빅데이터 term project 보고서

2019147531 박윤정

전체적인 pipeline은 다음과 같습니다.

- Geographical clustering (using Meanshift clustering)
- Visual Clustering (extract features using VGG16 & Meanshift clustering)
- Using Tag information & additional input (landmark dictionary)

일단 guideline 대로 geographical matching –> visual matching -> using textual information을 하기로 했습니다. 이미지의 개수가 4851개로 양이 많아 geographical clustering으로 어느 정도 landmark들을 분류하는 게 낫다고 생각했기 때문입니다. 그런 다음, visual clustering을 통해 다른 landmark가 같은 geo-cluster에 있는 경우를 해결하고, 마지막으로 tag 정보를 사용해 visual clustering을 했음에도 다른 landmark가 같은 visual cluster에 있는 경우와, 같은 landmark가 다른 geo cluster에 있는 경우를 해결하고자 하였습니다.

Read Photo.csv file & Preprocessing

In [86]:
```
import os
from os.path import join
import matplotlib.pyplot as plt
from random import randint
import pickle
import re #pre-process tag data
import pandas as pd
import numpy as np

firstposition = os.getcwd()
print(os.getcwd())
```

```
C:₩Users₩dbw21₩Desktop₩jupyter-notebook
```

In [3]:
```
photo_path = os.path.join(os.getcwd(),'Photos')
print(len(os.listdir(photo_path))) #4851 #127 = 4978
```

필요한 것들을 import해주고 Photos folder가 위치한 곳을 photo_path라고 하였습니다.

read CSV file

```
In [5]: csv_path = os.path.join(os.getcwd(),'Photo (1).csv')
        meta_data = pd.read_csv(csv_path,encoding='cp949', names = ['photo_ID','user_ID','latitude','longitude','datetime','-1'])

In [6]: meta_data #991766 rows × 6 columns
```

Out[6]:

	photo_ID	user_ID	latitude	longitude	datetime	-1
0	8230341121	24691457@N00	47.610007	-122.340823	11/30/2012 12:22:19 AM	-1
1	8230337045	41132143@N00	47.520077	-122.297433	11/30/2012 12:20:08 AM	-1
2	8231412802	24691457@N00	47.611099	-122.337157	11/30/2012 12:27:15 AM	-1
3	8230344741	24691457@N00	47.611099	-122.337157	11/30/2012 12:24:21 AM	-1
4	8231415454	24691457@N00	47.611099	-122.337157	11/30/2012 12:28:40 AM	-1
...	...	...	...	...	...	...
991761	5898739929	47919464@N07	47.654976	-122.301175	7/3/2011 11:48:22 PM	-1
991762	5899308576	29296993@N02	47.613987	-122.341397	7/3/2011 11:49:47 PM	-1
991763	5899306798	29296993@N02	47.613987	-122.341397	7/3/2011 11:49:04 PM	-1
991764	5899303004	29296993@N02	47.613987	-122.341397	7/3/2011 11:47:27 PM	-1
991765	5898721139	29296993@N02	47.613987	-122.341397	7/3/2011 11:41:01 PM	-1

991766 rows × 6 columns

Pandas.read_csv를 통해 meta data를 읽어오고 이때 각 column을 photo_ID, user_ID, latitude, longtitude, datetime, -1로 읽어왔습니다.

Preprocessing csv file

- Photo_ID가 중복인 행을 제거
- Photos 폴더 안에 있는 사진들의 행만 남기기

```
In [7]: #photo_ID 중복 행 제거
        rmdup_metadata = meta_data.drop_duplicates(subset = 'photo_ID', keep='first', ignore_index = True)

In [8]: rmdup_metadata #990696 rows × 6 columns
```

Out[8]:

	photo_ID	user_ID	latitude	longitude	datetime	-1
0	8230341121	24691457@N00	47.610007	-122.340823	11/30/2012 12:22:19 AM	-1
1	8230337045	41132143@N00	47.520077	-122.297433	11/30/2012 12:20:08 AM	-1
2	8231412802	24691457@N00	47.611099	-122.337157	11/30/2012 12:27:15 AM	-1
3	8230344741	24691457@N00	47.611099	-122.337157	11/30/2012 12:24:21 AM	-1
4	8231415454	24691457@N00	47.611099	-122.337157	11/30/2012 12:28:40 AM	-1
...	...	...	...	...	...	...
990691	5898739929	47919464@N07	47.654976	-122.301175	7/3/2011 11:48:22 PM	-1
990692	5899308576	29296993@N02	47.613987	-122.341397	7/3/2011 11:49:47 PM	-1
990693	5899306798	29296993@N02	47.613987	-122.341397	7/3/2011 11:49:04 PM	-1
990694	5899303004	29296993@N02	47.613987	-122.341397	7/3/2011 11:47:27 PM	-1
990695	5898721139	29296993@N02	47.613987	-122.341397	7/3/2011 11:41:01 PM	-1

990696 rows × 6 columns

Drop_duplicates를 사용하여 중복 행인 경우 처음의 행만 남기도록 하였습니다.

dictionary(photo_ID <-> index in dataframe)

In [9]:
```
photo_IDs = rmdup_metadata.photo_ID.values
#print(type(photo_IDs))

#dictionary: index(key) --> photo_ID(value)
index_to_photoID = {}
for i in range(0,len(photo_IDs)):
  index_to_photoID[i] = photo_IDs[i]

#dictionary: index(value) <-- photo_ID(key)
photoID_to_index = {}
for i in range(0, len(photo_IDs)):
  photoID_to_index[photo_IDs[i]] = i
```

Photos 폴더안에 있는 이미지의 Photo_ID행만 남기기 위해, dataframe에서의 index가 key, photo_ID가 value인 "index_to_phtoID" 와 phto_ID가 key, index가 value인 "photoID_to_index" dictionary 2개를 만들었습니다.

Photos folder안에 있는 photo_ID만 남기기

In [10]:
```
#Photos folder안에 있는 것만 남기기

print(len(os.listdir(photo_path))) #4978
sub_photo_ID = []

# get photo_ID list in Photos folder
for index,filename in enumerate(os.listdir(photo_path)):
  if filename.endswith(".jpg"): #except folder. only image files
      file = filename.rstrip(".jpg")
      sub_photo_ID.append(int(file))
print(len(sub_photo_ID)) #4851

#Photos folder안에 있는 photo_ID list를 dataframe에서 인덱스 list로
sub_phto_ID_index = []
for idx, i in enumerate(sub_photo_ID):
  index = photoID_to_index[i]
  sub_phto_ID_index.append(index)

print(len(sub_phto_ID_index)) #4851
```

```
4978
4851
4851
```

Photos folder안에 있는 photo들의 photoID들을 sub_photo_ID에 저장하고 photoID_to_index를 이용해 photos folder안에 있는 photo ID를 dataframe에서 index로 구해 sub_phto_ID_index에 저장했습니다.

In [11]:
```
new_df = rmdup_metadata.loc[[i for idx, i in enumerate(sub_phto_ID_index)], :]
new_df #4851 rows × 6 columns
```

Out[11]:

	photo_ID	user_ID	latitude	longitude	datetime	-1
9175	3286772707	24537256@N02	47.620483	-122.349286	2/17/2009 10:16:21 AM	-1
9177	3286773249	24537256@N02	47.620483	-122.349286	2/17/2009 10:16:51 AM	-1
9191	3287135921	61565201@N00	47.608535	-122.337666	2/17/2009 2:28:05 PM	-1
9193	3287136491	61565201@N00	47.608535	-122.337666	2/17/2009 2:28:22 PM	-1
9194	3287136871	61565201@N00	47.608535	-122.337666	2/17/2009 2:28:33 PM	-1
...	...	...	...	...	...	...
502199	8652137449	66356066@N02	47.607500	-122.343334	4/15/2013 7:10:22 PM	-1
502224	8653224746	66356066@N02	47.620500	-122.349000	4/15/2013 7:05:25 PM	-1
502247	8653225900	66356066@N02	47.620333	-122.349167	4/15/2013 7:05:51 PM	-1
502218	8653237196	66356066@N02	47.607500	-122.343334	4/15/2013 7:10:12 PM	-1
502190	8653238002	66356066@N02	47.607500	-122.343334	4/15/2013 7:10:32 PM	-1

4851 rows × 6 columns

[12]:
```
#reset index in dataframe
new_df = new_df.reset_index(drop=True)
new_df
```

t[12]:

	photo_ID	user_ID	latitude	longitude	datetime	-1
0	3286772707	24537256@N02	47.620483	-122.349286	2/17/2009 10:16:21 AM	-1
1	3286773249	24537256@N02	47.620483	-122.349286	2/17/2009 10:16:51 AM	-1
2	3287135921	61565201@N00	47.608535	-122.337666	2/17/2009 2:28:05 PM	-1
3	3287136491	61565201@N00	47.608535	-122.337666	2/17/2009 2:28:22 PM	-1
4	3287136871	61565201@N00	47.608535	-122.337666	2/17/2009 2:28:33 PM	-1
...	...	...	...	...	...	...
4846	8652137449	66356066@N02	47.607500	-122.343334	4/15/2013 7:10:22 PM	-1
4847	8653224746	66356066@N02	47.620500	-122.349000	4/15/2013 7:05:25 PM	-1
4848	8653225900	66356066@N02	47.620333	-122.349167	4/15/2013 7:05:51 PM	-1
4849	8653237196	66356066@N02	47.607500	-122.343334	4/15/2013 7:10:12 PM	-1
4850	8653238002	66356066@N02	47.607500	-122.343334	4/15/2013 7:10:32 PM	-1

4851 rows × 6 columns

Photos folder안에 있는 photo_ID행만 남기기 위해 loc을 이용해 새로운 dataframe을 만들고 Index를 0부터 다시 정해주었습니다.

이전과 마찬가지로, 새로운 dataframe의 index와 photo_ID간의 dictionary2개를 만들었습니다.

dictionary (photo_ID <-> index in new_df)

n [13]:
```
n_photo_IDs = new_df.photo_ID.values
#print(type(photo_IDs))

#dictionary: index(key) --> photo_ID(value)
n_index_to_photoID = {}
for i in range(0,len(n_photo_IDs)):
  n_index_to_photoID[i] = n_photo_IDs[i]

#dictionary: index(value) <-- photo_ID(key)
n_photoID_to_index = {}
for i in range(0, len(n_photo_IDs)):
  n_photoID_to_index[n_photo_IDs[i]] = i
```

Step1. Geo-clustering with meanshift clustering

Latitude와 longitude를 이용하여 geo-clustering을 하기 위해 matrix를 만들었습니다.

Matrix의 column[0]은 latitude, column[1]은 longitude이고 각 행은 한 photo의 latitude, longitude 입니다.

geo matrix 만들기 -> meanshift clustering

[

[latitiude0,longitude0],

[latitiude1,longitude1]

]

[14]:
```
latitude = new_df.latitude.values
longitude = new_df.longitude.values

#print(latitude[0])

# create 0 matrix
X = [[0 for x in range(2)] for x in range(len(latitude))]

#print(X.shape)

for i in range(0, len(latitude)):
  X[i][0] = float(latitude[i])
  X[i][1] = float(longitude[i])

X = np.array(X) #순서 중요!
print(X.shape)
```

```
(4851, 2)
```

Meanshift clustering

[15]:
```
from sklearn.cluster import MeanShift

geo_clustering = MeanShift(bandwidth=0.0015,bin_seeding = True)#, cluster_all = False) #,cluster_all = False)  #bin_seeding = True, clus
#bandwidth=0.0015 -> 127개
geo_clustering.fit(X)
geo_cluster_label = geo_clustering.labels_

labels_unique = np.unique(geo_cluster_label)
print(labels_unique)
n_cluster = len(labels_unique)

print("number of geo-clusters: ", n_cluster)
```

```
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126]
number of geo-clusters:  127
```

K-means clustering이 대표적인 clustering algorithm이지만 사전에 클러스터의 개수를 알거나 결정

해야한다는 문제가 있어 meanshift clustering을 사용하여 geo-clustering을 하기로 하였습니다.

이때, Bandwidth가 크면 한 cluster내에 여러 개의 landmark가 속하고, bandwidth가 작으면 같은 landmark가 다른 cluster에 속할 수 있기에 최대한 다른 landmark가 한 cluster안에 속하지 않게, 1개의 image만 갖는 cluster가 너무 많아지지 않게 geo-cluster결과를 각 폴더(= each geo_cluster) 안에 저장하고 확인하면서 empirical하게 정했습니다.

Geo Cluster 결과대로 폴더 만들어 copy한 후 눈으로 결과 확인하기

```
import shutil

photo_path = os.path.join(os.getcwd(),'Photos')

count_num = 0 #count number of all contents of geo_clusters

# 폴더 만들기 + 해당하는 파일 찾아 카피
for idx, i in enumerate(labels_unique):
    #index_list = []
    folder_name = "geo_cluster"+ str(i) # label number
    print("folder_name is", folder_name)
    folder_path = os.path.join(photo_path, folder_name)
    if(os.path.isdir(folder_path) == False):
        os.mkdir(folder_path)                  ## 제출 시 주석 해제  # 폴더 생성 코드
      #폴더 지우는 코드
      #os.rmdir(folder_path)
    index_cluster_k = []
    for j in range(len(geo_cluster_label)):
        if geo_cluster_label[j] == i: #i = k (unique label)
            index_cluster_k.append(j)

    #photoID value
    labelmk_ID = []
    for index, j in enumerate(index_cluster_k):
        labelmk_ID.append(n_index_to_photoID[j]) # 해당하는 포토 아이디 찾기 코드
    print(len(labelmk_ID))
    count_num += len(labelmk_ID)
```

```
    # 제출 시 주석 해제
    for index, photoID in enumerate(labelmk_ID):
        file_name = str(photoID)+'.jpg'
        src_dir = os.path.join(photo_path, file_name)
        dst_dir = os.path.join(photo_path, folder_name, file_name)
        shutil.copyfile(src_dir, dst_dir)#shutil.move(src_dir, dst_dir) # 해당하는 포토 파일 명 찾아 폴더 안에 이동

#geo_cluster내의 photo_IDs array{i}로 저장하기 -> tag values구할 때 사용
# 주석 해제하기
for i in range(n_cluster):
    globals()['array{}'.format(i)] = []
    folder_name = "geo_cluster"+ str(i) # label number
    #print("folder_name is", folder_name)
    folder_path = os.path.join(photo_path, folder_name)
    for idx, j in enumerate(os.listdir(folder_path)):
        if j.endswith(".jpg"):
            globals()['array{}'.format(i)].append(j.rstrip(".jpg"))
    #print(len(globals()['array{}'.format(i)]))
#print(array0)

print("count_num", count_num)
```

Geographical clustering 결과 분석

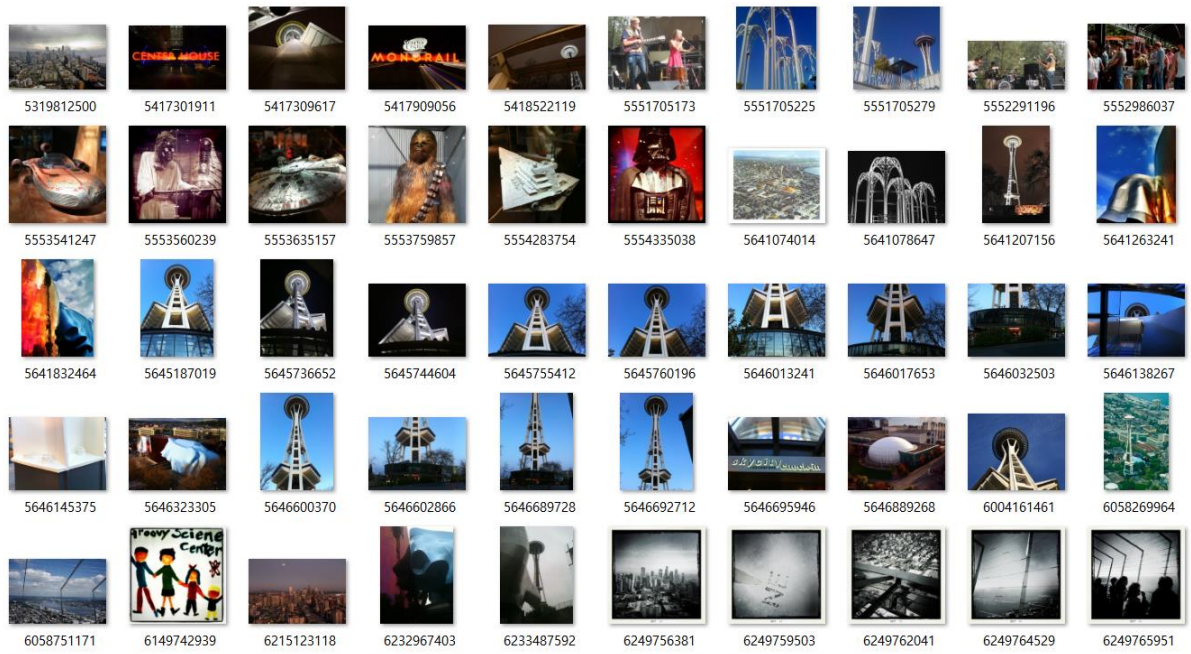
결과를 살펴보면, 다음과 같은 경우가 있었습니다.

1) 한 geo_cluster안에 한 개의 landmark만 존재 (잘 geo-clustering된 경우)

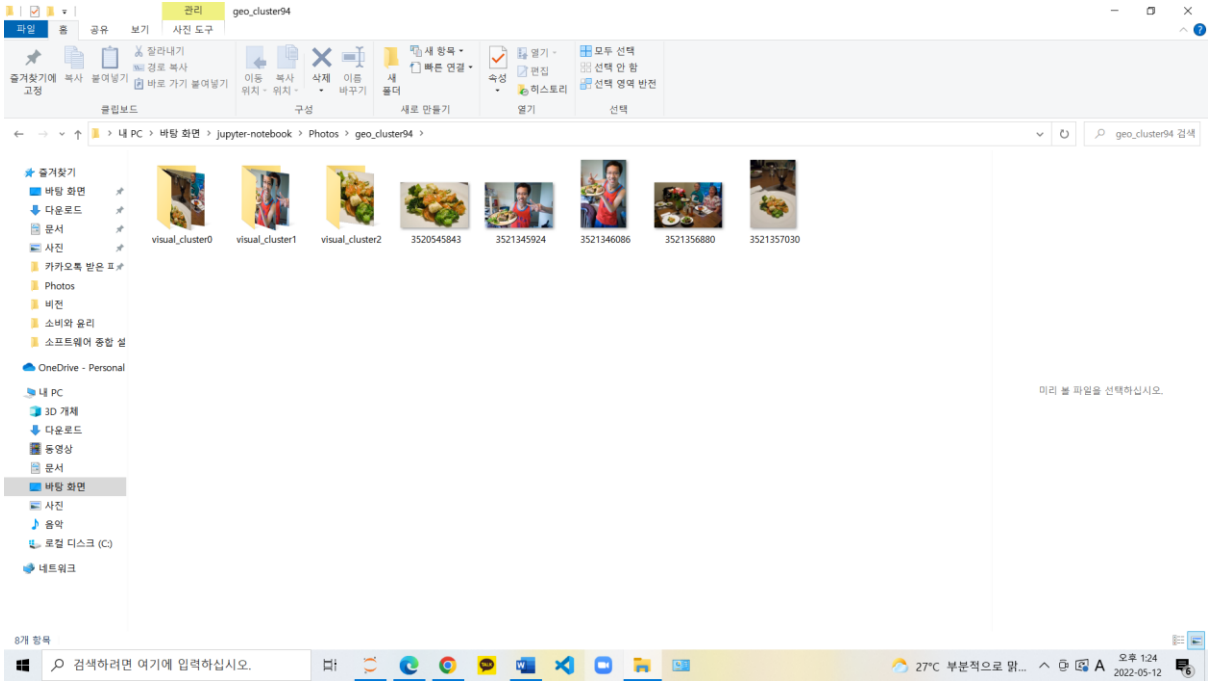
2) 한 geo_cluster안에 여러 개의 landmark, outlier 존재

3) 한 geo_cluster안에 Outlier만 존재

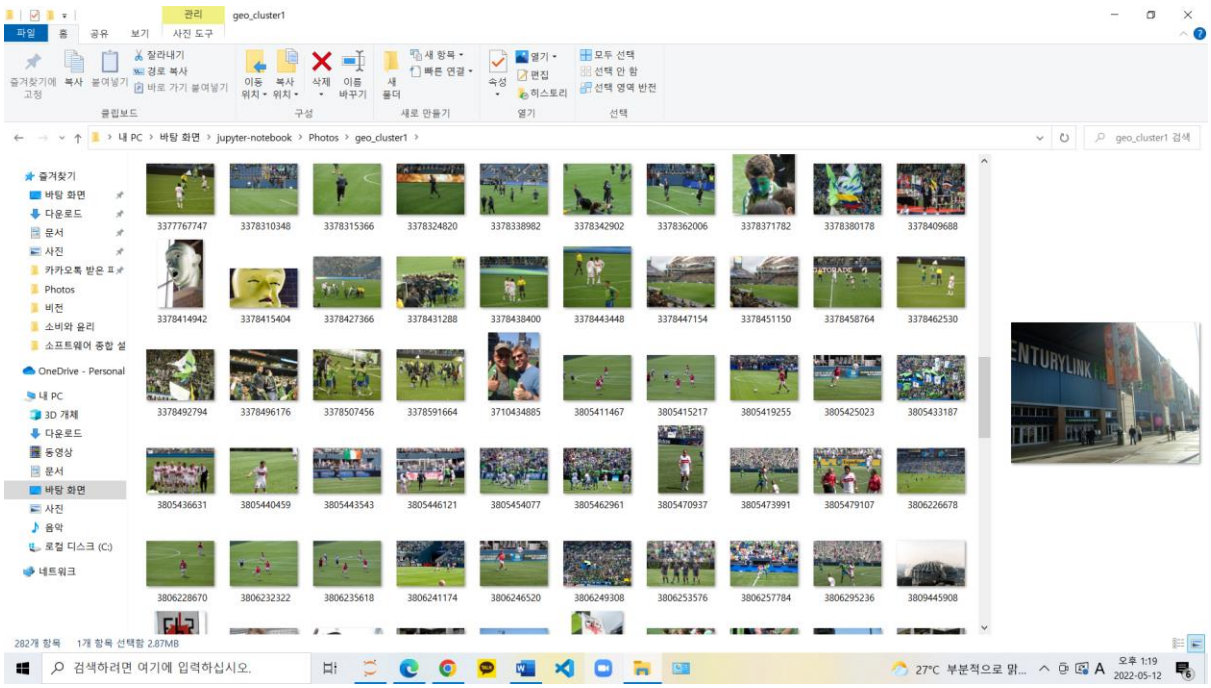
4) 같은 landmark가 다른 geo_cluster에 존재



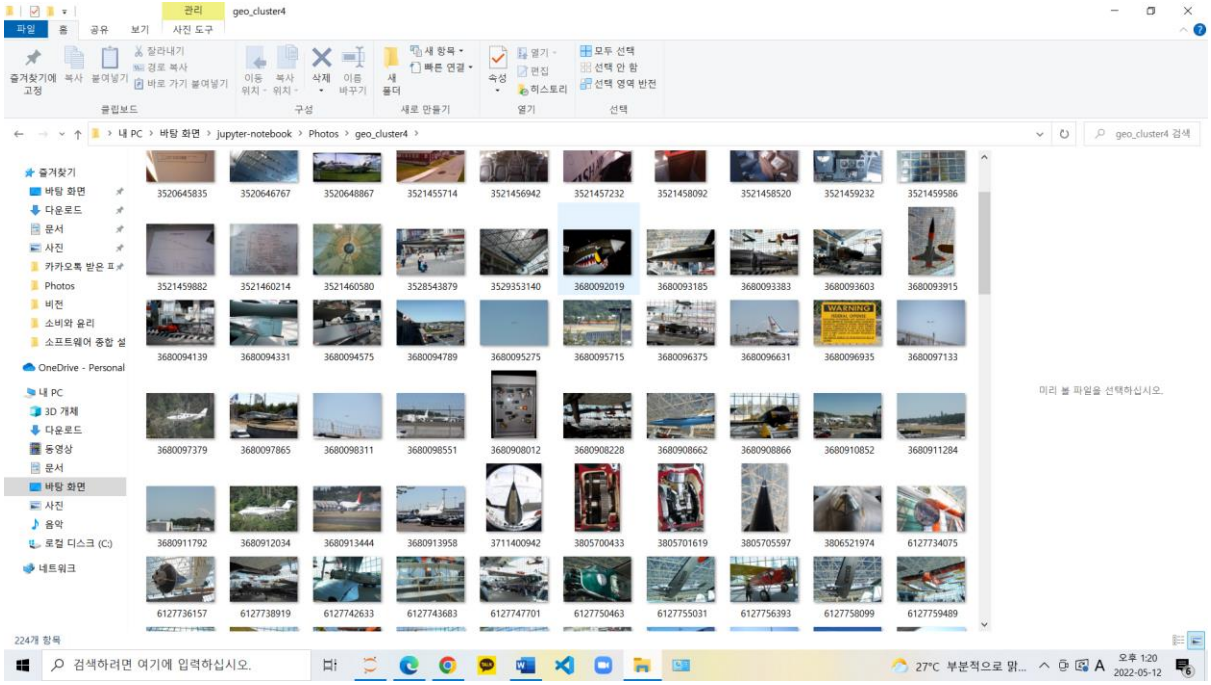
Geo_cluster0의 경우, space needle뿐만 아니라 monorail, outliers가 같이 속해 있었습니다.



Geo_cluster94의 경우, landmark가 아닌 outlier로만 이루어져있습니다.

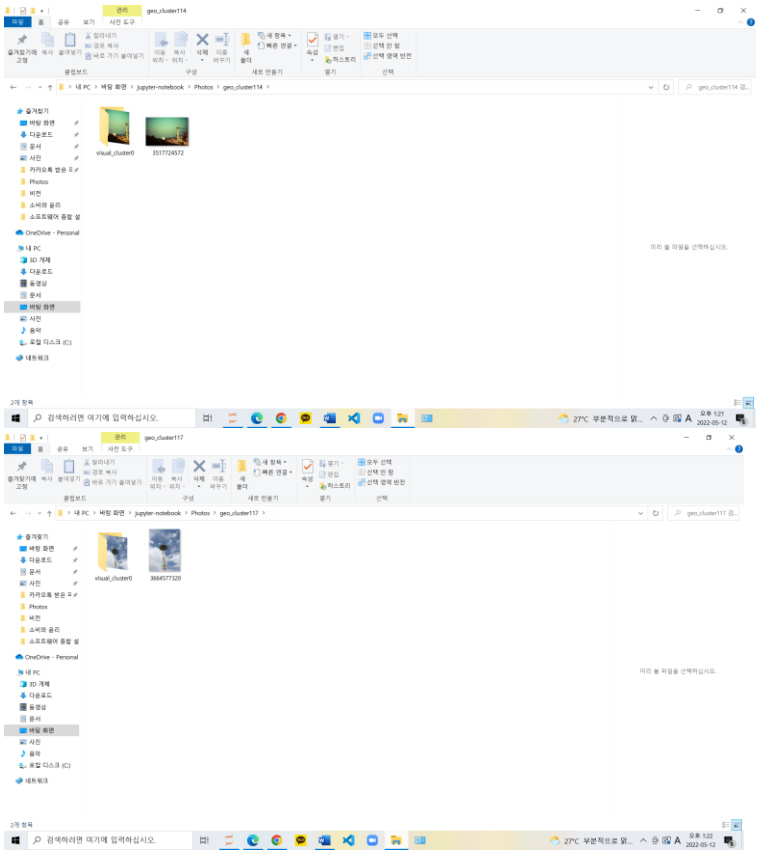


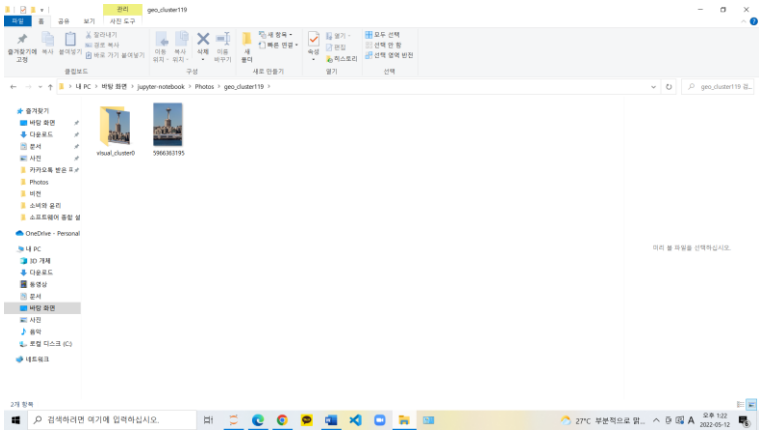
Geo_cluster1의 경우, century link field만으로 잘 clustering되어 있음을 볼 수 있었습니다.



마찬가지로, geo_cluster4의 경우, 항공 박물관(aviation space museum) 사진만으로 잘 clustering되어 있음을 볼 수 있었습니다.

또, 같은 landmark가 다른 geo_cluster에 속하는 경우도 있었는데,





Space needle의 경우, 찍힌 위치가 달라서 다른 geo_cluster로 clustering되었습니다.

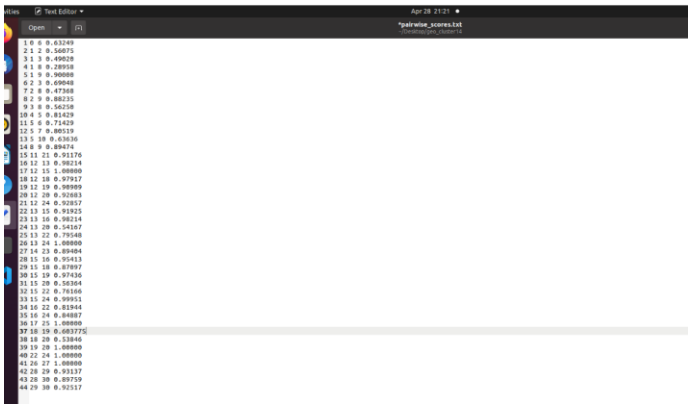
이처럼, 한 geo-cluster안에 한 개의 landmark만 있게 잘 clustering되기도 했지만, 하나의 geo-cluster안에 여러 개의 landmark와 outlier가 있거나 outlier만 있거나, 같은 landmark가 다른 geo-cluster에 속하는 경우가 있었습니다. 따라서, 한 geo-cluster에 대해 추가적으로 visual clustering을 함으로써 하나의 geo-cluster안에 여러 개의 landmark가 있거나 landmark와 outlier가 같이 있는 경우를 해결하려고 하였습니다.

Step2. Visual Clustering (extract features using VGG16 & Meanshift clusterig)

처음에 visual-clustering을 guideline을 따라 bundler를 사용하려고 했습니다. Bundler를 설치하고

32bit library가 없는 문제가 생겨 일일이 sudo apt get install로 설치하였고 RunBundler.sh에서 rm-rf pairwise_score.txt를 주석처리하고 그 외의 outputfile은 삭제하게 수정하여 번들러 실행 시 pairwise_score.txt만 생기게 하였습니다. 또, 대용량 파일을 vmware에 옮기기 위해 윈도우와 리눅스 공유 폴더를 만들었습니다.

다음과 같이 31장의 이미지를 갖는 한 geo-cluster에 대해 pairwise_score.txt가 생성되었고



Pairwise_score를 갖는 두 component가 같은 graph에 속하게 Pairwise_score.txt에 따라 graph를 만들고 한 graph의 components들이 같은 visual cluster에 존재하는 것으로 보고 visual clustering을 하려고 했으나 두가지 문제가 있었습니다.

첫 번째 문제는, 번들러를 돌리는 시간이 너무 오래 걸리는 것이었습니다. 31개의 이미지를 갖는 geo_cluster에 대해 번들러를 실행했을 때 2시간 이상이 걸렸고, 총 4851장의 이미지이기에 매우 많은 시간이 소요될 것으로 예상이 되는 문제가 있었습니다.

두 번째 문제는 이미지 크기와 관련하여 row+col< 8000이어야하는 조건이 있어 이미지 크기를 조정해야하는 문제가 있었습니다.

```
Finding keypoints...
21645 keypoints found.
Finding keypoints...
sift: util.c:361: ConvVertical: Assertion `rows + ksize < 8000' failed.
Aborted (core dumped)
Finding keypoints...
sift: util.c:361: ConvVertical: Assertion `rows + ksize < 8000' failed.
Aborted (core dumped)
Finding keypoints...
7677 keypoints found.
Finding keypoints
```

또, pairwise_score.txt의 결과가 이미지가 많은 cluster일 경우, 좋지 않았기에 bundler대신 다른 방식으로 visual clustering을 진행하기로 결정했습니다.

Visual clustering을 하는 방법에 대해 찾아보던 중, pretrained VGG16 모델을 사용해 output이 마지막 layer가 아닌 그 전 layer값이 되도록 하여 feature vector를 extract하여, extract한 feature vecotr를 PCA를 통해 차원을 축소한 후 k-mean clustering으로 visual-similarity에 따라 image를 clustering한 코드를 참고하여, 각 geo-cluster images를 pretrained VGG16을 사용해 feature vectors를 뽑아낸 다음, PCA를 통해 n_components로 차원을 축소하고 meanshift clustering으로 image를 clustering하였습니다. Geo clustering과 마찬가지로, cluster의 개수를 알 수 없기에 kmean 대신 meanshift algorithm을 사용했으며, 각 geo cluster안의 images 개수가 달라, PCA 시 n_components가 mean(n_samples, n_features)이기 때문에, n_components 값을 cluster image에 따라 다르게 해줘야한다는 문제가 있었습니다. 또, geo_cluster안의 image 개수(=n_samples)가 작아 n_componenets를 작게 한 경우 PCA 후 meanshift clustering을 할 때 bandwidth를 너무 크게 설정하면 구별이 어려워져 다른 landmark 이미지들이 같은 visual cluster에 속하는 문제가 있어, geo_cluster의 image개수에 따라 다른 for loop안에서 PCA시 n_components, meanshift clustering 시 bandwidth를 empirical하게 적절히 정해 visual clustering을 해주었습니다.

아래는 코드에 대한 설명입니다.

visual clustering

In [19]:
```
# using VGG16 -> extract feature vectors
# for loading/processing the images
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input

# models
from keras.applications.vgg16 import VGG16
from keras.models import Model

# clustering and dimension reduction
# from sklearn.cluster import KMeans
from sklearn.cluster import MeanShift
from sklearn.decomposition import PCA


model = VGG16()
model = Model(inputs = model.inputs, outputs = model.layers[-2].output)

def extract_features(file, model):
    # load the image as a 224x224 array
    img = load_img(file, target_size=(224,224))
    # convert from 'PIL.Image.Image' to numpy array
    img = np.array(img)
    # reshape the data for the model reshape(num_of_samples, dim 1, dim 2, channels)
    reshaped_img = img.reshape(1,224,224,3)
    # prepare image for model
    imgx = preprocess_input(reshaped_img)
    # get the feature vector
    features = model.predict(imgx, use_multiprocessing=True)
    return features
```

Pretrained VGG16 모델(Image classification model)을 불러오고, output이 마지막 layer전의 layer값 이어서 1000개의 class일 확률 값이 아닌 feature vectors를 추출합니다.

]:
```
### 0-12 geo clusters(which have more than 100 images)
for i in range(13):
# for i in range(2,n_cluster):
    geo_cluster_path = os.path.join(photo_path, 'geo_cluster'+ str(i))#"/content/drive/My
    print("geo_cluster_path is {}".format(geo_cluster_path))
    # # change the working directory to the path where the images are located
    os.chdir(geo_cluster_path)

    # this list holds all the image filename
    images = []

    # creates a ScandirIterator aliased as files
    with os.scandir(geo_cluster_path) as files:
    # loops through each file in the directory
        for file in files:
            if file.name.endswith('.jpg'):
            # adds only the image files to the flowers list
                images.append(file.name)
    print("len of image is {}".format(len(images)))
#     print(images)

    data = {}
    txt_file_name = 'geo'+str(i)+'_featurevectors.txt'
    txt_file_path = os.path.join(photo_path, txt_file_name)
    print("txt_file_path is {}".format(txt_file_path))
    #r"CHANGE TO A LOCATION TO SAVE FEATURE VECTORS"

    # lop through each image in the dataset
    for image in images:
        # try to extract the features and update the dictionary
        try:
            feat = extract_features(image,model)
            data[image] = feat
        # if something fails, save the extracted features as a pickle file (optional)
        except:
```

```
        except:
            with open(txt_file_path,'wb') as file:
                pickle.dump(data,file)

    # get a list of the filenames
    filenames = np.array(list(data.keys()))

    # get a list of just the features
    feat = np.array(list(data.values()))

    # reshape so that there are n samples of 4096 vectors
    feat = feat.reshape(-1,4096)
#     print(feat)
    print("len of data is {}".format(len(data)))

    # reduce the amount of dimensions in the feature vector
    pca = PCA(n_components=100, random_state=22) #n_components = 100, random_state=22
    pca.fit(feat)
    x = pca.transform(feat)

    # cluster feature vectors

    visual_clustering = MeanShift(bandwidth=40,bin_seeding = True,cluster_all = True)  #n_components = 100, bandwidth=40 -> 47개
    visual_clustering.fit(x)
    visual_cluster_label = visual_clustering.labels_
    # print(geo_cluster_label)
    labels_unique = np.unique(visual_cluster_label)
    #print(labels_unique)
    vn_cluster = len(labels_unique)

    print("number of {}th visual-clusters: {}".format(i, vn_cluster))

    #remove 주석처리
    # holds the cluster id and the images { id: [images] }
    groups = {}
    for file, cluster in zip(filenames, visual_cluster_label):
```

```
    vn_cluster = len(labels_unique)

    print("number of {}th visual-clusters: {}".format(i, vn_cluster))

    #remove 주석처리
    # holds the cluster id and the images { id: [images] }
    groups = {}
    for file, cluster in zip(filenames, visual_cluster_label):
        if cluster not in groups.keys():
            groups[cluster] = []
            groups[cluster].append(file)
        else:
            groups[cluster].append(file)

    for i in groups.keys():
      print(len(list(groups[i])))

    count_num = 0 #count number of all contents of geo_clusters

    for j in groups.keys():
      folder_name = "visual_cluster"+str(j)
      print("folder_name is", folder_name)
      folder_path = os.path.join(geo_cluster_path, folder_name)
      if(os.path.isdir(folder_path) == False):
          os.mkdir(folder_path)                ## 제출 시 주석 해제  # 폴더 생성 코드
      jpglist = []
      jpglist = list(groups[j])
      print("num_of_photos is {}".format(len(jpglist)))
      for jpg in jpglist:
        src_dir = os.path.join(geo_cluster_path, jpg)
        dst_dir = os.path.join(geo_cluster_path, folder_name, jpg)
        shutil.copyfile(src_dir, dst_dir) #shutil.copyfile(src_dir, dst_dir) # 해당하는 포토 파일 명 찾아 폴더 안에 이동
print("all done")
```

0-12 geo_cluster들은 이미지가 100개 이상을 가지고 있습니다. 각 geo_cluster안의 image들을 images list에 저장하고 각 image에 대해 feature vector를 뽑은 다음 "data" dictionary에 key값을 image, value 값을 feature vector로 저장합니다. 그런 다음, 4096차원의 n_samples(= geo_cluster내의 image개수)에 대해 n_components = 100으로 하여 PCA를 진행해 차원을 축소 시킨 후, bandwidth=40으로 meanshift를 통해 visual clustering을 해주고 geo_cluster folder안에 visual cluster folder를 만들어 저장하였습니다.

13-31 geo_cluster들은 이미지가 100개 미만, 40개 이상 가지고 있어, n_components = 40, bandwidth = 40으로 위와 같이 visual clustering을 하여 결과를 저장하였습니다.

32-58 geo_cluster들은 40개 미만, 10개 이상의 이미지를 갖고 있어, n_components = 10, bandwidth = 70으로 위와 같이 visual clustering을 하여 결과를 저장하였습니다.

58-126 geo_cluster들은 10개 미만의 이미지를 갖고 있어, n_components = (이미지개수//3+1), bandwidth = 50으로 위와 같이 visual clustering을 하여 결과를 저장하였습니다.

아래 코드는, empirical하게 n_components, bandwidth값을 정했기에, visual clustering의 결과를 본 후, 결과가 좋지 않은 경우, 만들어진 visual cluster folder를 삭제하는 코드입니다.

remove folders = wrong result of visual clustering

```
: # # # # print(array126)
  # # # # print(n_cluster)
  # import os, glob

  for i in range(59,127): #remove wrong result folder of visual clustering
      geo_cluster_path = os.path.join(photo_path, 'geo_cluster'+ str(i))#'/content/dri
      print("geo_cluster_path is {}".format(geo_cluster_path))
      # # change the working directory to the path where the images are located
      os.chdir(geo_cluster_path)
      subfolderlist = list(filter(os.path.isdir, glob.glob('*')))
      print(subfolderlist)
      for j in subfolderlist:
          removepath = os.path.join(geo_cluster_path, j)
          print(removepath)
          shutil.rmtree(removepath)
```

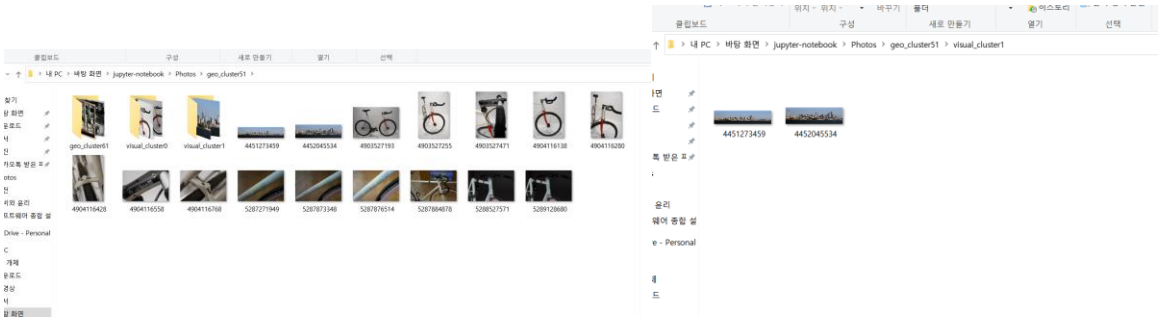
Visual clustering 결과 분석

결과를 살펴보면, 다음과 같았습니다.

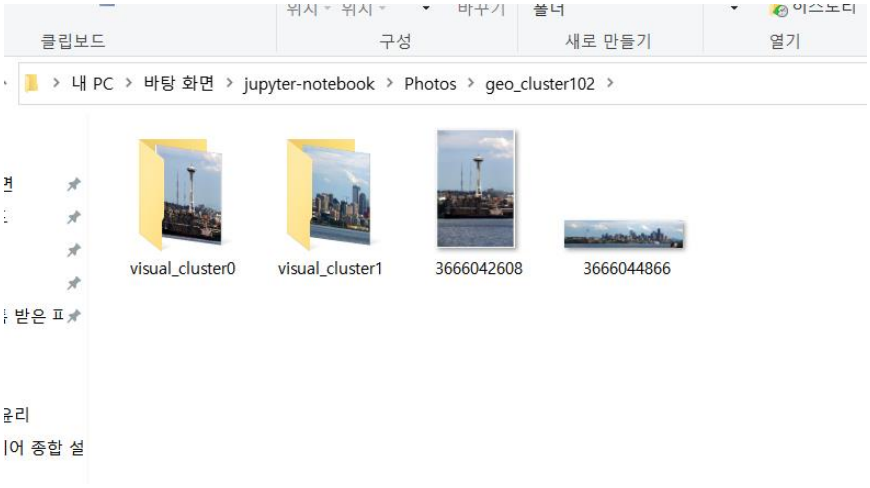
1) 다른 landmark인데 같은 geo_cluster였던 사진들이 visual clustering을 통해 잘 분리된 경우

2) 같은 landmark인데 다른 visual_cluster에 속하는 경우

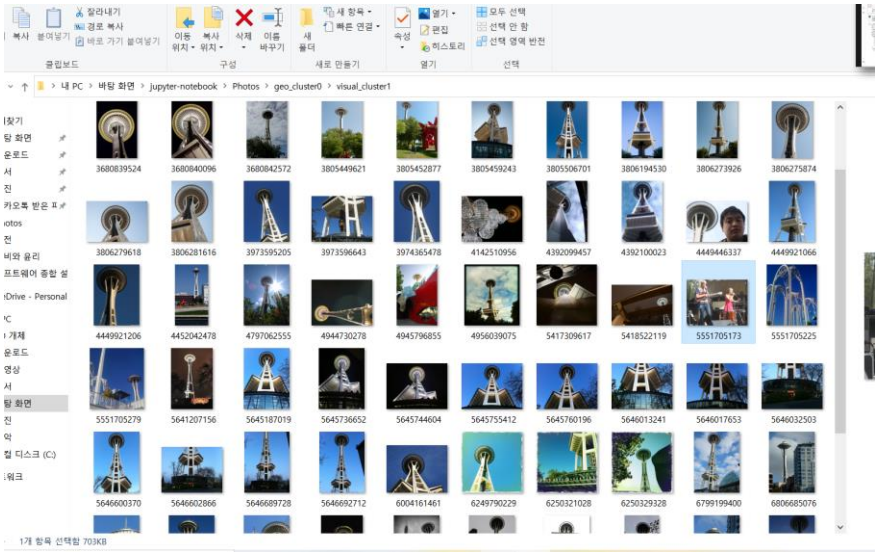
3) Outlier가 landmark와 같은 visual cluster에 속하는 경우



1) Outlier와 landmark가 같은 geo_cluster51에 존재하였는 데, visual_cluster로 잘 분리된 모습입니다.



2) 같은 landmark(space needle)인데 다른 visual cluster에 속하는 경우입니다.



3) outlier와 landmark가 같은 visual cluster에 속하는 경우입니다.

2,3번의 경우뿐만 아니라 geo-clustering에서 같은 landmark가 다른 geo_cluster에 속하는 문제가 여전히 visual clustering으로 해결되지 않는 문제가 있었습니다.

따라서, bandwidth와 n_components를 조정하는 것만으로는 한계가 있다고 생각하여 tag information을 통해 geo-clustering, visual-clustering의 한계를 해결하고자 하였습니다.

Step3. Using Tag information & additional input (landmark dictionary)

Geo-clustering, visual-clustering 후에도 여전히 다음과 같은 문제가 있어 tag 정보를 사용하여 해결하려고 하였습니다.

1) 같은 landmark 이지만 geo_cluster 가 다름
2) 같은 landmark, geo_cluster 이지만 visual_cluster 가 다름
3) 다른 landmark 이지만 같은 visual_cluster
4) Outlier 와 landmark 가 같은 visual_cluster 에 속함
5) Landmark 와 outlier 구별

Landmark에 대한 추가적인 input없이는 outlier와 landmark를 구별하기 어렵다고 생각했기에, 인터넷에서 seattle의 landmark list를 찾고 landmark 각각이 dictionary의 key값인, 그리고 그 landmark가 가질 수 있는 해시태그 list를 dictionary의 value값으로 하는 landmark dictionary를 만들었습니다.

그런 다음, 각 visual cluster에 속하는 image들의 tag 값들을 중복 없이(set) list로 구해, dictionary의 value값이 list 안에 있다면, 해당 visual cluster안의 image들의 tag list에 dictionary의 value값이 있는 지 확인하고 있다면 해당 image들을 landmark folder를 생성하고 그 안의 해당 landmark 이름의 폴더안으로 복사해주었습니다. 이럴 경우, 단점으로 제가 만든 landmark dictionary의 value값 이외의 해시태그를 가진 landmark image인 경우, landmark folder로 복사되지 않는 문제가 있어, 한 visual cluster의 tag list에 dictionary value값이 있다면, 그 visual cluster에 속하는 모든 image를 landmark folder로 복사하는 방식을 먼저 시도했으나, 이럴 경우, visual clustering이 제대로 되지 않았으면(3,4번의 경우), outlier도 같이 landmark folder로 복사되는 문제가 있어, image tag 별로 한 번 더 걸러내는 과정을 추가하였습니다.

코드는 다음과 같습니다.

Tag Matching

read csv file

In [87]:
```
os.chdir(firstposition)
csv_path = os.path.join(os.getcwd(),'Tag (1).csv')
tag_data = pd.read_csv(csv_path, names = ['photo_ID','num','tag'])
```

In [88]:
```
def preprocessing(tag): #영어, 숫자만 남기기
    tag = re.sub('[^a-zA-Z0-9]',"",tag)
    return tag
```

In [89]:
```
#tag_data['tag'].map(lambda x: print(type(x)))
print(tag_data.loc[103])
```

```
photo_ID              6248446723
num                            5
tag         uploaded:by=instagram
Name: 103, dtype: object
```

In [90]:
```
# 모두 소문자로 변경
tag_data['tag'] = tag_data['tag'].str.lower()

# 영어, 숫자만 남기기
tag_data['tag'] = tag_data['tag'].map(lambda x: (preprocessing(str(x))))
```

In [91]:
```
print(tag_data.loc[103])
```

먼저, tag.csv file을 읽어온 다음, 영어, 숫자만 남게 전처리하고 모두 영어의 경우 모두 소문자로 변경합니다.

In [89]:
```
#tag_data['tag'].map(lambda x: print(type(x)))
print(tag_data.loc[103])
```

```
photo_ID              6248446723
num                            5
tag         uploaded:by=instagram
Name: 103, dtype: object
```

In [90]:
```
# 모두 소문자로 변경
tag_data['tag'] = tag_data['tag'].str.lower()

# 영어, 숫자만 남기기
tag_data['tag'] = tag_data['tag'].map(lambda x: (preprocessing(str(x))))
```

In [91]:
```
print(tag_data.loc[103])
```

```
photo_ID              6248446723
num                            5
tag         uploadedbyinstagram
Name: 103, dtype: object
```

그러면, 다음과 같이 ':'은 제거가 됩니다.

tag에서 가장 빈도수 높은 것 출력

- park, westseattle, bellevue, lake (이런 거 말고 다른 몇개 단어 stopwords로 정의하기)

[96]:
```
tagwords = tag_data['tag'].value_counts()
# print(type(tagwords))
# print(tagwords.head(50))

#convert value_counts() output into a data frame
df_tagwords = tag_data['tag'].value_counts().reset_index()
df_tagwords.columns = ['word', 'word_count']
print(df_tagwords)
```

```
             word  word_count
0         seattle        7564
1      washington        2958
2             usa         785
3              wa         718
4      convention         667
...           ...         ...
11097       m33gs           1
11098  cloudycloud           1
11099     raincity           1
11100          dii           1
11101    sigma50mm           1

[11102 rows x 2 columns]
```

tag에서 가장 빈도수 높은 것들을 출력하여 확인하고

define stopwords

```
97]: #define stopwords
stopwords = []
not_stopwords = ['park', 'art','lakeunion','bellevue'] #'seattle', 'washington'
for i in range(50):
    word = df_tagwords['word'].values[i]
    if word not in not_stopwords:
        stopwords.append(word)
print(len(stopwords))
print(stopwords)
# result = []
# for word in word_tokens:
#     if word not in stop_words:
#         result.append(word)
```

```
46
['seattle', 'washington', 'usa', 'wa', 'convention', '2009', 'square', 'redmond', '2
012', 'iphoneography', 'uploadedbyinstagram', 'squareformat', 'instagramapp', 'rf',
'unitedstates', '2010', 'con', 'furry', 'rf2012', 'rainfurrest', 'eyefi', 'kinner',
'concerts', 'ratcityrollergirls', '2004', 'rcrg', 'rollerderby', 'westseattle', 'swa
rali', 'iphone', 'downtown', 'august', 'concert', '', '180550mm', 'june', 'nikon',
'cameraphone', 'canon', 'music', 'live', 'show', 'wedding', 'travel', 'dylandiana',
'water']
```

상위 50개의 빈도수 높은 단어 중, park, art, lakeunion, bellevue와 같이 중요한 단어를 제외한 46개의 단어를 stopwords(불용어)로 지정합니다.

tag에서 아까 정의한 46개의 stopwords 제거하기

```
: def removingSW(tag):
    if tag in stopwords:
        tag = ""
    return tag
tag_data['tag'] = tag_data['tag'].map(lambda x: (removingSW(str(x))))
```

dictionary(photo_ID -> array of tag values)

```
: #dictionary: key(photoID) --> value(array of tag values)
photoID_to_tags = {}
# for i in range(0,len(photo_IDs)):
#   index_to_photoID[i] = photo_IDs[i]

for idx, i in enumerate(tag_photo_IDs):
  condition = (tag_data['photo_ID']  == i)
  photoID_to_tags[i] = tag_data[condition].tag.values

# #   tags_wo_stopwords = []
#   for i in range(len(tag_data[condition].tag.values)): #remove stopwords
#     word = tag_data[condition].tag.values[i]
#     if word not in stopwords:
#         tags_wo_stopwords.append(word)
#   photoID_to_tags[i] = tags_wo_stopwords

print(photoID_to_tags)
```

tag값들에서 46개의 stopwords를 제거하고 Photo_ID로 tag를 접근할 수 있도록 photoID_to_tags dictionary를 생성합니다.

Seattle landmark dictionary 만들기

```
56]: # space needle, safecofield, pikeplacemarket, seattleaquarium,
     # centurylinkfield(lumen field), sam(seattleartsmuseum), volunteerpark, ballardlocks,
     # virginiav, newcastlebeachpark, spacemuseum...
     # urbanlightstudios

     dict_landmarks = {'space needle': ['spaceneedle', 'space', 'needle'], 'safeco field': ['safecofield', 'safeco'], 'pike place market': ['p
     'centurylink field':['centurylinkfield','centurylink','centurylinkstadium', 'stadium', 'stadiumns','instagramvenuenamecenturylinkfield'],
                     'Monorail':['monorail'], 'seattleaquarium': ['seattleaquarium']}

     # print(dict_landmarks)

     # print(dict_landmarks['space needle'])

     keys = list(dict_landmarks.keys())
     print(keys)
     # for idx, values in enumerate(dict_landmarks.values()):
     # #     print(values)
     # #     print(idx, values)
     #     print("key is ", keys[idx])
     #     for j in range(len(values)):
     #         print(values[j])
     # Photos folder안에 landmarks folder만들기
     os.chdir(firstposition)
     print(os.getcwd())
     landmark_path = os.path.join(os.getcwd(),'Photos','landmarks')
     if(os.path.isdir(landmark_path) == False):
             os.mkdir(landmark_path)

     for key in keys:
         eachlandmark_path = os.path.join(landmark_path, key)
         if(os.path.isdir(eachlandmark_path) == False):
             os.mkdir(eachlandmark_path)
```

Seattle landmark dictionary를 만들고, photos folder안에 landmark folder를 생성하고 dictionary의 key값이 이름인 folder들을 생성했습니다.

landmark 찾고 폴더에 모으고 폴더 이름 그에 맞게... + 각 폴더 위치를 photos안에 landmarks 폴더 안으로...?

```
07]: # 각 geo_cluster folder안의 visual_cluster folders의 tag구해서 landmark dictionary의 values에 있는 지(완전 같지 않아도 유사한지) 해줌,,;
     # 만약 해당한다면 그 tag를 갖는 사진이 속하는 visual_cluster를(아니면 사진만,,?)
         # 그 dictionary key값을 folder name으로 하는 folder가 photos에 없을 경우 만들기
         # 그 landmarks folder로 옮기기

     #visual cluster의 tag set구하기
     for i in range(0,127): #Geoclusters
         geo_cluster_path = os.path.join(photo_path, 'geo_cluster'+ str(i))
         print("geo_cluster_path is {}".format(geo_cluster_path))
         # # change the working directory to the path where the images are located
         os.chdir(geo_cluster_path)
         subfolderlist = list(filter(os.path.isdir, glob.glob('*')))
         #print(subfolderlist)
         for index,visualcluster in enumerate(subfolderlist): #Visual Clusters
             visualclusters = os.path.join(geo_cluster_path, visualcluster)
             globals()['tag_geo{}_vc{}'.format(i,index)] = []
             #print(visualclusters)
             for idx, j in enumerate(os.listdir(visualclusters)):
                 if j.endswith(".jpg"):
                     file = j.rstrip(".jpg")
                     if(int(file) in photoID_to_tags): #check ID is key of photoID_to_tags
                         globals()['tag_geo{}_vc{}'.format(i,index)].extend(photoID_to_tags[int(file)])
             print("tag_geo{}_vc{} is {}".format(i,index, list(set(globals()['tag_geo{}_vc{}'.format(i,index)])) ))
```

```
C:₩Users₩dbw21₩Desktop₩jupyter-notebook
```

각 visual cluster의 tag들을 중복없이 list로 구하는 코드입니다.

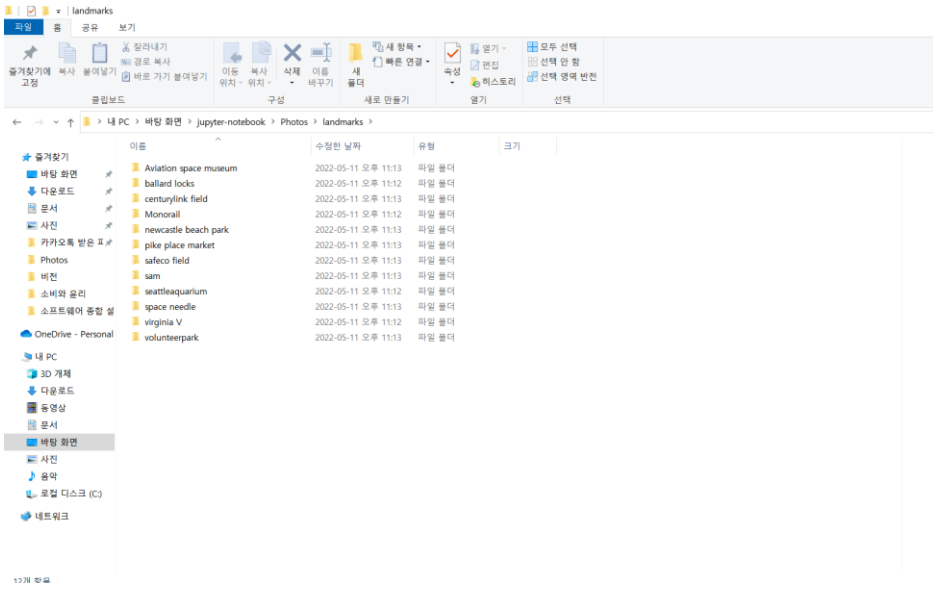
Tag_geo{geo_cluster_index}_vc{visual_cluster_index}에 저장이 됩니다.

```
158]: # 각 geo_cluster folder안의 visual_cluster folders의 tag구해서 landmark dictionary의 values에 있는 지(완전 같지 않아도 유사한지) 해줌,,?
      # 만약 해당한다면 그 tag를 갖는 사진이 속하는 visual_cluster를(아니면 사진만,,?)
          # 그 dictionary key값을 folder name으로 하는 folder가 photos에 없을 경우 만들기
          # 그 landmarks folder로 옮기기

      from distutils.dir_util import copy_tree

      #한 visual cluster에서 landmark tag 있으면 전부를 옮기니까 outlier도 같이 옴.
      #그러므로 한 visual cluster에서 landmark tag있으면 그 visual cluster내에 있는
      #각 이미지들(for loop)에 대해 tag구하고 landmark tag있는 이미지만 옮기기

      for geo_index in range(0,127): #Geoclusters #0-127
          geo_cluster_path = os.path.join(photo_path, 'geo_cluster'+ str(geo_index))
          print("geo_cluster_path is {}".format(geo_cluster_path))
          # # change the working directory to the path where the images are located
          os.chdir(geo_cluster_path)
          subfolderlist = list(filter(os.path.isdir, glob.glob('*')))
          #print(subfolderlist)
      #     os.chdir(photo_path)
          for visual_index,visualcluster in enumerate(subfolderlist): #Visual Clusters
              for idx, values in enumerate(dict_landmarks.values()):
                  for k in range(len(values)):
                      if values[k] in globals()['tag_geo{}_vc{}'.format(geo_index,visual_index)]:
                          for img in os.listdir(os.path.join(geo_cluster_path, visualcluster)):
                              ID = img.rstrip(".jpg")
                              if(int(ID) in photoID_to_tags): #check ID is key of photoID_to_tags
                                  if values[k] in photoID_to_tags[int(ID)]:
                                      src_dir = os.path.join(geo_cluster_path, visualcluster,img)
                                      dst_dir = os.path.join(photo_path,"landmarks",keys[idx], img)
                                      shutil.copyfile(src_dir, dst_dir)  #이미 존재하는 폴더에 복사
      print("All done")
```

그런 다음, 각 visual cluster에 속하는 image들의 tag 값들을 중복 없이(set) list로 구해, dictionary의 value값이 list 안에 있다면, 해당 visual cluster안의 image들의 tag list에 dictionary의 value값이 있는 지 확인하고 있다면 해당 image들을 landmark folder를 생성하고 그 안의 해당 landmark 이름의 폴더안으로 복사해주었습니다.

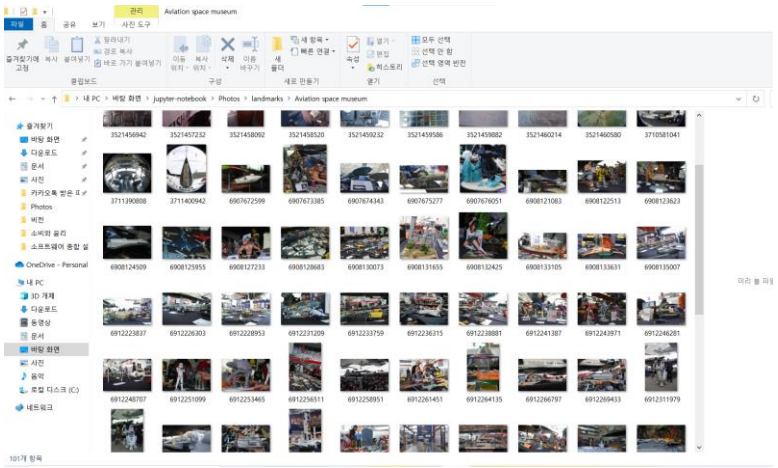
최종 결과

최종 결과는 다음과 같습니다.

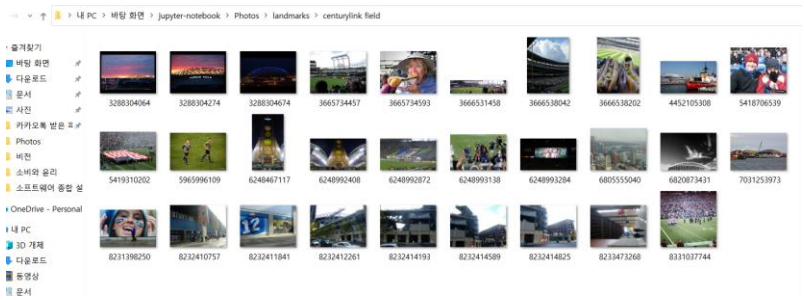


Photos folder 안에 각 landmark folder 가 생성되고 각 landmark folder 안에 해당하는 이미지가 들어 있습니다.

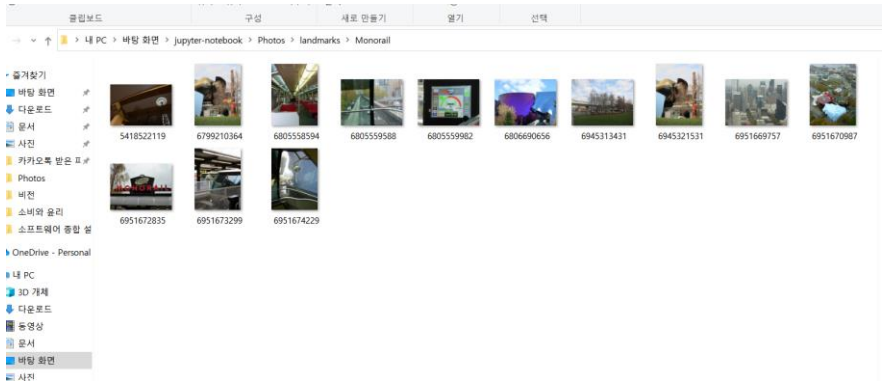
- Aviation space museum



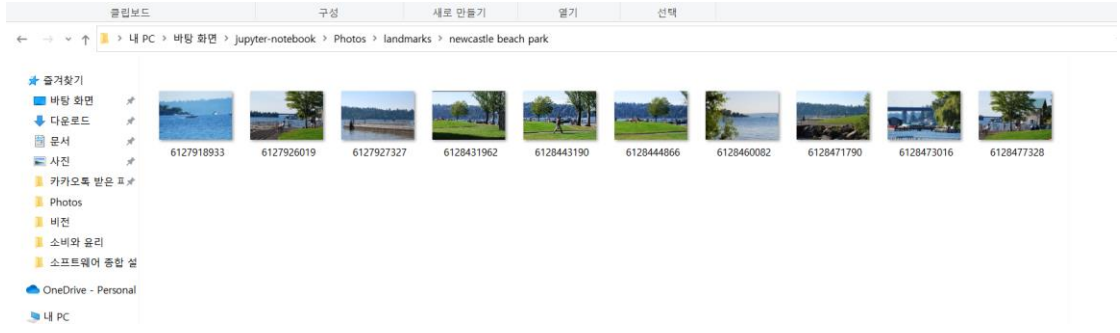
- Centurylink field



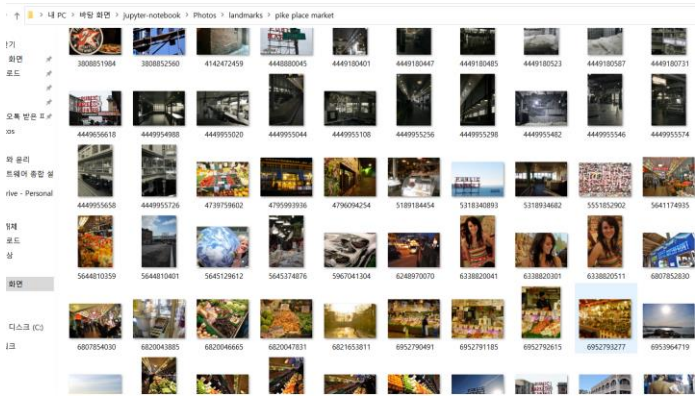
- Monorail



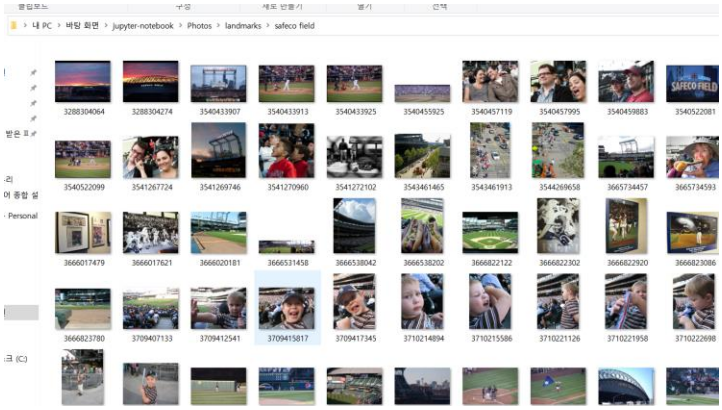
- Newcastle beach park



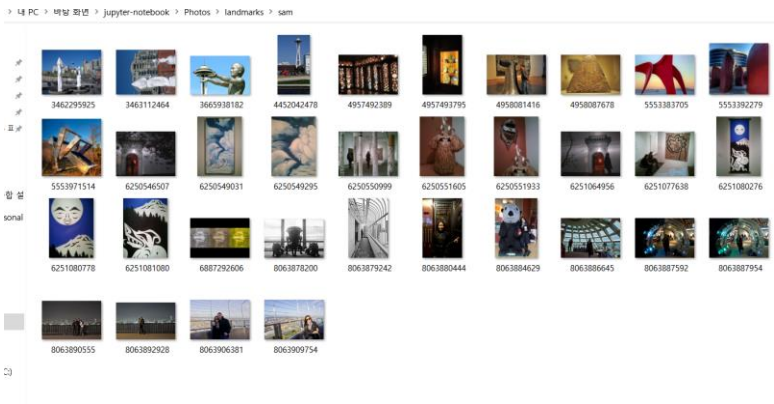
- Pike place market



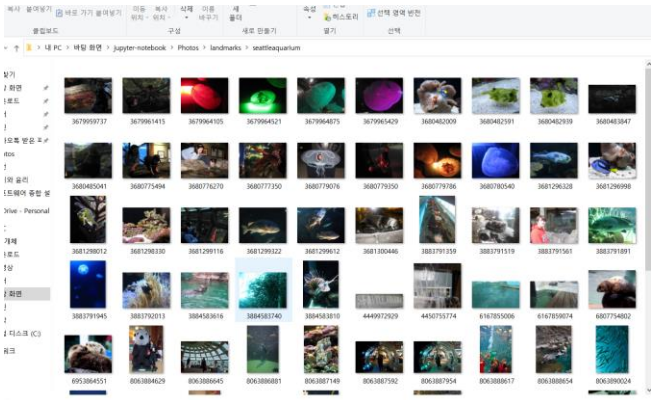
- Safeco field



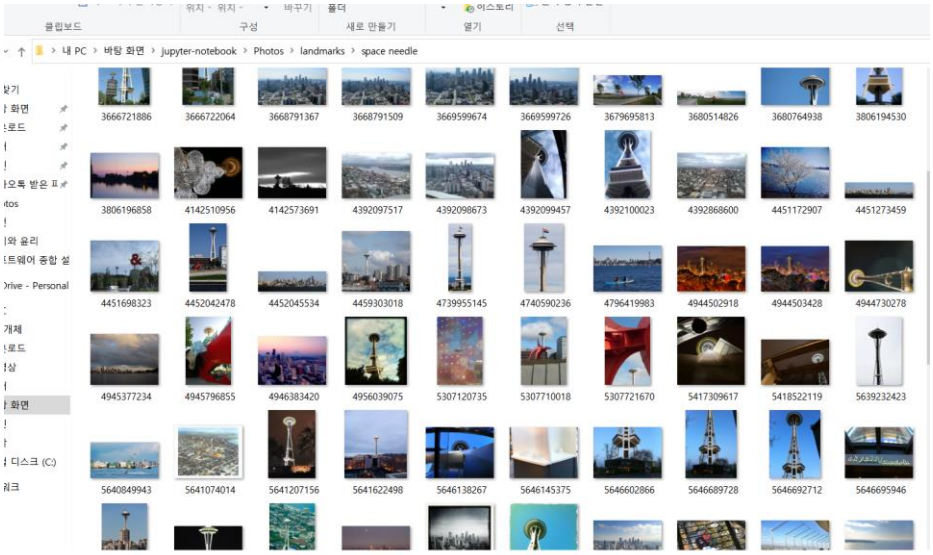
- Sam = seattle art museum



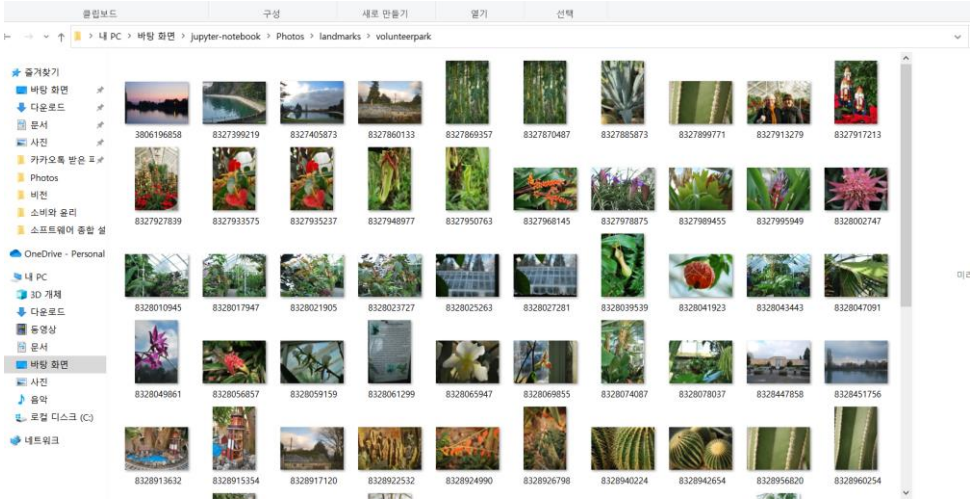
- Seattle aquarium



- Space needle



- Volunteerpark



Photos안의 이미지가 전체 이미지가 아닌 부분 이미지라 virginia V, ballard locks의 경우 tag information은 있었으나 해당 ID의 이미지가 없어 빈 폴더로 나타났습니다.