1_Preparation

February 17, 2020

1 Assignment2, CMPT826

1.1 STEP 1: Preparation

- Seyedeh Mina Mousavifar
- 11279515
- sem311

I have saved the data as a pickle object in the previous assignment, so I will just convert from a pickle object to the pandas data frame.

In [8]:
```
import pandas as pd

gps_data = pd.read_pickle('data/gps_bin_100.pkl')

# sort data
gps_data = gps_data.sort_values(['user_id', 'duty_num']).dropna()

# creating grid cell labels (x,y)
gps_data = gps_data.astype({'x_grid': 'int32',
                            'y_grid': 'int32'}).astype({'x_grid': 'str',
                                                        'y_grid': 'str'})
gps_data['grid_label'] = gps_data['x_grid'] + ',' + gps_data['y_grid']
gps_data = gps_data.astype({'x_grid': 'int32', 'y_grid': 'int32'})

gps_data
```

Out[8]:

	user_id	duty_num	lat	lon	x	y	\
0	514	1372	52.12090	-106.678000	385118.999838	5.775813e+06	
1	514	1373	52.12093	-106.678000	385119.076985	5.775817e+06	
2	514	1374	52.12090	-106.678000	385118.999838	5.775813e+06	
3	514	1375	52.12090	-106.678000	385118.999838	5.775813e+06	
4	514	1376	52.12090	-106.678000	385118.999838	5.775813e+06	
...	...	...	...	...	...	...	
249691	1364	10063	52.13645	-106.656276	386645.696759	5.777508e+06	
249692	1364	10064	52.13645	-106.656276	386645.696759	5.777508e+06	
249693	1364	10065	52.13645	-106.656276	386645.696759	5.777508e+06	
249694	1364	10067	52.13645	-106.656276	386645.696759	5.777508e+06	

```
249695     1364     10068  52.13645 -106.656276  386645.696759  5.777508e+06

        x_grid  y_grid grid_label
0           60      69      60,69
1           60      70      60,70
2           60      69      60,69
3           60      69      60,69
4           60      69      60,69
...        ...     ...        ...
249691      75      86      75,86
249692      75      86      75,86
249693      75      86      75,86
249694      75      86      75,86
249695      75      86      75,86

[249696 rows x 9 columns]
```

Code from previous assignment

In [ ]:
```
import datetime
import math
from pyproj import Proj

# retrieving saskatoon data with less than 100m accuracy
gps_data = pd.read_pickle('data/gps_filter_final_50.pkl')

# removing unnecessary columns
gps_data = gps_data.drop(['alt', 'bearing',
                          'speed', 'record_time_minute',
                          'timestamp', 'pokemon'], 1)



# sorting based on time
gps_data = gps_data.sort_values(['user_id', 'record_time']).dropna().reset_index()

# Converting record time to separate Date and Time variable
gps_data['Dates'] = pd.to_datetime(gps_data['record_time']).dt.date
gps_data['Time'] = pd.to_datetime(gps_data['record_time']).dt.time
gps_data['Hour'] = pd.to_datetime(gps_data['record_time']).dt.hour
gps_data['Minute'] = pd.to_datetime(gps_data['record_time']).dt.minute
gps_data['Second'] = pd.to_datetime(gps_data['record_time']).dt.second



# removing December test data
testdate = datetime.datetime.strptime('2016-12-09', "%Y-%m-%d").date()
gps_data = gps_data[(gps_data['Dates'] > testdate)]

# finding study start date by finding minimum date after test date in December!
```

```
start_time = gps_data.record_time.min()

# finding study end date by finding maximum date
end_time = gps_data.record_time.max()

# total number of duty cycles during study
n_duty = math.ceil((((end_time - start_time).total_seconds())/60)/5)

# first column as each duty cycle start time
start_duty = pd.date_range(start_time, periods=n_duty, freq='5min')

# getting second item of previous dataframe as first duty cycle end time
# second column as each duty cycle end time
end_duty = pd.date_range(start_duty[1], periods=n_duty, freq='5min')

duty_num = pd.Series(range(1,n_duty+1))

duty_data = pd.DataFrame({'duty': duty_num,
                          'start_time': start_duty,
                          'end_time': end_duty})

def calc_duty(time):
    '''
    This functions find duty cycle of specific time during study
    :param time: record time
    :return: duty cycle of given record time
    '''
    result = duty_data[(duty_data['start_time'] <= time) & (duty_data['end_time'] > ti
    if result.empty:
        print('no duty cycle')
    return result.iloc[0].duty

# finding duty cycle for gps records
gps_data['duty_num'] = gps_data.apply(lambda x: calc_duty(x.record_time), axis=1)



# calculating mean of latitude and longitude for every duty cycle
gps_data = gps_data.astype({'lat': 'float64', 'lon': 'float64'})
gps_data = gps_data.groupby(['user_id',
                             'duty_num']).agg(lat=('lat', 'mean'),
                                              lon=('lon', 'mean')).reset_index()


# converting to UTM
myproj = Proj('epsg:32613', proj='utm', zone=13,
              ellps='WGS84', preserve_units=True)
```

```
gps_data['x'], gps_data['y'] = myproj(gps_data['lon'].values,
                                      gps_data['lat'].values)

# binning
GRID_SIZE = 100

# find grid start point
start_x, start_y = gps_data.x.min(), gps_data.y.min()

# labeling grids
gps_data['x_grid'] = np.ceil((gps_data['x'] - start_x)/GRID_SIZE)
gps_data['y_grid'] = np.ceil((gps_data['y'] - start_y)/GRID_SIZE)

# save grid for future use
gps_data.to_pickle('data/gps_bin_100.pkl')
```