Step 1

- Remove entries of cars that have only gone through 1 siteid such as c1 below

probeid	siteid
c1	s1
c2	s1
c2	s2

- After wards we will have to deal with the more complicated case which exist for time differences such as c3 below which also needs removal

probeid	siteid	time
c1	s1	Monday
c2	s1	monday
c2	s2	monday
c3	s1	monday
c3	s2	Wednesday

Step 2

- Group the data by cars and then sort by time in order to get the routes for the cars

probeid	siteid	time
c2	s1	Monday 12:00:00 PM
c2	s2	Monday 12:01:00 PM
c3	s1	Monday 12:04:00 PM
c3	s2	Wednesday 6:00:00 PM
c5	s1	Monday 12:00:00 PM
c5	s7	Monday 12:30:00 PM
c5	s5	Monday 12:35:00 PM

- Using the time, calculate the time in range between consecutive siteid sensors for each car.

	Start	End	Time(s)
c2	s1	s2	60
c3	s1	s2	172800
c5	s1	s7	1800
c5	s7	s5	300

- However, we should now create an id to keep track of which start-end trip corresponds to which entry such as like in below:

tripid	probeid	siteid	time
1	c2	s1	Monday 12:00:00 PM
1	c2	s2	Monday 12:01:00 PM
2	c3	s1	Monday 12:04:00 PM
2	c3	s2	Wednesday 6:00:00 PM
3	c5	s1	Monday 12:00:00 PM
3	c5	s7	Monday 12:30:00 PM
4	c5	s5	Monday 12:35:00 PM

	Start	End	Time(s)	tripid
c2	s1	s2	60	1
c3	s1	s2	172800	2
c5	s1	s7	1800	3
c5	s7	s5	300	4

- This will be important later on when we want to remove some “trips” that are found to be outliers and also to mark the end and start of trips.

Step 3

- When we repeat this process again with a lot of car routes we should be able to build a distribution of actual trips made and the consecutive differences of time in range between all the possible two siteid routes that have been made.

probeid	Start	End	Time(s)	tripid
c2	s1	s2	60	1
c3	s1	s2	172800	2
c5	s1	s7	1800	3
c5	s7	s5	300	4
c6	s1	s2	70	5
c7	s1	s2	54	6
c8	s1	s7	1680	7

- Thus in this instance we were able to detect one of the trips as an outlier
- In theory we could consider all possible combinations/links of siteid1 to siteid2 but will be very tedious and we most likely won’t have the time for all the scenarios. So it may be assumed that since we have enough data, all the cars should go through the sites within some logical order that eliminates a lot of routes that don’t make sense practically.

Step 4

- Once we have been able to identify all outlier trips, we are then ready identify when trips begin and end. And it follows the following rule:
    - If a tripid is an outlier then it is marked as a trip point. This trip point can either later be the start or end of a trip depending on how many trips are after it for the same car.
    - If a car has no outliers, then we simply denote the first and last entry as the start and end of the trip.

tripid	probeid	siteid	Trip
1	c2	s1	start
1	c2	s2	end
2	c3	s1	NOT TRIP do not consider
2	c3	s2	NOT TRIP do not consider
3	c3	s1	start
3	c3	s7	end
4	c5	s1	start
4	c5	s7	journey
5	c5	s5	end
6	c6	s1	start
6	c6	s2	end
7	c7	s1	start
7	c7	s2	end
8	c8	s1	start
8	c8	s7	end
9	c9	s1	start
9	c9	s2	journey
10	c9	s4	journey
11	c9	s8	end

Step 5

- Then we may preprocess the data into however form we need to do other stuff. But one thing I am not sure is if there is like a data structure that we could use to do some of the stuff effectively.