```
'''
input
    data : DataFrame / participants data
                    shape = (days*48, 3)
    normal_data : numpy array / normal data
                    shape = (days, 48, 3)
'''
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import sqlite3
import datetime

N_COMPONENTS = 2


class alert():
    def __init__(self, data, participant_id, normal_data_path, args,
db_path = "DB/dev.db"):
        self.data = data
        self.data =
self.data.groupby(["Timestamp","participantId"]).max().reset_index()
        self.data["date"] = self.data["Timestamp"].dt.strftime("%Y%m%d")

        self.normal_data = np.load(normal_data_path, allow_pickle=True)
        self.normal_days = self.normal_data.shape[0]
        self.result = pd.DataFrame()
        print(db_path)
        self.conn = sqlite3.connect(db_path)

        self.alert_table = "alert"
        self.elec1_table = "elec1"
        self.elec2_table = "elec2"
        self.sleep_table = "sleep"
        self.participant_id = participant_id
        self.args = args
        self.add_scatter = "python_scripts/DB/sql/add_scatter.sql"
        self.delete_sql = "python_scripts/DB/sql/delete.sql"

    def Run(self):
```

```
        self.result = pd.DataFrame()
        # days
        self.result["date"] = pd.to_datetime(self.data["date"].unique())
        # Anomaly
        self.result["Anomaly"] = self.anomaly_alert(self.normal_data,
"Anomaly") > self.args.anomaly_thr
        # Change Lifestyle
        self.result["ChangeLifeStyle"] =
self.anomaly_alert(self.normal_data[:,:,0], "ChangeLifeStyle") >
self.args.cls_thr
        # HeatStroke
        self.result["HeatStroke"] = self.heat_stroke() > 0

        # participantId
        self.result["participantId"] = int(self.participant_id)
        # push Alert db
        self.result.to_sql(self.alert_table, self.conn,
if_exists="append", index=False)

        self.conn.commit()
        self.conn.close()

    # Alert No1, No2
    # Culculate 2 alert(Anomaly, Change Lifestyle)
    def anomaly_alert(self, normal, mode = None):
        assert mode in ["Anomaly", "ChangeLifeStyle"]; "select correct
mode"
        if mode == "Anomaly":
            use_cols = ["InHome", "Sleep","Aircon"]

        elif mode == "ChangeLifeStyle":
            use_cols = ["Sleep"]

        normal = np.reshape(normal,(self.normal_days, -1))
        km = KMeans(n_clusters=N_COMPONENTS,
                    n_init=10,
                    max_iter=300,
                    random_state=0)
        km.fit(normal)
        mean1 = km.cluster_centers_[0]
```

```
        mean2 = km.cluster_centers_[1]
        normal_embd = [np.linalg.norm(normal - mean1[np.newaxis], axis=1),
                       np.linalg.norm(normal - mean2[np.newaxis], axis=1)]
        # culculate anomaly score per day
        Anomaly_Score = []
        created_date = datetime.datetime.today().strftime('%Y-%m-%d')
        for day in sorted(self.data["date"].unique()):
            data_one_day = self.data.query("date == @day")
            if len(data_one_day) != 48:
                Anomaly_Score.append(np.nan)
                continue
            data_one_day = np.array(data_one_day[use_cols])[np.newaxis]
            data_one_day = data_one_day.reshape((1,-1))
            pred_embd = [np.linalg.norm(data_one_day - mean1, axis=1),
                         np.linalg.norm(data_one_day - mean2, axis=1)]
            Anomaly_Score.append(np.linalg.norm(pred_embd))
            x = float(pred_embd[0])
            y = float(pred_embd[1])
            day = datetime.datetime.strptime(day, '%Y%m%d')
            day_1 = day + datetime.timedelta(days=1)
            day_1 = day_1.strftime('%Y-%m-%d')
            day = day.strftime('%Y-%m-%d')

            cur = self.conn.cursor()
            sql = open(self.delete_sql).read().format("scatterplot",
participant_id, "date", day, day_1)
            sql = sql + "AND mode = {}".format(str(mode))
            cur.execute(sql)

            sql = open(self.add_scatter).read()
            sql = sql.format(day, x, y, int(self.participant_id),
str(mode), created_date)
            cur.execute(sql)

        output = np.array(Anomaly_Score, dtype="float").flatten()
        return output

    # Alert No3(Heat Stroke)
    def heat_stroke(self):
        Heatstroke = []
```

```
        for day in self.data["date"].unique():

            data_one_day = self.data.query("date ==
@day").reset_index(drop=True)
            data_one_day["hour"] = data_one_day["Timestamp"].dt.hour
            start = self.args.heatst_st
            end = self.args.heatst_en
            temp = data_one_day["Temp"] > self.args.heatst_temp
            home = data_one_day.query("hour >= @start & hour <=
@end")["InHome"] >= 0.5
            aircon = data_one_day.query("hour >= @start & hour <=
@end")["Aircon"] < 0.5
            Heatstroke.append(np.sum(np.array(home&aircon) >= 3))
        return np.array(Heatstroke, dtype="int").flatten()
```