**Kafka Project**

Gachon univ.

Ellie, Edgar, Hayeon, Wonjun

**SET\_C**

**<Pro3. Finding nearest parking spot>**

**Description:** In this task, we will learn and practice accessing real sensors data and some simple data preparation and visualisation. We **will use open sensor data** from Melbourne's open data platform in particular on-street parking bay sensors data. This data contains information from in-ground car parking bay sensors across the city with status indicating if a car is present or not present, the spatial coordinate of the sensor, and the street marker id. We will use this data and **present the available parking spaces in the google maps** as well as **display the closest parking spot from your location.**

**Technique:**

* Python 3.7
* Gmplot, Socrata: <https://dev.socrata.com/foundry/data.melbourne.vic.gov.au/dtpv-d4pf>
* On-street parking data: <https://data.melbourne.vic.gov.au/Transport-Movement/On-street-Parking-Bay-Sensors/vh2v-4nfs>
* Euclidean distance algorithm

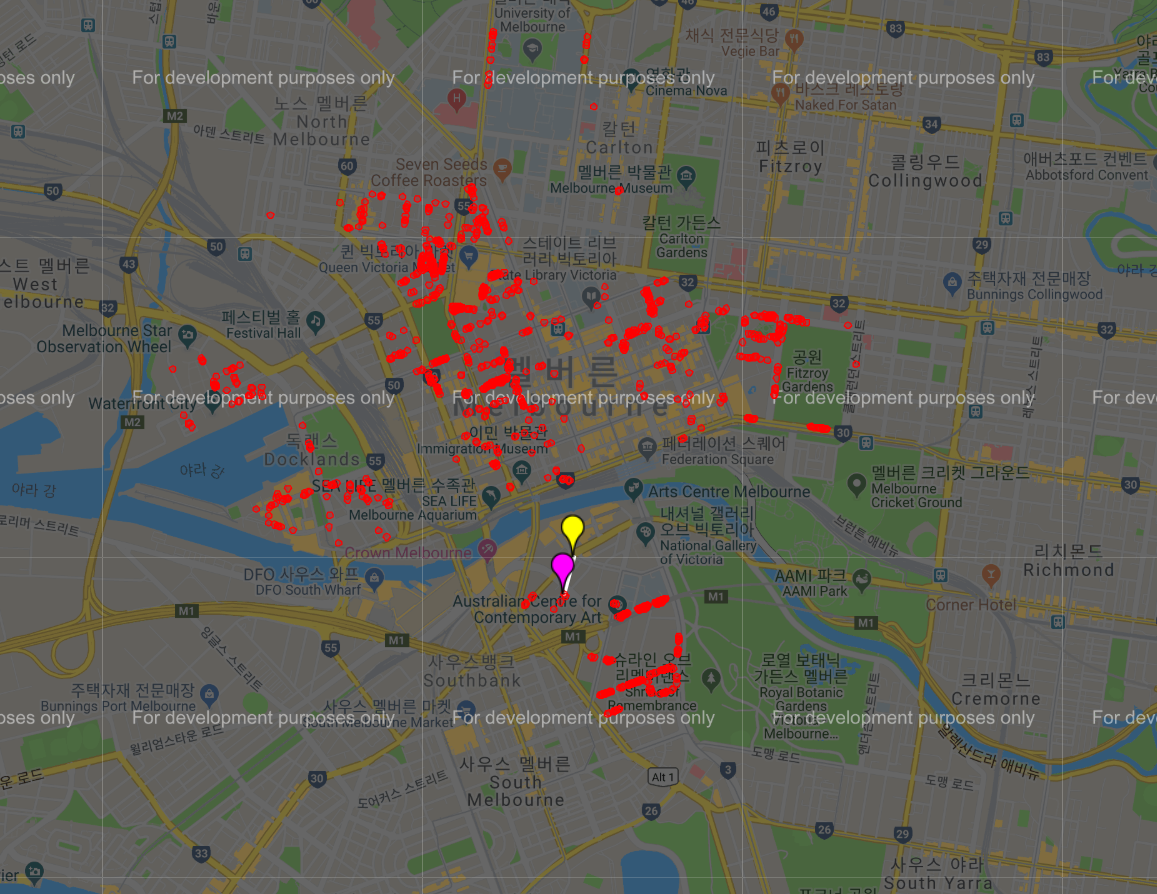
**Solving steps:**

1. Receive real time parking spot data using Socrata.
2. Convert Socrata data to pandas\_data.
3. That parking spot data are stored for coordinator.
4. Then, find the available parking space. And draw the map using that data. (scatter)
5. Using Euclidean distance algorithm, find the nearest parking spot from current location.
6. Finally, marking the closest parking spot and plotting root.

**Code**

import gmplot  
import webbrowser  
import pandas as pd  
from sodapy import Socrata  
import geocoder  
from scipy.spatial import distance  
  
# real time data using socrata  
client = Socrata("data.melbourne.vic.gov.au", None)  
results = client.get("dtpv-d4pf", limit=2000)  
  
# convert data to pandas\_data  
results\_df = pd.DataFrame.from\_records(results)  
  
# latitude list, longitude list  
lats, lons, status, lats2, lons2, = [], [], [], [], []  
coords = []  
temp, temp2 = [], []  
  
# store coordinator data to each value  
for index, row in results\_df.iterrows():  
 # print(row)  
 status.append(row[6])  
 lats.append(float(row[2]))  
 lons.append(float(row[4]))  
  
# find the available parking space  
for i in range(0, len(status)):  
 if status[i] == "Unoccupied":  
 lats2.append(lats[i])  
 lons2.append(lons[i])  
  
# my location  
myloc = geocoder.ip('me')  
cur, cur2 = myloc.lat, myloc.lng  
  
# place map Melbourne  
gmap = gmplot.GoogleMapPlotter(cur, cur2, 13)  
  
# Scatter Drawing. parking spot  
gmap.scatter(lats2, lons2, '#FF0000', size=20, marker=False)  
  
# marker about current location  
gmap.marker(cur, cur2, '#FFFF00', title="my location")  
  
# Make coordinator list  
for i in range(0, len(lats2)):  
 temp.append(lats2[i])  
 temp2.append(lons2[i])  
 coords.append([temp[i], temp2[i]])  
  
loc = [[cur, cur2], ]  
  
# calculate euclidean distance  
e\_dst = distance.cdist(loc, coords, "euclidean")  
e\_dst = e\_dst.tolist()  
idx, m\_val, i = 0, 1000000, 0  
  
# find coordinator of the closest parking spot  
for dst in e\_dst[0]:  
 if dst < m\_val:  
 m\_val = dst  
 idx = i  
 i += 1  
  
# marking the closest parking spot  
gmap.marker(coords[idx][0], coords[idx][1], '#FF00FF', title="The nearest parking spot")  
  
# plotting root  
dis\_lat = [coords[idx][0], cur]  
dis\_lng = [coords[idx][1], cur2]  
  
gmap.plot(dis\_lat, dis\_lng, '#FFFFFF', edge\_width=3)  
  
# Draw  
gmap.draw("parking\_map.html")  
# Run Browser  
webbrowser.open\_new("parking\_map.html")

**Screen shot**



**<Pro4. Find the coolest and hottest place in Australia in real time>**

**Description:** In the task you will extend the ideas from the home work last week. In your homework, you have plotted the temperature of the three campuses. **In this task,** **you will show the coolest and hottest place in real time.** You can query the open weather in real time using the API, I have given you last week. You can find the list of all the postcode of Australia in this link :[http://www.corra.com.au/australian-postcode-location-data/.](http://www.corra.com.au/australian-postcode-location-data/.%20) If for some reasons (e.g., your computer memory limitation or server refusal to response to large number of request) you cannot access the API, try only the postcodes of Victoria (VIC).

**Technique:**

* Python 3.7
* Import requests
* JSON parsing

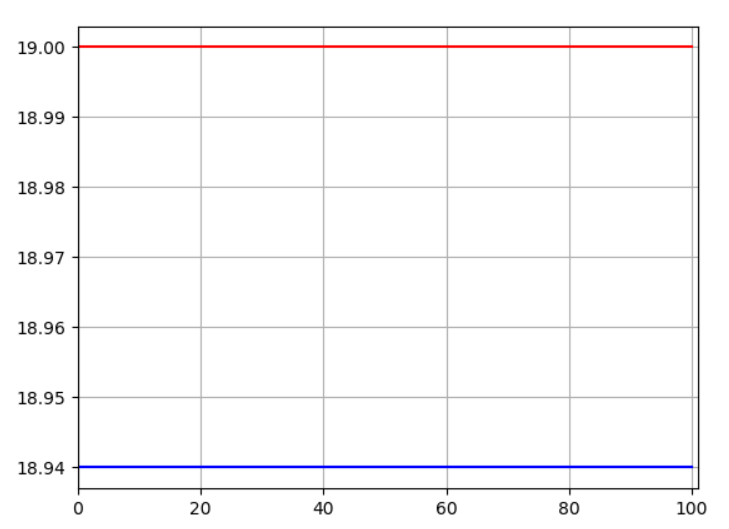
**Solving steps:**

1. Using JSON parsing, gather temperature data from the weather api.
2. And make a request to the collect the data. Then, get data as json format.
3. Find the city and temperature that have max and min temperature.
4. Draw the graph about temperature.

**Code**

import requests  
import pandas  
import time  
import matplotlib  
matplotlib.use('TkAgg')  
import matplotlib.pyplot as plt  
  
df = pandas.read\_csv('aus\_vic.csv')  
cities = []  
suburb = []  
for p in df['postcode']:  
 cities.append('http://api.openweathermap.org/data/2.5/weather?zip=' + str(  
 p) + ',au&units=metric&APPID=d69a4b6015c26ca2ef20c16aecdeaee8')  
# for a in df['suburb']:  
# suburb.append(df['suburb'])  
# print(suburb)  
  
# draw graphs  
plt.rcParams['animation.html'] = 'jshtml'  
plt.rcParams['axes.grid'] = True  
fig = plt.figure()  
ax = fig.add\_subplot(111)  
fig.show()  
i = 0  
x = []  
  
  
# json parsing  
def parsing():  
 # make a request to the collect the data  
 reses = []  
 idx = 0  
 for res in cities:  
 reses.append(requests.get(res))  
 idx += 1  
 if idx == 100:  
 break  
  
 # get data as json format  
 datas = []  
 for data in reses:  
 datas.append(data.json()['main']['temp'])  
 return datas  
  
  
datas = parsing()  
y, z = [], []  
while True:  
 ax.clear()  
 ax.set\_xlim(left=max(0, i - 100), right=i + 1)  
 x.append(i)  
 y.append(max(datas))  
 z.append(min(datas))  
 ax.plot(x, y, color='r')  
 ax.plot(x, z, color='b')  
  
 fig.canvas.draw()  
 time.sleep(2)  
 i += 10

**Screen shot**



**<Pro5. Real time Plotting of Fitbit (**[**www.fitbit.com**](http://www.fitbit.com)**) data>**

**Description:** We use heart rate API to get heart rate data. Producer transform the data in 5 minutes interval and Consumer is plotting. In Consumer , if value is less than 90 or greater then 140 then store it in the mongoDB.

**Technique:**

* Apache Kafka 1.3.5
* Python 3.7
* Kafka-python 1.4.4
* Pymongo 3.7.2
* Fitbit API

**Solving steps:**

1. Run the zookeeper server and Kafka server.
2. Create Kafka producer and Kafka consumer.
3. In producer, connect Fitbit API and server
4. Transfer value list and time list every 300seconds(5 minutes)
5. In consumer, Connect the mongoDB (command mongod).
6. And then receive each data and plotting.
7. If heart\_rate value is over 140 or under 90 , then insert the data into mongoDB

**Code:**

<Fitbit\_Consumer.py>

from json import loads  
from pymongo import MongoClient  
from kafka import KafkaConsumer  
import matplotlib  
  
matplotlib.use('TkAgg')  
import matplotlib.pyplot as plt  
  
# % matplotlib notebook  
  
# Consumer connection  
consumer = KafkaConsumer(  
 'Fitbit',  
 bootstrap\_servers=['localhost:9092'],  
 auto\_offset\_reset='earliest',  
 enable\_auto\_commit=True,  
 group\_id='my\_group',  
 value\_deserializer=lambda x: loads(x.decode('utf-8')))  
  
# Mongodb  
client = MongoClient('localhost:27017')  
collection = client.p5.p5  
  
# construct plotting  
plt.rcParams['animation.html'] = 'jshtml'  
plt.rcParams['axes.grid'] = True  
fig = plt.figure()  
ax = fig.add\_subplot(111)  
fig.show()  
i = 0  
val = []  
time\_s = []  
tmp = []  
tmp\_time = []  
tmptmp = 0  
key = []  
  
for message in consumer:  
 message = message.value  
 if 'data1' in message:  
 val.append(message['data1'])  
 tmp\_time.append(message['time'])  
 time\_s.append(i)  
  
 # heart\_rate < 90 or heart\_rate > 140 , insert into mongoDB  
 if message.get('data1') < 90 or message.get('data1') > 140:  
 collection.insert\_one(message)  
 print(message.get('data1'))  
  
 plt.title('plotting heartbeat')  
 ax.clear()  
 ax.set\_xlim(left=max(0, i - 10), right=i + 1)  
 # y axis  
 # if i==100:  
 # break  
 ax.plot(tmp\_time, val, color='r')  
 plt.gcf().autofmt\_xdate()  
 fig.canvas.draw()  
 i += 1  
  
plt.close('all')

Fitbit\_Producer.py

import fitbit  
import gather\_keys\_oauth2 as Oauth2  
import datetime  
from time import sleep  
from json import dumps  
from kafka import KafkaProducer  
import matplotlib  
  
matplotlib.use('TkAgg')  
  
# Update Fitbit API client\_id and client\_secret  
CLIENT\_ID = '22DD2Z'  
CLIENT\_SECRET = '34e491d057e9f7ac217349eea6876823'  
  
# connect to server  
server = Oauth2.OAuth2Server(CLIENT\_ID, CLIENT\_SECRET)  
server.browser\_authorize()  
  
ACCESS\_TOKEN = str(server.fitbit.client.session.token['access\_token'])  
REFRESH\_TOKEN = str(server.fitbit.client.session.token['refresh\_token'])  
auth2\_client = fitbit.Fitbit(CLIENT\_ID, CLIENT\_SECRET, oauth2=True, access\_token=ACCESS\_TOKEN,  
 refresh\_token=REFRESH\_TOKEN)  
  
# to loading todays time info  
today = str(datetime.datetime.now().strftime("%Y-%m-%d"))  
  
fit\_statsHR = auth2\_client.intraday\_time\_series('activities/heart', base\_date=today, detail\_level='1sec')  
  
time\_list = []  
val\_list = []  
  
for i in fit\_statsHR['activities-heart-intraday']['dataset']:  
 val\_list.append(i['value'])  
 time\_list.append(i['time'])  
  
# Make a Kafka producer  
producer = KafkaProducer(bootstrap\_servers=['localhost:9092'],  
 value\_serializer=lambda x:  
 dumps(x).encode('utf-8'))  
# Update the data every 5 minutes  
i = 0  
for e in range(len(val\_list)):  
 data1 = {'data1': val\_list[i], 'time': time\_list[i]}  
 # transfer the data(heart\_rate , time)  
 producer.send('Fitbit', value=data1)  
 print(val\_list[i], time\_list[i])  
  
 sleep(300)  
 i += 1

**Screen shot**

****