distribution assertions.**

NOTE: The assertions highlighted in **Red** are the ones we implemented.

Existence	<ol style="list-style-type: none"> <li>1. Every Breadcrumb record should have a valid not NULL tstamp</li> <li>2. Every record of Trip table should have a valid not NULL vehicle id</li> <li>3. Every record of Trip table should have a not NULL trip id</li> <li>4. Every Breadcrumb record should have a non empty tstamp field</li> </ol>
Limit	<ol style="list-style-type: none"> <li>1. Direction for each breadcrumb record should be between 0-359 inclusive</li> <li>2. Satellites count lies between 0 and 12</li> <li>3. Trip ID of Trip table should be 9 digits long (not sure)</li> <li>4. The speed field for each breadcrumb record should not exceed 250 miles/hr</li> </ol>
intra-record check	<ol style="list-style-type: none"> <li>1. Day X's bread crumbs should not occur on day X+1. (check tstamp)</li> </ol>

	<ol style="list-style-type: none"> <li>Each breadcrumb record with non zero speed field should have a non-zero direction field and vice-versa</li> </ol>
Inter-record check	<ol style="list-style-type: none"> <li>tstamp should not vary more than 5 seconds record to record</li> <li>Speed = <math>\Delta D / \Delta T</math> which is actually the difference of meters and time of two subsequent records for a specific vehicle_id.</li> </ol>
Summary	<ol style="list-style-type: none"> <li>Across all the Breadcrumb records, combination of trip id and tstamp should be unique</li> <li>Every record of Trip table should have a unique and not NULL trip id</li> <li>Every trip record should have a not NULL vehicle_id field.</li> <li>Every trip record should have a unique and not NULL trip id field</li> <li>If all the records have a NULL value for a specific field then drop that field. For e.g. Radio_Quality is not populated for breadcrumb data and it can be dropped.</li> </ol>
Referential integrity	<ol style="list-style-type: none"> <li>For each vehicle id, there should be a generated trip id - see the trip table</li> <li>Each breadcrumb record with non zero speed field should have a non-zero direction field and vice-versa</li> </ol>
Statistical distribution	<ol style="list-style-type: none"> <li>The range of speed for all the vehicles should not exceed a certain max value.</li> <li>The distance covered by a bus servicing a specific route should be increasing from midnight to midnight.</li> </ol>

## D. Data Transformation

**Add code to your Kafka consumer to transform your data. Your transformations should be driven by either the need to resolve validation assertion violations or the need to shape the data for your schema design.**

**Few Transformations on our data to align it with the schema:**

- TRANSFORMATION 1 : Extract specific selected columns to new DataFrame as a copy

```
Breadcrumbedf = df.filter(['OPD_DATE', 'ACT_TIME', 'GPS_LATITUDE', 'GPS_LONGITUDE',
'DIRECTION', 'VELOCITY', 'EVENT_NO_TRIP', 'VEHICLE_ID'], axis=1)
```

- TRANSFORMATION 2 : Replace all the empty fields by “NaN” when reading the json data into pandas dataframe.

```
Breadcrumbedf = Breadcrumbedf.replace(r'^\s*$', np.nan, regex=True)
```

- TRANSFORMATION 3 : Type cast the OPD\_DATE (which is a string format e.g. 03-SEP-20) and ACT\_TIME fields ( in seconds) to datetime in dd-mm-yyyy hh:mm:ss format. This datetime format is acceptable by postgres.

Note: We handled the timestamp column where some of day X's bread crumbs actually occur on day X+1.

Example : 3 SEP, 1day 00:04:56 becomes 4 SEP, 00:04:56 after transformation

```
tstampdf = Breadcrumbdf.filter(['OPD_DATE','ACT_TIME'], axis=1)
timestamps=[]
dic={'JAN':'01', 'FEB':'02', 'MAR':'03', 'APR':'04','MAY':'05','JUN':'06',
    'JUL':'07','AUG':'08','SEP':'09','OCT':'10','NOV':'11','DEC':'12'}
for index in range(len(tstampdf['OPD_DATE'])):
    date=df['OPD_DATE'][index][0:2]
    month=dic[df['OPD_DATE'][index][3:6]]
    year=df['OPD_DATE'][index][7:10]
    time=df['ACT_TIME'][index]
    str_time = str(timedelta(seconds=int(time)))
    if 'day' in str_time:
        add_days = ''
        for i in str_time:
            if i==' ':
                break
            add_days+=i
        date=int(date)+int(add_days)
    str_time=re.sub("[^0-9:]", "",str_time )
    dt=str(date)+month+year+str_time
    tstamp=datetime.strptime(dt, '%d%m%y%H:%M:%S')
    timestamps.append(tstamp)
```

- TRANSFORMATION 4 : Drop the "OPD\_DATE, ACT\_TIME" and insert "tstamp" into Breadcrumb data

```
Breadcrumbdf.insert(0, "tstamp", timestamps)
Breadcrumbdf.drop(columns=['OPD_DATE','ACT_TIME'],inplace=True, axis=1)
```

- TRANSFORMATION 5 : Rename all the columns of the dataframe to match the schema as shown in the section B.

```
Breadcrumbdf = Breadcrumbdf.rename(columns={"EVENT_NO_TRIP": "trip_id",
```

```
"VELOCITY": "speed", "GPS_LONGITUDE": "longitude", "GPS_LATITUDE": "latitude",  
"DIRECTION" : "direction", "VEHICLE_ID": "vehicle_id"})
```

- TRANSFORMATION 6 : Align the data types of each column of the Breadcrumb data frame to match the schema as shown in the section B.

```
Breadcrumbedf['trip_id'] = Breadcrumbedf['trip_id'].astype('Int32')  
Breadcrumbedf['vehicle_id'] = Breadcrumbedf['vehicle_id'].astype('Int32')  
Breadcrumbedf['speed'] = Breadcrumbedf['speed'].astype(float)  
Breadcrumbedf['direction'] =  
Breadcrumbedf['direction'].astype(float).astype('Int32')  
Breadcrumbedf['latitude'] = Breadcrumbedf['latitude'].astype(float)  
Breadcrumbedf['longitude'] = Breadcrumbedf['longitude'].astype(float)
```

- TRANSFORMATION 7 : Create a separate view for TRIP DF and add the route\_id, service\_key and direction columns with NaN values.

```
tripdf = Breadcrumbedf.filter(['trip_id', 'vehicle_id'])  
Breadcrumbedf = Breadcrumbedf.drop("vehicle_id", axis = 1)  
  
tripdf["direction"] = np.nan  
tripdf["service_key"] = np.nan  
tripdf["route_id"] = np.nan  
tripdf["direction"] = tripdf["direction"].astype('Int32')  
tripdf["service_key"] = tripdf["service_key"].astype(str)  
tripdf["route_id"] = tripdf["route_id"].astype('Int32')
```

#### **IMPLEMENTATION OF VALIDATION ASSERTIONS AND THEIR OUTPUT:**

*Note: This was run on 2021-02-07 breadcrumb data.*



```

=====DATA VALIDATION AND TRANSFORMATION=====

----- CASE 1: Every Breadcrumb record should have a non empty tstamp field
All the records passed Case 1 check!

=====LIMIT VALIDATIONS=====

----- CASE 2: Every Breadcrumb record should have Direction between 0-359 inclusive -----
All the records passed Case 2 check!

----- CASE 3: The speed field for each breadcrumb record should not exceed 250 miles/hr-----
-----LIMIT ASSERTION VOILATION!! Speed shouldn't exceed 250 miles/hr-----
Count of invalid records: 1

----- CASE 4: The number of satellites for breadcrumb record should be between 0 and 12-----
All the records passed Case 4 check!

=====SUMMARY VALIDATIONS=====

----- CASE 5: Every record of Breadcrumb should have unique combination of trip id and tstamp
All the records passed Case 5 check!

----- CASE 6: Every Breadcrumb record with non zero speed field should have a non-zero direction field and vice-ver
sa -----
-----REFRENTIAL INTEGRITY ASSERTION violation!! Breadcrumb records with empty Direction when the speed is
non-zero-----
Count of invalid records: 25

=====EXISTENCE VALIDATIONS=====

----- CASE 7: Every record of Trip table should have a valid not NULL vehicle id
All the records passed Case 7 check!

----- CASE 8 & 9: Every record of Trip table should have a unique and not NULL trip id
-----SUMMARY ASSERTION VOILATION!! trip ID should be unique value-----

=====REFRENTIAL INTEGRITY VALIDATIONS=====

----- CASE 10: For each vehicle id, there should be a generated trip id-----
All the records passed Case 10 check!
Finished Loading. Elapsed Time: 0.004398 seconds
Finished Loading. Elapsed Time: 1.229 seconds
(confluent-exercise) agrawal@kafka:~/examples/clients/cloud/python$ █

```

Listed below are the assertion violations which we found and resolved in our data:

- **SUMMARY ASSERTION VIOLATION!!** The trip ID should be a unique value for the trip table.

Resolved the violation by dropping the duplicate rows.

```

invalid_record_count = 0
output = pd.Series(df['trip_id']).is_unique
for item, data in enumerate(df['trip_id']):
    if( math.isnan(data)):
        df = df.drop(df.index[item])
        invalid_record_count += 1

    if (invalid_record_count > 0 and output == False):
        print("-----Summary ASSERTION VOILATION!! trip ID should be not
null and unique value-----")

    elif (invalid_record_count == 0 and output == False):
        print("-----SUMMARY ASSERTION VOILATION!! trip ID should be
unique value-----")
        df = df.drop_duplicates()

```

- **LIMIT ASSERTION VIOLATION!! Speed should not exceed 250 miles/hr**

Here we choose to “DISCARD” the violating rows.

```
for item, data in enumerate(df['speed']):
    if(data<=250 or pd.isnull(data)):
        pass
    else:
        df = df.drop(df.index[item])
```

- **REFERENTIAL INTEGRITY ASSERTION VIOLATION!! Breadcrumb records with empty Direction when the speed is non-zero**

Here we choose to “IGNORE” the violating rows.

## E. Storage in Database Server

1. Install and configure a PostgreSQL database server on your Virtual Machine. [Refer this document to learn how.](#)
2. Enhance your data pipeline to load your transformed data into your database server. You are free to use any of the data loading techniques that we discussed in class. The data should be loaded ASAP after your Kafka consumer receives the data, validates the data and transforms the data so that you have a reliable end-to-end data pipeline running daily, automatically.

**Loaded 2021-01-24 data into postgres:**

```
(confluent-exercise) agrawal@kafka:~/examples/clients/cloud/python$ sudo -u postgres psql postgres
psql (11.10 (Debian 11.10-0+deb10u1))
Type "help" for help.

postgres=# \dt
          List of relations
 Schema |   Name   | Type  | Owner
-----+-----+-----+-----
 public | breadcrumb | table | postgres
 public | test      | table | postgres
 public | trip      | table | postgres
(3 rows)

postgres=# select count(*) from breadcrumb;
 count
-----
 130745
(1 row)

postgres=# select count(*) from trip;
 count
-----
    695
(1 row)
```

## F. Example Queries

Answer the following questions about the C-Tran system using your sensor data database. In your submission document include your query code, number of rows in each query result (if applicable) and first five rows of the result (if applicable).

**Note:** All the results are for following days of breadcrumb data: 02/05, 02/06, 02/07, 02/08, 02/09, 02/11, 02/12, 02/13, 02/14

1. How many vehicles are there in the C-Tran system?

```
Select COUNT(DISTINCT vehicle_id) FROM Trip;
```

```
postgres=# Select COUNT(DISTINCT vehicle_id) FROM Trip;
count
-----
    101
(1 row)
```

**Number of rows returned : 1**

2. How many bread crumb reading events occurred on October 2, 2020?

```
Select COUNT(b.trip_id) FROM breadcrumb b, trip t where DATE(timestamp) =
'2020-10-02' and b.trip_id = t.trip_id;
```

```
postgres=# Se
count
-----
 373520
(1 row)
```

**Number of rows returned : 1**

3. How many bread crumb reading events occurred on October 3, 2020?

```
Select COUNT(b.trip_id) from breadcrumb b, trip t where DATE(timestamp) =
'2020-10-03' and b.trip_id = t.trip_id;
```

```
count
-----
 175600
(1 row)
```

**Number of rows returned : 1**

4. On average, how many bread crumb readings are collected on each day of the week?

```
select count(*) as "Breadcrumb readings Count", To_Char(tstamp, 'DAY') as "Day"
from breadcrumb group by To_Char(tstamp, 'DAY');
```

```
tamp, 'DAY');
Breadcrumb readings Count | Day
-----+-----
382261 | FRIDAY
376728 | MONDAY
352864 | SATURDAY
135257 | SUNDAY
364495 | THURSDAY
371598 | TUESDAY
4312 | WEDNESDAY
(7 rows)
```

Number of rows returned : 7

5. List the C-Tran trips that crossed the I-5 bridge on October 2, 2020. To find this, search for all trips that have bread crumb readings that occurred within a lat/lon bounding box such as [(45.620460, -122.677744), (45.615477, -122.673624)].

```
select trip_id from breadcrumb where latitude > 45.615477 and latitude <
45.620460 and longitude < -122.677744 and longitude > -122.673624 and
DATE(tstamp) = '2020-10-02';
```

```
postgres=#
677744 and
trip_id
-----
(0 rows)
```

Number of rows returned : 0

6. List all bread crumb readings for a specific portion of Highway 14 (bounding box: [(45.610794, -122.576979), (45.606989, -122.569501)]) during Mondays between 4pm and 6pm. Order the readings by tstamp. Then list readings for Sundays between 6am and 8am. How do these two time periods compare for this particular location?

```
create view DAYS as select *, extract(isodow from TSTAMP) from breadcrumb;

Select * from DAYS where latitude < 45.610794 and latitude > 45.606989 and
longitude > -122.576979 and longitude < -122.569501 and tstamp::time between
time '16:00:00' and '18:00:00' and date_part=5;
```

```
Select * from DAYS where latitude<45.610794 and latitude>45.606989 and longitude>-122.576979 and longitude < -122.569501 and tstamp::time between time '16:00:00' and '18:00:00' and date_part=2;
```

tstamp	latitude	longitude	to_char
2020-10-02 17:46:45	45.609148	-122.57619	FRIDAY
2020-10-02 17:46:50	45.608788	-122.57445	FRIDAY
2020-10-02 17:46:55	45.608413	-122.572703	FRIDAY
2020-10-02 17:47:00	45.60803	-122.570923	FRIDAY
2020-10-02 17:38:55	45.608168	-122.570962	FRIDAY

Number of rows returned : 108

tstamp	latitude	longitude	to_char
2020-09-29 17:33:39	45.609093	-122.576073	TUESDAY
2020-09-29 17:33:44	45.608737	-122.574355	TUESDAY
2020-09-29 17:33:49	45.608378	-122.572685	TUESDAY
2020-09-29 17:33:54	45.608028	-122.57104	TUESDAY
2020-09-29 17:47:20	45.608095	-122.570573	TUESDAY

Number of rows returned : 65

7. What is the maximum velocity reached by any bus in the system?

```
Select max(speed) from breadcrumb;
```

max
165.53356
(1 row)

Number of rows returned : 1

8. List all possible directions and give a count of the number of vehicles that faced precisely that direction during at least one trip. Sort the list by most frequent direction to least frequent.

```
select b.direction,count(t.vehicle_id) from breadcrumb b,trip t where b.trip_id=t.trip_id group by b.direction order by count(t.vehicle_id) desc;
```

```

select direction, count(*) as count
from breadcrumb
group by direction
order by count desc

```

direction	count
0	122551
180	71381
90	57838
270	57561
181	39518

(5 rows)

Number of rows returned : 5

9. Which is the longest (in terms of time) trip of all trips in the data?

```

create view time as select max(timestamp)-min(timestamp) as "longest", trip_id from
breadcrumb group by trip_id;

select trip_id, longest from time order by longest desc LIMIT 1;

```

```

select trip_id, longest
from time
order by longest desc
limit 1

```

trip_id	longest
169276194	7162 days 23:59:56

(1 row)

Number of rows returned : 1

10. Devise three new, interesting questions about the C-Tran bus system that can be answered by your bread crumb data. Show your questions, their answers, the SQL you used to get the answers and the results of running the SQL queries on your data (the number of result rows, and first five rows returned).

- Delete the duplicate records from breadcrumb table

```

delete from breadcrumb a using breadcrumb b where a=b and a.ctid < b.ctid;

```

- List the C-Tran vehicles which are headed towards the North.

```

select distinct(t.vehicle_id) from breadcrumb b inner join trip t on
b.trip_id=t.trip_id where b.direction=0 limit 5;

```

```

select vehicle_id
from breadcrumb
where direction=0
order by vehicle_id
limit 5

```

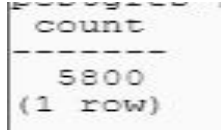
vehicle_id
2231
2293
4019
6003
2223

(5 rows)

Number of rows returned : 5

- List the C-Tran trips during the daylight time (assumed daylight between 06:00AM to 5:00PM)

```
select count(distinct(trip_id)) from breadcrumb where tstamp::time between  
time '06:00:00' and '17:00:00';
```



count  
-----  
5800  
(1 row)

Number of rows returned : 1

## DataEng Project Assignment 2 Submission Document

Construct a table showing each day for which your pipeline successfully, automatically processed one complete day's worth of sensor readings. The table should look like this:

Date	Day of Week	# Sensor Readings	# updates/insertions into your database
2021-02-05	Friday	384540	$384540(\text{breadcrumb table}) + 1661(\text{trip table}) = 386201$
2021-02-06	Saturday	175811	$175811 + 852 = 176663$
2021-02-07	Sunday	134608	$134607 + 700 = 135307$
2021-02-08	Monday	378021	$378021 + 1697 = 379718$
2021-02-09	Tuesday	371613	$371612 + 1652 = 373264$
2021-02-11	Thursday	366506	$366506 + 1679 = 368185$
2021-02-12	Friday	375777	$375777 + 1702 = 377479$
2021-02-13	Saturday	176418	$176418 + 851 = 177269$
2021-02-14	Sunday	135160	$135160 + 706 = 135866$