# Problem Description

The problem this program tries to solve is to extract information from the location data of person. More specifically, it tries to identify if a person has visited a certain location, or a certain area, and it tries to identify the workplace and the home of the person, assuming the person has a day job.

The input data consists of the following variables:

* Latitude: The latitude of the detected GPS coordinates
* Longitude: The longitude of the detected GPS coordinates
* Timestamp: The start time of the stationary in the following format:
* YYYY = year
* MM = month of year
* dd = day of month
* HH=hourofday
* mm = minute of hour
* Z = timezone offset
* Duration: The length of time the person was stationary (in milliseconds)

# Program Design

The program uses a node structure to solve the problem, where every node is class. It tries to mirror the structure of a team, where each person has their own responsibilities and work area.

#### Engineer Node

The purpose of the engineer node is to load the data and perform the needed data engineering tasks in order to turn the data into the correct format. Every other node relies on this one for data and if additional nodes are added to the program, all that needs to be done is to extend the engineer node with a function that returns the data in the required format.

* **Function getdf(self):** Returns the original data in a dataframe.
* **Function parseData(self):** Main engineer function, that returns a dataframe with the data in the required format.
* **Function calcTimeDistance(self):** A function used by the parseData function to calculate the haversine distances between consecutive locations.

**Function getRadianCoordDf(self):** A function which returns only the latitudes and longitudes in radians.

#### Analytics Node

The purpose of the analytics node is to print out analytics from the passed in data, as well as to show histograms of distributions and a map of locations a person visited. If additional or more complex analytics are required, the node can be easily modified with additional function to accommodate the analytical needs. The input data is the latitude and longitude coordinates in radians.

* **Function printStat(self, FUN, col\_1, col\_2 = None, by = None):** The function to print out descriptive statistics about the dataframe, using pandas analytics function. e.g.: df.groupby([by])[col\_1].FUN(), where “by” is the column to group by, col\_1 is the measure we would like to analyze, col\_2 is the optional column if we are doing correlation between columns, and FUN is the function we wish to print out. FUN must be a function form the descriptive statistics:

https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#computations-descriptive-stats

* **Function showDist(self, col\_1, bins = 30):** A function to show a histogram of the distribution of a column.
* **Function showTravelLocations(self):** A function to visualize all the latitude and longitude pairs on a world map.

#### Lookup Node

The lookup node checks if a person has visited a location or an area in the past. First based on an input latitude, longitude and distance measures, the locations the person has visited is filtered out using bounding boxes with a simple distance calculation, in order to make the program more efficient. After the filtering, the data that fell within the bounding boxes then turned into distances using the haversine function to account for the Earth’s curvature. If there is a distance smaller than the input distance measure, then the function returns a positive answer, if not a negative response.

* **Function calcBondCoord(self, lat, lon, dist):** Function to calculate the bounding box(es) around the input GPS location.
* **Function lookupCoord(self, lat, lon, dist = 3):** Function to check if there is a distance smaller the input measure within the bounding box(es).

#### Detection Node

The detection node uses a DBSCAN clustering algorithm to filter out the noise from the locations, and to identify places the person has frequently visited, and the rules to identify their workplaces and their homes. The DBSCAN algorithm uses the standard Euclidean distance since we are trying to identify points that relate to the same location, and therefore the Earth’s curvature can be safely ignored in this case.

**Function createSummary(self, eps = 0.0004):** Creates a dataframe with the DBSCAN clustering algorithm to identify frequently visited locations.

**Function getHome(self):** Prints out the probable locations of a person’s home based on the following rules:

* Home is where a person regularly sleeps.
* This means that for a person with a day job would not spend the nights at home and leave for work in the mornings.This would show in the data:
* A person would spend 6 to 8 hours motionless
* Since the nights are spent at home, the min\_hour should be close to 0
* Since the nights are spent at home, the max\_hour should be close to 24
* Since it is highly unlikely for a normal person to work on weekends, only for a home, the max weekday should be 6 = Sunday

**Function getWork(self):** Prints out the probable locations of a person’s workplaces based on the following rules:

* Work is where a person with a day job spends his workdays.
* We assume that a person doesn't work on weekends and works Monday to Friday. This would show in the data:
  + min\_hour would be between 6 and 10
  + max\_hour would be between 16 and 20
  + max\_weekday would be 4 = Friday

Function printLocationMap(self, work\_home = 'home'): Function to visualize the persons work places and homes on a worldmap.