**국내 여행, 어디로 떠날까? –여행계획 보조 프로그램**

8조 여행보내조

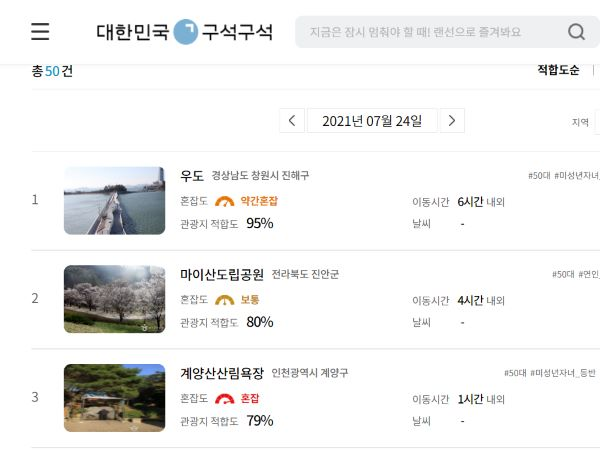
문주원 이하경 염다빈 민황선 송은채

1. 문제 정의

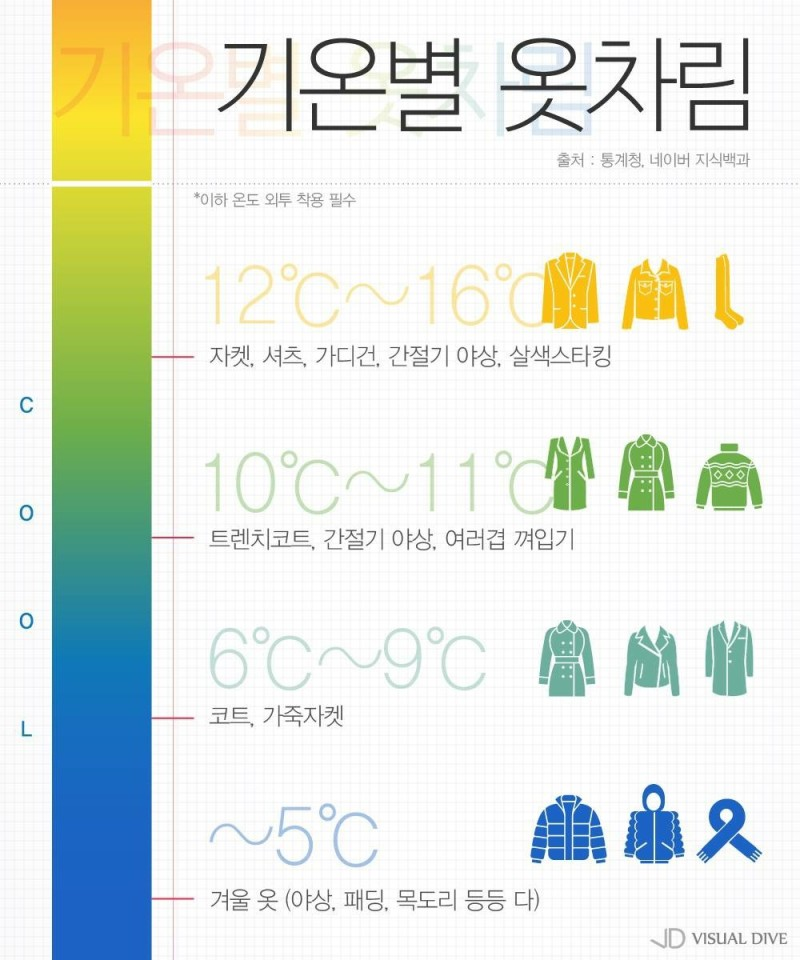
코로나19의 장기 유행으로 사람들의 국내 여행에 대한 관심이 커져오고 있다. 그러나 막상 여행 계획을 세우려다 보면 어디로 떠날지, 무엇을 해야할지 등 다양한 고민을 마주하게 된다. 이 때 활용할 수 있는 정보를 제공하는 서비스는 다수 존재하지만, 실제 여행을 계획하는 사람들에게 유용하게 쓰이고 있지는 않은 실정이다.



위 이미지는 현재 기상청 날씨마루 사이트에서 제공하는 관광코스별 상세날씨 조회 서비스 실행 모습이다. 여행코스를 선택하면, 그 시점(접속 일시) 또는 다음날 9시에 출발하는 일정으로 고정되어 코스가 안내 되며 여행 일시를 선택할 수 없다. 또한, 1~2일 동안 4개 이상의 도시를 순회하는 무리한 여행코스를 추천하는 등, 실제 여행 계획에 참고하기 어렵다는 문제점이 있다.



위 이미지는 한국관광공사에서 제공했었던 대한민국 구석구석 사이트에서 제공하는 여행지 추천 베타서비스 실행 모습이다. 그러나 출발지로부터 이동시간이 6시간 내외로 걸리는 멀리 떨어진 지역의 적합도가 높게 나타나거나, 날씨 정보는 제공되지 않는 등의 문제점이 있었다.



위 이미지는 기존에 널리 퍼져있는 기온별 옷차림 추천 정보이다. 이 표는 기온만을 기준으로 옷차림을 추천해주므로, 바람이 강한 날, 일교차가 큰 날, 강수가 있는 날 등에는 세부적으로 적용하기 힘든 단점이 있다. 또한, 먼 시일에 여행을 계획중인 경우 날씨를 예측하기 힘들기 때문에 여행 계획 시에 옷차림 결정에 어려움이 존재한다.

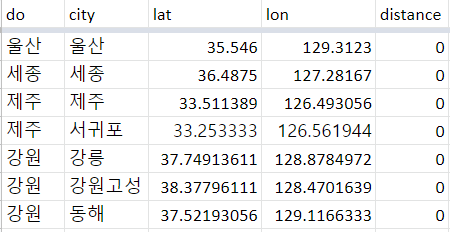
본 프로젝트에서는 위 서비스와 정보들의 단점을 서로 보완할 수 있는 여행 계획 보조 프로그램을 기획하였다. 여행 날짜, 출발지, 선호 여행 테마를 사용자가 직접 선택할 수 있게 했으며, 일기예보 OpenAPI 및 지역별 과거 30년 평균 기후데이터를 활용해 여행일의 날씨를 보다 정확히 예측하고자 했다. 또한 이를 활용하여 체감기온, 바람, 일교차등을 모두 고려한 추천 옷차림 정보도 제공하고자 하였다.

1. 분석 데이터 설명 및 수집 방법
2. 대한민국 도시별 위·경도 데이터

원본에는 시/도와 시/군/구 분류, 위도 및 경도 데이터가 존재한다. 편리한 프로그램 실행을 위하여 다음과 같은 데이터 가공 과정을 거친다.

1. 해당 지역이 특별시/광역시 등에 해당하는 경우 ‘city’ 열은 ‘구’ 이름이 저장되어 있다. 구 수준의 세분화된 정보는 불필요하다고 판단했기 때문에 이 경우 ‘city’에는 ‘do’와 동일하게 ‘시’ 이름을 저장하였다.
2. 원본 파일의 ‘city’ 열에는 ‘XX시’, ‘XX군’ 형태로 데이터가 저장되어 있으나, 편의상 ‘시’와 ‘군’ 텍스트를 엑셀 내장 함수(left, len)를 이용해 제거하였다.
3. 경기도 광주와 광주광역시, 경상남도 고성과 강원도 고성의 ‘city’ 열 값 중복을 피하기 위해 각각을 ‘경기광주’, ‘광주’, ‘경상고성’, ‘강원고성’ 으로 저장한다. 엑셀 내장 함수(if, ifs, iserror, and)를 사용해 값을 치환해 저장되도록 한다.

다음은 가공 완료된 데이터셋의 일부이다.

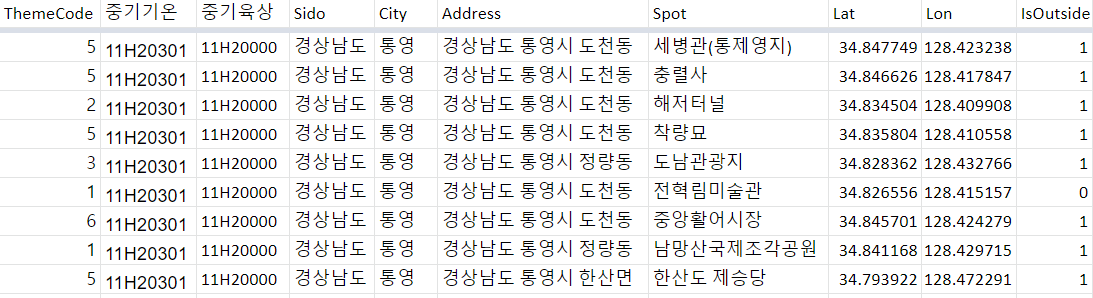


1. 관광코스 지점 정보

기상청 날씨마루의 관광코스별 상세날씨 조회 서비스에서 제공하는 ‘관광코스별 관광지 상세날씨 조회 지점 정보’ csv 파일을 사용한다. 원본 데이터에 포함된 정보는 총 11종이며, 본 프로젝트에서는 그 중에서 테마분류, 지역 아이디, 관광지명, 경도, 위도, 실내구분 등 총 6개 데이터를 다음과 같이 가공해 활용한다.

1. 편의상 열의 이름을 영어로 변경한다.
2. ‘관광지명’은 ‘(지역명)관광지명’의 형태로 저장되어 있는데, 엑셀 내장 함수(mid, find, len)를 활용해 새로운 city와 spot 열로 텍스트를 분리해 저장한다.
3. 지역아이디는 행정동, 법정동코드를 기반으로 한다. 코드 정보와 엑셀 내장 함수(vlookup, find, mid)를 활용해 지역아이디를 ‘시/도 시/군/구 읍/면/동’ 형태의 주소로 변환하고 ‘Address’ 열을 추가하여 저장한다.
4. 실내구분 열의 데이터는 ‘실내’와 ‘실외’로 저장되어 있는데, 이를 0과 1로 변환해 ‘isOutside’ 열에 저장한다.
5. 지역별 중기예보 API 파싱을 위해 ‘중기기온’과 ‘중기육상’ 예보구역 코드를 추가한다. 기상청에서 제공하는 ‘날씨 데이터 지점명 정보’ 문서에 첨부된 코드 표와 엑셀 내장 함수(vlookup)를 이용해 해당 관광지가 속한 구역의 코드를 새 열에 저장한다. 관광지가 위치한 도시가 중기기온예보구역에 해당하지 않는 경우, 파이썬 geopy의 distance.distance 모듈을 이용해 각 구역과의 직선거리를 계산하고, 가장 가까운 구역의 코드를 입력한다.

다음은 가공 완료된 데이터셋의 일부이다.

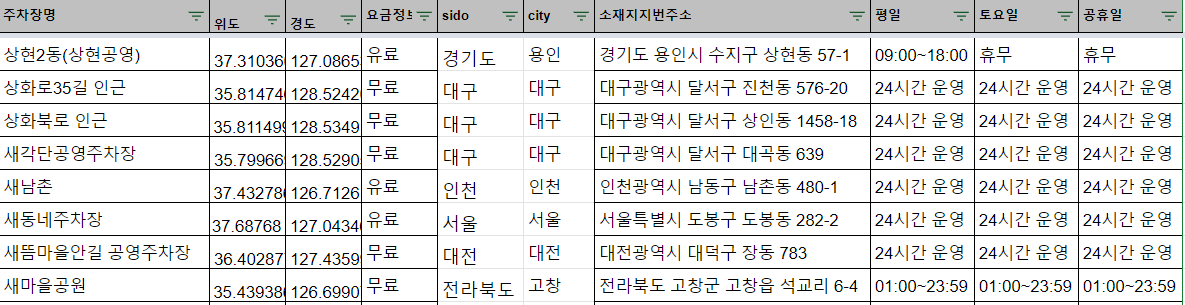


1. 전국 주차장 정보

공공데이터포털에서 제공하는 국토교통부 ‘전국주차정정보표준데이터’ csv 파일을 사용한다. 원본 데이터에 포함된 정보는 총 31종이며, 본 프로젝트에서는 그 중에서 주차장명, 위도, 경도, 요금정보, 소재지지번주소, 평일/토요일/공휴일 운영시작(종료)시각 등 총 11개 데이터를 다음과 같이 가공해 활용한다.

1. 지번주소 데이터에 엑셀 내장 함수(left, mid, find 등)를 활용해, 각 주차장이 위치한 시/도 및 시/군/구 정보를 추출해 새 열을 추가한다.
2. 평일/토요일/공휴일의 운영 시작 및 종료시각 데이터에 엑셀 내장 함수(textjoin)를 이용해, ‘00:00~00:00’ 형태로 운영시간을 새 열에 저장한다. 운영 시간이 ‘00:00~23:59’인 경우는 엑셀 내장 함수(if)를 사용해 해당 셀에 ‘24시간 운영’이라는 텍스트가 저장되도록 한다.
3. 경기도 광주와 광주광역시, 경상남도 고성과 강원도 고성의 ‘city’ 열 값 중복을 피하기 위해 각각을 ‘경기광주’, ‘광주’, ‘경상고성’, ‘강원고성’ 으로 저장한다. 엑셀 내장 함수(if, ifs, iserror, and)를 사용해 값을 치환해 저장되도록 한다.
4. 지번주소 혹은 위경도 값이 존재하지 않는 행을 엑셀 내 필터 기능을 이용해 삭제한다.

다음은 가공 완료된 데이터셋의 일부이다.



1. 기상청 일기예보 및 과거 30년 평균 기후 데이터

기상청에서 제공하는 단기예보와 중기 예보 API, 과거 30년 평균 기후데이터 자료를 사용한다.

1. 단기예보, 중기예보

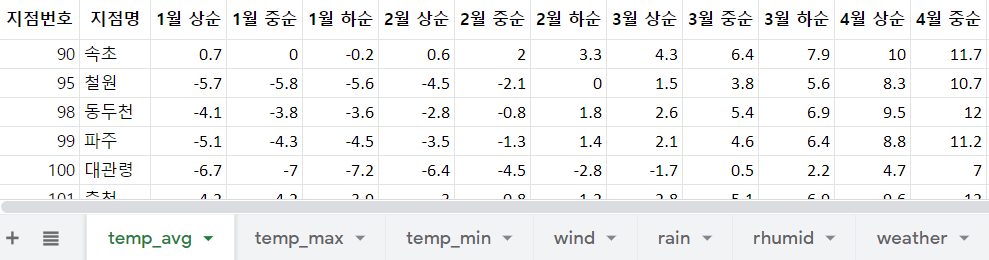
기상청에서 제공하는 openAPI를 파싱해서 사용한다. 이때 필요한 예보 데이터만 남기고 다른 예보는 제거한다. 단기 예보에서는 당일에서 2일 후까지의 예보 데이터로, 1시간 기온, 풍속, 하늘상태, 강수형태, 강수확률, 1시간 강수량, 습도, 1시간 적설량 데이터를 사용한다. 중기 예보는 3일 후부터 7일후까지의 예보 데이터로, 오전/오후 강수확률, 오전/오후 운량, 최고/최저 기온 데이터를 사용한다.

1. 과거 30년 평균 기후 데이터

기상자료개방포털의 기후평년값 (`91~`20) 상세 조회 서비스에서 제공하는 30년(1991~2020) 평균 지점별 순별 평년값 xls 파일을 사용한다.

1. 다운로드할 기후 요소는 평균 기온, 최고 기온, 최저 기온, 풍속, 강수량, 상대습도, 전운량을 선택하였다.
2. 원본 데이터의 경우 하나의 시트에 1월 상순부터 12월 하순까지의 7개 종류 데이터가 한번에 저장되어 있다. 편리를 위해 시트를 변수 종류에 따라 나누어, 파이썬에서 불러올 때 데이터프레임에 깔끔하고 가독성이 좋게 저장되도록 한다.

다음은 가공 완료된 데이터셋의 일부이다.



1. 날씨별 적정 옷차림 데이터

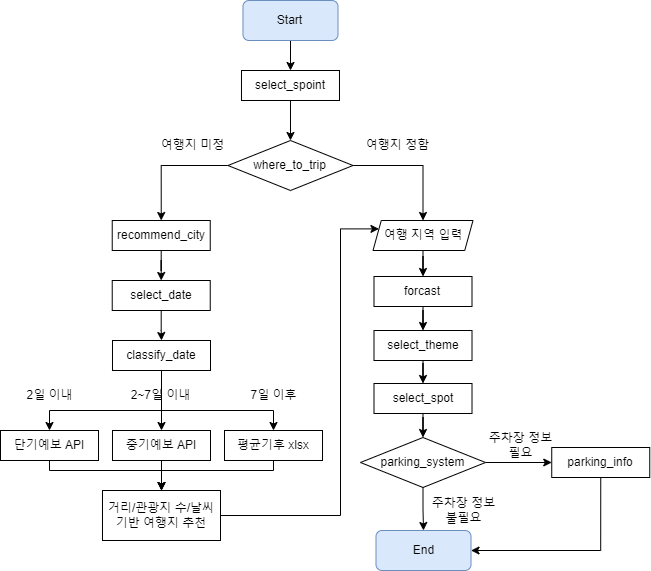
각종 날씨별 옷차림 추천 자료들을 참고하여 자체적으로 제작한 ‘날씨별 옷차림’ xlsx 파일을 사용한다.

1. ‘tmin’과 ‘tmax’ 열을 통해 해당 행의 옷차림이 적합한 기온의 범위를 저장한다.
2. 바람이 강해 체감기온이 낮은 경우 추천할 ‘warmer’ 열을 추가한다.
3. 같은 종류의 옷이라도 소재에 따라 보온 정도가 다르기 때문에, ‘memo’ 열에 추천하는 옷의 소재를 저장한다.

다음은 가공 완료된 데이터셋이다.



1. 프로그램의 기능
2. 여행 도시를 정하지 못한 사용자에게, (1)출발지에서 가깝고, (2)관광스팟이 10개 이상 있는 상위 15개의 도시를 추천한다.
3. 여행 일자를 입력받았을 때, 일기예보가 존재하는 가까운 시일이라면 해당 일자의 일기예보를 이해하기 쉽게 시각화해 보여준다. 여행 도시를 정하지 못했다면, 선택을 돕기 위해 해당 15개 도시의 일기예보(일평균 강수확률 및 전운량) 정보를 제공한다.
4. 일기예보 및 기후 데이터를 이용해 여행 일자에 적절한 옷차림을 추천한다.
5. 여행 테마를 선택하면 각 테마의 여행지를 지도로 보여주며, 사용자가 방문할 관광지를 선택할 수 있다.
6. 최종 선택한 여행지와 그 인근 주차장 정보를 제공한다.
7. 구현 프로젝트만의 강점
   1. 기존에 존재하던 서비스와 달리, 본 프로젝트는 사용자 맞춤 서비스이기 때문에 여행 가는 날짜, 출발지, 선호 테마 등을 선택할 수 있다. 이를 통해 사용자에게 보다 실용적인 정보를 제공하여 여행 계획 단계에서 많은 도움을 줄 수 있는 프로그램이다.
   2. 관광지 인근의 주차장 유무, 운영시간, 운영 형태 정보 등 다양한 정보를 제공하여 사용자가 더욱 편리하게 여행 계획을 세우는데 도움을 줄 수 있다. 선택한 관광지를 지도에 표시하고, 테마 선호도 순서를 부여하고 다른 색으로 표시하였기 때문에 사용자가 관광지에 대한 정보를 한 눈에 알아 볼 수 있어 사용자 선택에 도움을 준다.
   3. 기존의 서비스는 날씨 정보만 제공하였지만, 본 프로그램은 날씨에 따른 추천 옷차림 정보 또한 같이 제공한다. 추천 옷차림 서비스 역시 기존의 서비스들과 달리 기온, 강수 유무, 바람 등을 고려하여 자세한 옷차림을 추천하기 때문에 기온만 고려한 추천 옷차림보다 더욱 적절한 옷차림을 선택할 수 있다.
8. 프로그램 구조 및 실행 방법
9. 플로우 차트



메인 프로그램을 실행시키면, 출발지를 입력 받는 ‘select\_spoint’ 함수가 실행된다. 이후, ‘where\_to\_trip’ 함수로 여행지 선택 여부를 입력 받고, 여행지가 미정 상태라면 ‘recommend\_city’ 함수가 실행된다. ‘select\_date’ 함수로 여행갈 날짜를 입력받고, ‘classify\_date’로 프로그램을 실행시킨 시점으로부터 얼마나 먼 미래인지 판단하여, 2일 이내라면 단기예보 API를, 2~7일 이내라면 중기예보 API를, 7일 이후라면 30년 평균 기후 데이터를 이용한다. 이후 출발지에서 가깝고, 관광할 거리가 많으며 날씨가 좋은 여행지를 추천하고, 사용자는 원하는 여행지역을 선택한다. ‘forecast’ 함수로 여행가는 날의 날씨와 추천 옷차림 정보를 제공한다. 이후, ‘select\_theme’ 함수로 선호하는 테마를 최대 3개까지 입력받고, 각 테마별 관광지를 지도와 함께 출력한다. ‘select\_spot’ 함수에서는 선호 테마에 속한 관광지 중 가고 싶은 관광지를 선택한다. ‘parking\_system’에서 주차장 정보 필요 여부를 입력 받고, 필요하다면 선택한 관광지 인근의 주차장 정보를 지도와 함께 출력한다.

1. 실행 방법
2. 폰트 설치 후 런타임 재시작

| # 폰트 설치  !sudo apt-get install -y fonts-nanum !sudo fc-cache -fv !rm ~/.cache/matplotlib -rf |
| --- |

1. ‘사전 작업’ 속 코드를 모두 실행
   1. 구글 드라이브 연동, 라이브러리 설치/업데이트

| # 구글 드라이브 연동 from google.colab import drive drive.mount('/content/gdrive') directory\_path = '/content/gdrive/My Drive/Colab Notebooks/2022-1 파데분 팀프로젝트/data\_file/' |
| --- |

| # 라이브러리 설치 !pip install folium # 지도 시각화 !pip install haversine # 거리 계산 !pip install tabulate # pretty table |
| --- |

* 1. 라이브러리/모듈 import 작업

| # import import pandas as pd from pandas.compat.numpy import np\_datetime64\_compat from pandas import Series, DataFrame  import numpy as np import time from datetime import datetime, timedelta import math import csv from tabulate import tabulate  # 거리 계산, 지도 시각화 from geopy import distance # distance.distance([lat,lon]).km from haversine import haversine  import folium from folium import plugins ## 지도 출력할 때 필요한 라이브러리 추가(은채) from folium.plugins import MarkerCluster from folium.plugins import MiniMap  # 일기예보 api import requests import json from pytz import timezone from termcolor import colored import matplotlib.pyplot as plt |
| --- |

* 1. Dataset 불러오기

| # 도시별 위경도 데이터 latlon = pd.read\_excel(directory\_path + 'korea\_administrative\_division\_latitude\_longitude.xlsx') latlon\_np = pd.DataFrame.to\_numpy(latlon) city\_list = latlon\_np[:,1] # 전체 도시명 목록 리스트로 저장  # 관광코스 (+위경도) 데이터 spot = pd.read\_excel(directory\_path + '관광코스\_지점정보.xlsx') spot\_np = pd.DataFrame.to\_numpy(spot)  # 각 도시별 관광스팟 개수 카운트 후 저장 for i, city in enumerate(spot.City.value\_counts().index):  latlon.loc[latlon.city==city, 'spots\_num'] = spot.City.value\_counts()[i]  latlon['spots\_num'] = latlon['spots\_num'].fillna(0) latlon = latlon.astype({'spots\_num':'int'})  # 오늘로부터 7일 이상 이후에 여행을 가는 경우 : 과거 기후 데이터 불러오기 df\_temp = pd.read\_excel(directory\_path + '기상청\_순별\_평균기후값.xlsx', sheet\_name='temp\_avg') df\_temp\_min = pd.read\_excel(directory\_path + '기상청\_순별\_평균기후값.xlsx', sheet\_name='temp\_min') df\_temp\_max = pd.read\_excel(directory\_path + '기상청\_순별\_평균기후값.xlsx', sheet\_name='temp\_max') df\_wind = pd.read\_excel(directory\_path + '기상청\_순별\_평균기후값.xlsx', sheet\_name='wind') df\_rain = pd.read\_excel(directory\_path + '기상청\_순별\_평균기후값.xlsx', sheet\_name='rain') df\_rhumid = pd.read\_excel(directory\_path + '기상청\_순별\_평균기후값.xlsx', sheet\_name='rhumid') df\_weather = pd.read\_excel(directory\_path + '기상청\_순별\_평균기후값.xlsx', sheet\_name='weather')  # 기온별 추천 옷차림 데이터 clothes = pd.read\_excel(directory\_path + '날씨별옷차림.xlsx') clothes\_np = pd.DataFrame.to\_numpy(clothes) clothes\_col = ['겉옷', '상의', '하의', '신발', '보온', '강풍', '참고']  # 전국 공영주차장 정보 데이터 parking\_lot = pd.read\_excel(directory\_path + '전국주차장정보.xlsx') |
| --- |

1. ‘함수 정의’ 속 코드 모두 실행
   1. 위경도 ↔ 좌표 변환

기상청 API 자료에서는 위치 정보가 x, y 좌표지만, 관광코스 지점정보는 위경도 자료이다. 따라서 이 두 자료의 위치 정보를 통일 하기 위해, 위경도를 x, y 좌표로 변환한다.

| # 좌표 <-> 위경도 변환  import math NX = 149 ## X축 격자점 수 NY = 253 ## Y축 격자점 수  Re = 6371.00877 ## 지도반경 grid = 5.0 ## 격자간격 (km) slat1 = 30.0 ## 표준위도 1 slat2 = 60.0 ## 표준위도 2 olon = 126.0 ## 기준점 경도 olat = 38.0 ## 기준점 위도 xo = 210 / grid ## 기준점 X좌표 yo = 675 / grid ## 기준점 Y좌표 first = 0  if first == 0 :  PI = math.asin(1.0) \* 2.0  DEGRAD = PI/ 180.0  RADDEG = 180.0 / PI   re = Re / grid  slat1 = slat1 \* DEGRAD  slat2 = slat2 \* DEGRAD  olon = olon \* DEGRAD  olat = olat \* DEGRAD   sn = math.tan(PI \* 0.25 + slat2 \* 0.5) / math.tan(PI \* 0.25 + slat1 \* 0.5)  sn = math.log(math.cos(slat1) / math.cos(slat2)) / math.log(sn)  sf = math.tan(PI \* 0.25 + slat1 \* 0.5)  sf = math.pow(sf, sn) \* math.cos(slat1) / sn  ro = math.tan(PI \* 0.25 + olat \* 0.5)  ro = re \* sf / math.pow(ro, sn)  first = 1  def mapToGrid(lat, lon, code = 0 ): #위경도->그리드  ra = math.tan(PI \* 0.25 + lat \* DEGRAD \* 0.5)  ra = re \* sf / pow(ra, sn)  theta = lon \* DEGRAD - olon  if theta > PI :  theta -= 2.0 \* PI  if theta < -PI :  theta += 2.0 \* PI  theta \*= sn  x = (ra \* math.sin(theta)) + xo  y = (ro - ra \* math.cos(theta)) + yo  x = int(x + 1.5)  y = int(y + 1.5)  return x, y |
| --- |

* 1. 출발지 입력

| global area\_input global latlon\_sort  def select\_spoint():  while(1):  area\_input = input('''\n어디에서 출발하시나요? 도시명만 입력하세요. [주의] 광주(광주,경기광주), 고성(강원고성,경상고성)을 구분해 적어주세요. \n''')    if not (latlon.city==area\_input).any(): # 우리나라 도시명에 포함되지 않은 값을 입력한 경우  print('정보가 없는 지역입니다. 다른 지역을 입력해주세요.')  else:  # 출발지에 해당하는 (위도(lat), 경도(lon)) 값을 city1에 저장  city1 = (latlon.lat[(latlon.city == area\_input)].values[0],  latlon.lon[(latlon.city == area\_input)].values[0])  # lat, lon 데이터를 이용해 출발지(city1)와 다른 도시들 간 거리를 계산해 distance 열에 저장  latlon['distance'] = latlon[['lat', 'lon']].apply(lambda x:   round(distance.distance(city1, (x['lat'], x['lon'])).km, 3), axis=1)  break   return area\_input |
| --- |

* 1. 여행지 입력

| def where\_to\_trip():  global city\_input, month, day    # 여행 도시 결정  while(1):  trip\_code = int(input('여행 지역을 정하셨나요?\n [1] 네\n [2] 아니오\n'))-1  if trip\_code not in (0, 1):  print('잘못된 코드입니다. 정확한 코드를 입력해주세요.\n')  else:  break  # 여행갈 도시 입력  if trip\_code == 0: # 여행지역 정한 경우 --> 날짜 선택으로 넘어감  while(1):  city\_input = input('\n어디로 여행을 가시나요?: ')  if not (latlon.city==city\_input).any():  print('정보가 없는 지역입니다. 다른 지역을 입력해주세요.')  else:   print(city\_input+'에 있는 관광 코스를 추천할게요.\n')  month, day = select\_date(trip\_code)  break    elif trip\_code == 1: # 여행지역 미정 --> 여행지역 추천코드로 넘어감  city\_input = recommend\_city(area\_input, trip\_code)    return city\_input |
| --- |

* 1. 여행지 추천 및 선택

| def recommend\_city(area\_input, citydefined):  global month, day, latlon\_sort   # 여행지 추천의 기준  # 1: 가까울 것  # 2: 날씨가 좋을 것  # 3: 관광스팟이 다양할 것   if citydefined == 1: # 여행지 안정해진 경우 --> 날짜 정한 후 추천하는 코드 실행  month, day = select\_date(citydefined)    print('출발 도시 ['+area\_input+']에서 가깝고, 관광스팟이 많은 도시 상위 15곳을 추천합니다.\n')   # 출발도시와 전국 도시들 간의 거리 & 관광지 수에 따라 정렬  latlon\_sort = latlon.sort\_values(by=['distance', 'spots\_num'],  ascending=[True, False]).values[:]   # 상위 15개 도시들의 정보만 출력  pcount = 0  for i in range(len(latlon\_sort)):  if latlon\_sort[i,6] < 10:  continue  print('['+latlon\_sort[i,1]+'] 거리: '+str(round(latlon\_sort[i,5],1))  +' km 관광스팟 수: '+str(latlon\_sort[i,6]))  pcount += 1  if pcount > 15:  break   while(1):  city\_input = input('\n여행지를 선택하세요. 위 목록에 없는 도시도 선택할 수 있습니다. ')  if not (latlon.city==city\_input).any(): # 도시가 존재하지 않는 경우  print('정보가 없는 지역입니다. 다른 지역을 입력해주세요.')  elif latlon['spots\_num'][latlon.city==city\_input].values[0] == 0:  # 도시는 존재하지만 도시에 포함된 관광스팟이 0개인 경우  print('추천 관광지가 없는 지역입니다. 다른 지역을 입력해주세요.')  else:   print(city\_input+'에 있는 관광 코스를 추천할게요.\n')  break   return city\_input |
| --- |

* 1. 여행 예정 날짜 입력

| def select\_date(code):  global diffdate   while(1): # 올바른 형식의 여행 날짜 입력받기  mmdd = input('\n정확한 여행 날짜를 입력해주세요 (m dd, ex: 7 31): ')  if mmdd.find(' ') not in range(1,3):  print('잘못된 형식입니다. 정확하게 입력해주세요.')  else:  try:  month = int(mmdd.split(' ', maxsplit=1)[0])  day = int(mmdd.split(' ', maxsplit=1)[1])  if month not in range(1,13) or day not in range(1,32):  print('존재하지 않는 날짜입니다. 정확하게 입력해주세요.')  elif month == 2 and day > 29:  print('존재하지 않는 날짜입니다. 정확하게 입력해주세요.')  else:  # 실행일과 여행일 간의 날짜 차이 계산  diffdate = classify\_date(day,month,code)   break  except ValueError: # 입력값에 오타 등이 있을 경우 에러를 고려  print('잘못된 형식입니다. 정확하게 입력해주세요.')   return month,day |
| --- |

* 1. 여행일까지 남은 일수 계산

| def classify\_date(day,month,citydefined):  global city\_input, diffdate, tripdate   # 프로그램 실행 일시 저장  now = datetime.now() + timedelta(hours=9)  mnow = now.month  dnow = now.day   # 일기예보 정보가 존재하는 기한 저장  nextweek = now + timedelta(weeks=1)  mweek = nextweek.month  dweek = nextweek.day   # 날짜 형식 바꾸기  datestring = '2022 '+str(month)+' '+str(day)  tripdate = datetime.strptime(datestring, '%Y %m %d')   print('오늘은 '+str(mnow)+'월 '+str(dnow)+'일 입니다.')  print('일주일 후인 '+str(mweek)+'월 '+str(dweek)+'일까지의 일기예보 데이터가 존재합니다.')  print('')  print('선택하신 여행 일자는 '+str(month)+'월 '+str(day)+'일 입니다.')   diffdate = tripdate - now + timedelta(days=1) # 시간단위는 올림 --> day+1    return diffdate.days |
| --- |

* 1. 날씨 데이터 불러오기- 단기 예보

| def all\_weather\_short(latlon\_city): #추천 도시 단기 예보 가져오기   all\_mean\_fcstrain0=[] #평균 강수확률  all\_mean\_skystate0=[] #평균 구름량  all\_mean\_fcstrain1=[]   all\_mean\_skystate1=[]  all\_mean\_fcstrain2=[]   all\_mean\_skystate2=[]   all\_mean\_fcstrain=[] #평균 강수확률  all\_mean\_skystate=[] #평균 구름량   import warnings  warnings.filterwarnings('ignore')   #예보 날짜와 시간 맞추기  today = datetime.now(timezone('Asia/Seoul'))  nowdate=today.strftime("%Y%m%d") #현재 날짜 YYYYMMDD  nowtime=today.strftime('%H') #현재 시간 HH   # 0200, 0500, 0800, 1100, 1400, 1700, 2000, 2300 발표 예보 중 가까운 시간 예보 선택  if int(nowtime)>2 and int(nowtime)<=5: #03시~05시는 당일 02시 예보  base\_time='0200' #예보발표시간  base\_date=nowdate #예보발표날짜  elif int(nowtime)>5 and int(nowtime)<=8: #06시~08시는 당일 05시 예보  base\_time='0500'  base\_date=nowdate  elif int(nowtime)>8 and int(nowtime)<=11: #09시~11시는 당일 08시 예보  base\_time='0800'  base\_date=nowdate  elif int(nowtime)>11 and int(nowtime)<=14: #12시~14시는 당일 11시 예보  base\_time='1100'  base\_date=nowdate  elif int(nowtime)>14 and int(nowtime)<=17: #15시~17시는 당일 14시 예보  base\_time='1400'  base\_date=nowdate  elif int(nowtime)>17 and int(nowtime)<=20: #18시~20시는 당일 17시 예보  base\_time='1700'  base\_date=nowdate  elif int(nowtime)>20 and int(nowtime)<=23: #21시~23시는 당일 20시 예보  base\_time='2000'  base\_date=nowdate  elif int(nowtime)>=0 and int(nowtime)<=2: #00시~02시는 전날 23시 예보  base\_time='2300'  base\_date=str(int(nowdate)-1)   nx\_city=[] #추천 도시 x좌표  ny\_city=[] #추천 도시 y좌표   for city in latlon\_city: #추천 도시 latlon\_sort 반복문에 넣기  for index in spot.index:  if spot.iloc[index][4]==city:   lat\_city=spot.iloc[index][7]  lon\_city=spot.iloc[index][8]   nx\_city.append(str(mapToGrid(lat\_city, lon\_city)[0]))  ny\_city.append(str(mapToGrid(lat\_city, lon\_city)[1]))  break   #단기예보 API 추천 도시 데이터 가져오기  for x, y in zip(nx\_city,ny\_city):  url='http://apis.data.go.kr/1360000/VilageFcstInfoService\_2.0/getVilageFcst'  key='75Nc/StoGwpn4Jl5F6UM47m2RQdsgGuNKIOxlNcu8b96uJVbNTl/8LbW4Za0h2OQkmYEST+D/L8ZxCcvo2RUbw=='  para={'serviceKey':key, 'pageNo' : '1', 'numOfRows' : '1000', 'dataType' : 'JSON', 'base\_date':base\_date, 'base\_time':base\_time, 'nx':x, 'ny':y }   res = requests.get(url, params=para)   res\_json = json.loads(res.content)  items=res\_json["response"]['body']['items']['item']   #예보 날짜별로 분류  now = datetime.now(timezone('Asia/Seoul'))  after1 = now + timedelta(days=1)  after2 = now + timedelta(days=2)  after\_1day=after1.strftime("%Y%m%d")  after\_2day=after2.strftime("%Y%m%d")   d0=[] #예보일 당일 날씨  d1=[] #예보일에서 1일 날씨  d2=[] #예보일에서 2일 날씨   for i in items:  if i['fcstDate']==nowdate:  d0.append(i)  elif i['fcstDate']==after\_1day:  d1.append(i)  elif i['fcstDate']==after\_2day:  d2.append(i)   #예보 날짜 별 데이터프레임 생성  day0=pd.DataFrame(d0)  day1=pd.DataFrame(d1)  day2=pd.DataFrame(d2)   #예보 종류 한글로 바꾸기('하늘상태','강수확률')  day0.replace(['SKY','POP'],['하늘상태','강수확률'],inplace=True)  day1.replace(['SKY','POP'],['하늘상태','강수확률'],inplace=True)  day2.replace(['SKY','POP'],['하늘상태','강수확률'],inplace=True)   #0일  fcstrain0=[]   for i in day0.index[day0['category']=='강수확률']:  fcstrain0.append(int(day0.iloc[i][5]))  skystate0=[]   for i in day0.index[day0['category']=='하늘상태']:  skystate0.append(int(day0.iloc[i][5]))   mean\_fcstrain0=sum(fcstrain0)/len(fcstrain0)  all\_mean\_fcstrain0.append(mean\_fcstrain0)    mean\_skystate0=sum(skystate0)/len(skystate0)  all\_mean\_skystate0.append(mean\_skystate0)   #1일  fcstrain1=[]   for i in day1.index[day1['category']=='강수확률']:  fcstrain1.append(int(day1.iloc[i][5]))  skystate1=[]   for i in day1.index[day1['category']=='하늘상태']:  skystate1.append(int(day1.iloc[i][5]))   mean\_fcstrain1=sum(fcstrain1)/len(fcstrain1)  all\_mean\_fcstrain1.append(mean\_fcstrain1)    mean\_skystate1=sum(skystate1)/len(skystate1)  all\_mean\_skystate1.append(mean\_skystate1)   #2일  fcstrain2=[]   for i in day2.index[day2['category']=='강수확률']:  fcstrain2.append(int(day2.iloc[i][5]))  skystate2=[]   for i in day2.index[day2['category']=='하늘상태']:  skystate2.append(int(day2.iloc[i][5]))   mean\_fcstrain2=sum(fcstrain2)/len(fcstrain2)  all\_mean\_fcstrain2.append(mean\_fcstrain2)   mean\_skystate2=sum(skystate2)/len(skystate2)  all\_mean\_skystate2.append(mean\_skystate2)   #데이터 프레임에 추가 : df\_latlon\_fcst  df\_allfcst\_day0 = []  df\_allfcst\_day1 = []  df\_allfcst\_day2 = []  for i in range(len(latlon\_city)):  df\_allfcst\_day0.append('['+latlon\_city[i]+'] 평균강수확률 : '+str(round(all\_mean\_fcstrain0[i],1))+' % 평균구름량: '+str(round(all\_mean\_skystate0[i],1)))  for i in range(len(latlon\_city)):  df\_allfcst\_day1.append('['+latlon\_city[i]+'] 평균강수확률 : '+str(round(all\_mean\_fcstrain1[i],1))+' % 평균구름량: '+str(round(all\_mean\_skystate1[i],1)))  for i in range(len(latlon\_city)):  df\_allfcst\_day2.append('['+latlon\_city[i]+'] 평균강수확률 : '+str(round(all\_mean\_fcstrain2[i],1))+' % 평균구름량: '+str(round(all\_mean\_skystate2[i],1)))  return df\_allfcst\_day0, df\_allfcst\_day1, df\_allfcst\_day2   return df\_allfcst\_day0, df\_allfcst\_day1, df\_allfcst\_day2 |
| --- |

* 1. 날씨 데이터 불러오기 - 중기예보

| def all\_weather\_mid(YYYYMMDD): #전국 중기예보 (3일~7일) #파라미터 : YYYYMMDD ex)20220610 (주의! 당일로 부터 3일~7일 후 날짜 중에 입력해야함)   #예보 날짜와 시간 맞추기  today = datetime.now(timezone('Asia/Seoul'))  nowdate=today.strftime("%Y%m%d") #현재 날짜 YYYYMMDD  nowtime=today.strftime('%H') #현재 시간 HH   #06시 18시 발표 예보 중 가까운 시간 예보 선택  if int(nowtime)>6 and int(nowtime)<=18: #07시~18시는 당일 06시 예보  time='0600'  elif int(nowtime)>18 and int(nowtime)<=23: #19시~23시는 당일 18시 예보  time='1800'  elif int(nowtime)>=0 and int(nowtime)<=6: #0시~6시는 어제 18시 예보  time='1800'  nowdate=str(int(nowdate)-1)   tmFc=nowdate+time #예보발표시각   # landcode : 중기육상예보코드 리스트   landcode\_before=[]  for i in spot.iloc[:, 2]:  landcode\_before.append(i)   landcode\_set=set(landcode\_before) #중복코드 제거  landcode=list(landcode\_set)   #날짜에 맞게 출력 할 예보 가져오기  now=datetime.now(timezone('Asia/Seoul')).replace(tzinfo=None) #현재 날짜  nowdate=now.strftime('%Y%m%d')  nowcompare = datetime.strptime(nowdate, "%Y%m%d")  datecompare = datetime.strptime(str(YYYYMMDD), "%Y%m%d") #여행 날짜  datediff=datecompare-nowcompare #날짜 차이  daydiff=datediff.days #day 차이   #출력에 필요한 날짜 분리  YYYY=str(YYYYMMDD)[0:4]  MM=str(YYYYMMDD)[4:6]  DD=str(YYYYMMDD)[6:]   print(YYYY,"년", MM, "월", DD, "일의 전국 날씨 예보입니다.\n")   for code in landcode: #전국 중기육상예보 조회 (오전/오후 운량, 강수확률)   url1 = 'http://apis.data.go.kr/1360000/MidFcstInfoService/getMidLandFcst'  key1='75Nc/StoGwpn4Jl5F6UM47m2RQdsgGuNKIOxlNcu8b96uJVbNTl/8LbW4Za0h2OQkmYEST+D/  L8ZxCcvo2RUbw=='  para1 ={'serviceKey':key1, 'pageNo' : '1', 'numOfRows' : '10', 'dataType' : 'JSON',  'regId' : code , 'tmFc' : tmFc }   res1 = requests.get(url1, params=para1)   res1\_json = json.loads(res1.content)  items1=res1\_json["response"]['body']['items']['item'][0]   day3=dict() #3일 후 예보  day4=dict() #4일 후 예보  day5=dict() #5일 후 예보  day6=dict() #6일 후 예보  day7=dict() #7일 후 예보   #3일 후 예보  day3['amrain']=items1['rnSt3Am'] #amrain:오전강수확률  day3['pmrain']=items1['rnSt3Pm'] #pmrain:오후강수확률  day3['amcloud']=items1['wf3Am'] #amcloud:오전운량  day3['pmcloud']=items1['wf3Pm'] #pmcloud:오후운량   #4일 후 예보  day4['amrain']=items1['rnSt4Am'] #amrain:오전강수확률  day4['pmrain']=items1['rnSt4Pm'] #pmrain:오후강수확률  day4['amcloud']=items1['wf4Am'] #amcloud:오전운량  day4['pmcloud']=items1['wf4Pm'] #pmcloud:오후운량   #5일 후 예보  day5['amrain']=items1['rnSt5Am'] #amrain:오전강수확률  day5['pmrain']=items1['rnSt5Pm'] #pmrain:오후강수확률  day5['amcloud']=items1['wf5Am'] #amcloud:오전운량  day5['pmcloud']=items1['wf5Pm'] #pmcloud:오후운량   #6일 후 예보  day6['amrain']=items1['rnSt6Am'] #amrain:오전강수확률  day6['pmrain']=items1['rnSt6Pm'] #pmrain:오후강수확률  day6['amcloud']=items1['wf6Am'] #amcloud:오전운량  day6['pmcloud']=items1['wf6Pm'] #pmcloud:오후운량   #7일 후 예보  day7['amrain']=items1['rnSt7Am'] #amrain:오전강수확률  day7['pmrain']=items1['rnSt7Pm'] #pmrain:오후강수확률  day7['amcloud']=items1['wf7Am'] #amcloud:오전운량  day7['pmcloud']=items1['wf7Pm'] #pmcloud:오후운량   #예보구역코드에 해당하는 도시이름 가져오기 #citylist:도시이름 저장된 리스트  allcitylist=[]  for i in spot.index[spot.iloc[:,2]==code]:  allcitylist.append(spot.iloc[i,4])  allcityset=set(allcitylist)  citylist=list(allcityset)  print('지역 : ', citylist)   if daydiff==3: # N일 후이면 dayN 가져오기  print('오전강수확률:', day3['amrain'], '%/ 오후강수확률:', day3['pmrain'],  '%\n오전하늘상태:', day3['amcloud'], '/ 오후하늘상태:', day3['pmcloud'])    elif daydiff==4:  print('오전강수확률:', day4['amrain'], '%/ 오후강수확률:', day4['pmrain'],  '%\n오전하늘상태:', day4['amcloud'], '/ 오후하늘상태:', day4['pmcloud'])   elif daydiff==5:  print('오전강수확률:', day5['amrain'], '%/ 오후강수확률:', day5['pmrain'],  '%\n오전하늘상태:', day5['amcloud'], '/ 오후하늘상태:', day5['pmcloud'])   elif daydiff==6:  print('오전강수확률:', day6['amrain'], '%/ 오후강수확률:', day6['pmrain'],  '%\n오전하늘상태:', day6['amcloud'], '/ 오후하늘상태:', day6['pmcloud'])   elif daydiff==7:  print('오전강수확률:', day7['amrain'], '%/ 오후강수확률:', day7['pmrain'],  '%\n오전하늘상태:', day7['amcloud'], '/ 오후하늘상태:', day7['pmcloud'])   print('\n') |
| --- |

* 1. 날씨 데이터 출력

| def print\_all\_weather():   global inputdate # 입력한 여행 날짜 YYYYMMDD 형  global daydiff #날짜 차이   if len(str(month))==1:  getmonth='0'+str(month)  else:  getmonth=str(month)   if len(str(day))==1:  getday='0'+str(day)  else:  getday=str(day)   inputdate='2022'+getmonth+getday #inputdate : 입력한 여행 날짜 YYYYMMDD 형   now=datetime.now(timezone('Asia/Seoul')).replace(tzinfo=None) #현재 날짜  nowdate=now.strftime('%Y%m%d')  nowcompare = datetime.strptime(nowdate, "%Y%m%d")  datecompare = datetime.strptime(inputdate, "%Y%m%d") #여행 날짜  datediff=datecompare-nowcompare #날짜 차이  daydiff=abs(datediff.days)   if daydiff<=2:  print('추천 지역의 날씨를 가져오는 중입니다. 잠시만 기다려 주세요.\n')  all\_weather\_short(latlon\_city)  print\_all\_weather\_short()  elif daydiff>=3 and daydiff<=7:  print('추천 지역의 날씨를 가져오는 중입니다. 잠시만 기다려 주세요.\n')  all\_weather\_mid(int(inputdate))  else:  print("7일 이후 날짜는 일기예보 데이터가 없습니다.\n거리와 관광스팟 수를 참고해 원하는 여행지를 선택해주세요.") |
| --- |

| def print\_all\_weather\_short(): #추천 도시 단기 예보 출력  if len(str(month))==1:  getmonth='0'+str(month)  else:  getmonth=str(month)   if len(str(day))==1:  getday='0'+str(day)  else:  getday=str(day)   inputdate='2022'+getmonth+getday #inputdate : 입력한 여행 날짜 YYYYMMDD 형   #날짜에 맞게 출력 할 예보 가져오기  now=datetime.now(timezone('Asia/Seoul')).replace(tzinfo=None) #현재 날짜  nowdate=now.strftime('%Y%m%d')  nowcompare = datetime.strptime(nowdate, "%Y%m%d")  datecompare = datetime.strptime(inputdate, "%Y%m%d") #여행 날짜  datediff=datecompare-nowcompare #날짜 차이  daydiff=abs(datediff.days) #day 차이   get\_all\_weather\_short=all\_weather\_short(latlon\_city)   if daydiff == 0:  print('거리가 가까운 상위 15개 지역의 날씨 예보 입니다.')  for i in range(len(get\_all\_weather\_short[0])):  print(get\_all\_weather\_short[0][i])  elif daydiff==1:  print('거리가 가까운 상위 15개 지역의 날씨 예보 입니다.')  for i in range(len(get\_all\_weather\_short[1])):  print(get\_all\_weather\_short[1][i])   elif daydiff==2:  print('거리가 가까운 상위 15개 지역의 날씨 예보 입니다.')  for i in range(len(get\_all\_weather\_short[2])):  print(get\_all\_weather\_short[2][i]) |
| --- |

* 1. 여행지역 날씨 데이터 불러오기 - 날짜 분류(단기 / 중기 / 그 이후)

| def forecast():  global diffdate  # 프로그램 실행일과 여행일의 날짜 차이(diffdate)에 따라 다른 코드를 실행함   if diffdate in range(0,3):  # 단기예보 정보 불러오기  print('오늘로부터 2일 이내에 여행을 가시네요.\n')  weather\_short(int(inputdate), city\_input)   elif diffdate in range(3,8):  # 중기예보 정보 불러오기  print('오늘로부터 7일 이내에 여행을 가시네요.\n')  weather\_mid(int(inputdate), city\_input)   else:  # 과거 평년 기후값 불러오기  print('오늘로부터 7일 이상 지난 후에 여행을 가시네요.\n')  weather\_past(day, month) |
| --- |

* 1. 여행지역 날씨 데이터 불러오기 - 단기예보

| # 단기예보 (당일~2일 후)  def weather\_short(datecompare, city): #city는 string으로 넣기   #예보 날짜와 시간 맞추기  today = datetime.now(timezone('Asia/Seoul'))  nowdate=today.strftime("%Y%m%d") #현재 날짜 YYYYMMDD  nowtime=today.strftime('%H') #현재 시간 HH   # 0200, 0500, 0800, 1100, 1400, 1700, 2000, 2300 발표 예보 중 가까운 시간 예보 선택  if int(nowtime)>2 and int(nowtime)<=5: #03시~05시는 당일 02시 예보  base\_time='0200' #예보발표시간  base\_date=nowdate #예보발표날짜  elif int(nowtime)>5 and int(nowtime)<=8: #06시~08시는 당일 05시 예보  base\_time='0500'  base\_date=nowdate  elif int(nowtime)>8 and int(nowtime)<=11: #09시~11시는 당일 08시 예보  base\_time='0800'  base\_date=nowdate  elif int(nowtime)>11 and int(nowtime)<=14: #12시~14시는 당일 11시 예보  base\_time='1100'  base\_date=nowdate  elif int(nowtime)>14 and int(nowtime)<=17: #15시~17시는 당일 14시 예보  base\_time='1400'  base\_date=nowdate  elif int(nowtime)>17 and int(nowtime)<=20: #18시~20시는 당일 17시 예보  base\_time='1700'  base\_date=nowdate  elif int(nowtime)>20 and int(nowtime)<=23: #21시~23시는 당일 20시 예보  base\_time='2000'  base\_date=nowdate  elif int(nowtime)>=0 and int(nowtime)<=2: #00시~02시는 전날 23시 예보  base\_time='2300'  base\_date=str(int(nowdate)-1)   #관광지 city 위경도 -> xy좌표 변환   for index in spot.index:  if spot.iloc[index][4]==city:  lat\_city=spot.iloc[index][7]  lon\_city=spot.iloc[index][8]   nx\_city=str(mapToGrid(lat\_city, lon\_city)[0]) #nx\_city : x좌표  ny\_city=str(mapToGrid(lat\_city, lon\_city)[1]) #ny\_city : y좌표   #단기예보 API 가져오기  url='http://apis.data.go.kr/1360000/VilageFcstInfoService\_2.0/getVilageFcst'  key='75Nc/StoGwpn4Jl5F6UM47m2RQdsgGuNKIOxlNcu8b96uJVbNTl/8LbW4Za0h2OQkmYEST+D/L8ZxCcvo2RUbw=='  para={'serviceKey':key, 'pageNo' : '1', 'numOfRows' : '1000', 'dataType' : 'JSON', 'base\_date':base\_date, 'base\_time':base\_time, 'nx': nx\_city, 'ny': ny\_city }   res = requests.get(url, params=para)   res\_json = json.loads(res.content)  items=res\_json["response"]['body']['items']['item']   # 예보 날짜별로 분류  now = datetime.now()  after1 = now + timedelta(days=1)  after2 = now + timedelta(days=2)  after\_1day=after1.strftime("%Y%m%d")  after\_2day=after2.strftime("%Y%m%d")   d0=[] #예보일 당일 날씨  d1=[] #예보일에서 1일 날씨  d2=[] #예보일에서 2일 날씨   for i in items:  if i['fcstDate']==nowdate:  d0.append(i)  elif i['fcstDate']==after\_1day:  d1.append(i)  elif i['fcstDate']==after\_2day:  d2.append(i)   #예보 날짜 별 데이터프레임 생성  day0=pd.DataFrame(d0)  day1=pd.DataFrame(d1)  day2=pd.DataFrame(d2)   #필요없는 예보 삭제하기  del0=day0['category'].isin(['UUU','VVV','VEC','WAV'])   day0=day0[~del0]  del1=day1['category'].isin(['UUU','VVV','VEC','WAV'])   day1=day1[~del1]  del2=day2['category'].isin(['UUU','VVV','VEC','WAV'])   day2=day2[~del2]   #예보 종류 한글로 바꾸기('1시간 기온','풍속','하늘상태','강수형태','강수확률','1시간 강수량','습도','1시간 적설량')  day0.replace(['TMP','WSD','SKY','PTY','POP','PCP','REH','SNO'],['1시간 기온','풍속','하늘상태','강수형태','강수확률','1시간 강수량','습도','1시간 적설량'],inplace=True)  day1.replace(['TMP','WSD','SKY','PTY','POP','PCP','REH','SNO'],['1시간 기온','풍속','하늘상태','강수형태','강수확률','1시간 강수량','습도','1시간 적설량'],inplace=True)  day2.replace(['TMP','WSD','SKY','PTY','POP','PCP','REH','SNO'],['1시간 기온','풍속','하늘상태','강수형태','강수확률','1시간 강수량','습도','1시간 적설량'],inplace=True)   #날짜에 맞게 출력 할 예보 가져오기  now=datetime.now() #현재 날짜  #datecompare = datetime.strptime(str(YYYYMMDD), "%Y%m%d") #여행 날짜  datediff=datecompare-now #날짜 차이  daydiff=datediff.days #day 차이   if daydiff==0: # N일 후이면 dayN 가져오기  fcstday=day0  elif daydiff==1:  fcstday=day1  elif daydiff==2:  fcstday=day2   fcsttime=[] #fcsttime:예보시간  for i in np.unique(np.array(fcstday['fcstTime'])):  fcsttime.append(i[0:2])   fcsttemp=[] #fcsttemp:1시간기온  for i in fcstday.index[fcstday['category']=='1시간 기온']:  fcsttemp.append(int(fcstday.loc[i][5]))   fcstrain=[] #fcstrain:강수확률  for i in fcstday.index[fcstday['category']=='강수확률']:  fcstrain.append(int(fcstday.loc[i][5]))   skystate=[] #skystate:하늘상태  for i in fcstday.index[fcstday['category']=='하늘상태']:  skystate.append(int(fcstday.loc[i][5]))   fcstwind=[] #fcstwind:풍속  for i in fcstday.index[fcstday['category']=='풍속']:  fcstwind.append(float(fcstday.loc[i][5]))   fcsthumid=[] #fcsthumid:습도  for i in fcstday.index[fcstday['category']=='습도']:  fcsthumid.append(int(fcstday.loc[i][5]))   fcstrain=[] #fcstrain:강수확률  for i in fcstday.index[fcstday['category']=='강수확률']:  fcstrain.append(int(fcstday.loc[i][5]))   #최고 최저 기온 구하기  min=fcsttemp[0]  max=fcsttemp[0]  for i in fcsttemp:  if i<min :  min=i  for i in fcsttemp:  if i>max :  max=i   ## 기온/강수확률/하늘상태 출력   date=fcstday.iloc[0][3] #예보날짜YYYYMMDD   #기온그래프그리기  plt.figure(figsize=(12,6))  plt.rc('font', family='NanumBarunGothic')  plt.title(date[0:4]+'년 '+date[4:6]+'월 '+date[6:]+'일'+' 기온 예보입니다', fontsize=20)  plt.plot(fcsttime, fcsttemp, color='#ff7e05', linewidth=5, marker='.',markersize=10,markeredgecolor='#fa5902', markeredgewidth=3) #x:예보시간, y:1시간기온  plt.xlabel('예보시간 [시]')  plt.ylabel('기온 [°C]')  for i, v in enumerate(fcsttime): #좌표에 기온 표시  plt.text(v, fcsttemp[i], fcsttemp[i], fontsize = 13, color='black')  plt.show()   #강수확률 그래프 그리기  plt.figure(figsize=(12,6))  plt.rc('font', family='NanumBarunGothic')  plt.title(date[0:4]+'년 '+date[4:6]+'월 '+date[6:]+'일'+' 강수확률 예보입니다', fontsize=20)  plt.plot(fcsttime,fcstrain, color='#1e99eb', linewidth=5, marker='.',markersize=10,markeredgecolor='#0865bd', markeredgewidth=3) #x:예보시간, y:강수확률  plt.ylim(0,100)  plt.xlabel('예보시간 [시]')  plt.ylabel('강수확률 [%]')  plt.show()   #하늘상태출력  for i in skystate:  if i==1:  skystate[skystate.index(i)]='맑음'  elif i==2:  skystate[skystate.index(i)]='구름조금'  elif i==3:  skystate[skystate.index(i)]='구름많음'  elif i==4:  skystate[skystate.index(i)]='흐림'   skystate\_frame=pd.DataFrame(skystate, index=fcsttime, columns=['하늘상태']).T  skystate\_frame.columns.name='예보 시간 [시]'  needcolumns=[]  for i in range(0, len(skystate\_frame.columns), 3): #3시간 간격만 뽑아오기  needcolumns.append(i)   print(colored(date[0:4]+'년 '+date[4:6]+'월 '+date[6:]+'일'+' 하늘상태 예보입니다.', 'blue', attrs=['bold'])) #출력   ## 옷차림 추천  recommend\_clothes(month, fcsttemp, min, max, fcstwind, fcstrain, fcsthumid, skystate)    return skystate\_frame.iloc[0,needcolumns] |
| --- |

* 1. 여행지역 날씨 데이터 불러오기 - 중기예보

| # 중기예보 (3일~7일 후) def weather\_mid(datecompare, city): #city는 string으로 넣기  global fcstday   #예보 날짜와 시간 맞추기  today = datetime.now(timezone('Asia/Seoul'))  nowdate=today.strftime("%Y%m%d") #현재 날짜 YYYYMMDD  nowtime=today.strftime('%H') #현재 시간 HH   #06시 18시 발표 예보 중 가까운 시간 예보 선택  if int(nowtime)>6 and int(nowtime)<=18: #07시~18시는 당일 06시 예보  time='0600'  elif int(nowtime)>18 and int(nowtime)<=23: #19시~23시는 당일 18시 예보  time='1800'  elif int(nowtime)>=0 and int(nowtime)<=6: #0시~6시는 어제 18시 예보  time='1800'  nowdate=str(int(nowdate)-1)   tmFc=nowdate+time #예보발표시각   #예보구역 코드 맞추기(regld)   #중기육상 : regld\_1  for index in spot.index:  if spot.iloc[index][4]==city:  regld\_1=spot.iloc[index][2]  break   #중기기온 : regld\_2  for index in spot.index:  if spot.iloc[index][4]==city:  regld\_2=spot.iloc[index][1]  break   #1:중기육상예보조회(오전오후 강수확률,운량)   url1 = 'http://apis.data.go.kr/1360000/MidFcstInfoService/getMidLandFcst'  key1='75Nc/StoGwpn4Jl5F6UM47m2RQdsgGuNKIOxlNcu8b96uJVbNTl/8LbW4Za0h2OQkmYEST+D/L8ZxCcvo2RUbw=='  para1 ={'serviceKey':key1, 'pageNo' : '1', 'numOfRows' : '10', 'dataType' : 'JSON', 'regId' : regld\_1 , 'tmFc' : tmFc }   res1 = requests.get(url1, params=para1)   res1\_json = json.loads(res1.content)  items1=res1\_json["response"]['body']['items']['item'][0]   #2:중기기온조회(최고최저기온)   url2 = 'http://apis.data.go.kr/1360000/MidFcstInfoService/getMidTa'  key2='75Nc/StoGwpn4Jl5F6UM47m2RQdsgGuNKIOxlNcu8b96uJVbNTl/8LbW4Za0h2OQkmYEST+D/L8ZxCcvo2RUbw=='  para2 ={'serviceKey' : key2 , 'pageNo' : '1', 'numOfRows' : '10', 'dataType' : 'JSON', 'regId' : regld\_2, 'tmFc' : tmFc }   res2 = requests.get(url2, params=para2)   res2\_json = json.loads(res2.content)  items2=res2\_json["response"]['body']['items']['item'][0]    day3=dict() #3일 후 예보  day4=dict() #4일 후 예보  day5=dict() #5일 후 예보  day6=dict() #6일 후 예보  day7=dict() #7일 후 예보   #3일 후 예보  day3['amrain']=items1['rnSt3Am'] #amrain:오전강수확률  day3['pmrain']=items1['rnSt3Pm'] #pmrain:오후강수확률  day3['amcloud']=items1['wf3Am'] #amcloud:오전운량  day3['pmcloud']=items1['wf3Pm'] #pmcloud:오후운량  day3['maxtemp']=items2['taMax3'] #maxtemp:최고기온  day3['mintemp']=items2['taMin3'] #mintemp:최저기온   #4일 후 예보  day4['amrain']=items1['rnSt4Am'] #amrain:오전강수확률  day4['pmrain']=items1['rnSt4Pm'] #pmrain:오후강수확률  day4['amcloud']=items1['wf4Am'] #amcloud:오전운량  day4['pmcloud']=items1['wf4Pm'] #pmcloud:오후운량  day4['maxtemp']=items2['taMax4'] #maxtemp:최고기온  day4['mintemp']=items2['taMin4'] #mintemp:최저기온   #5일 후 예보  day5['amrain']=items1['rnSt5Am'] #amrain:오전강수확률  day5['pmrain']=items1['rnSt5Pm'] #pmrain:오후강수확률  day5['amcloud']=items1['wf5Am'] #amcloud:오전운량  day5['pmcloud']=items1['wf5Pm'] #pmcloud:오후운량  day5['maxtemp']=items2['taMax5'] #maxtemp:최고기온  day5['mintemp']=items2['taMin5'] #mintemp:최저기온   #6일 후 예보  day6['amrain']=items1['rnSt6Am'] #amrain:오전강수확률  day6['pmrain']=items1['rnSt6Pm'] #pmrain:오후강수확률  day6['amcloud']=items1['wf6Am'] #amcloud:오전운량  day6['pmcloud']=items1['wf6Pm'] #pmcloud:오후운량  day6['maxtemp']=items2['taMax6'] #maxtemp:최고기온  day6['mintemp']=items2['taMin6'] #mintemp:최저기온   #7일 후 예보  day7['amrain']=items1['rnSt7Am'] #amrain:오전강수확률  day7['pmrain']=items1['rnSt7Pm'] #pmrain:오후강수확률  day7['amcloud']=items1['wf7Am'] #amcloud:오전운량  day7['pmcloud']=items1['wf7Pm'] #pmcloud:오후운량  day7['maxtemp']=items2['taMax7'] #maxtemp:최고기온  day7['mintemp']=items2['taMin7'] #mintemp:최저기온   #날짜에 맞게 출력 할 예보 가져오기  now=datetime.now() #현재 날짜  #datecompare = datetime.strptime(str(YYYYMMDD), "%Y%m%d") #여행 날짜  datediff=datecompare-now #날짜 차이  daydiff=datediff.days #day 차이    if daydiff==3: # N일 후이면 dayN 가져오기  fcstday=day3  elif daydiff==4:  fcstday=day4  elif daydiff==5:  fcstday=day5  elif daydiff==6:  fcstday=day6  elif daydiff==7:  fcstday=day7   #출력하기  #YYYY=str(YYYYMMDD)[0:4]  #MM=str(YYYYMMDD)[4:6]  #DD=str(YYYYMMDD)[6:]  YYYY = str(datecompare.years)  MM = str(datecompare.months)  DD = str(datecompare.days)   print(YYYY,"년", MM, "월", DD, "일의 날씨 예보입니다.\n") #입력한 여행날짜 넣기  print('최고기온:', fcstday['maxtemp'], ' °C/최저기온:', fcstday['mintemp'], ' °C\n오전강수확률:', fcstday['amrain'], '%/오후강수확률:', fcstday['pmrain'], '%\n오전하늘상태:', fcstday['amcloud'], '/오후하늘상태:', fcstday['pmcloud']) |
| --- |

* 1. 여행지역 날씨데이터 불러오기 - 과거데이터

| # 그 이후 (과거 30년 평균 기후 데이터)  def weather\_past(day, month):    # day값의 범위에 따라 상순/중순/하순으로 분류  if day in range(1,11):  day\_time = '상순'  elif day in range(11,21):  day\_time = '중순'  else:  day\_time = '하순'   when = str(month)+'월 '+day\_time   # 해당 여행지, 해당 일시의 날씨 정보를 각 변수에 저장  temp = df\_temp.loc[:,[when]][df\_temp.지점명 == city\_input].values[0][0]  temp\_min = df\_temp\_min.loc[:,[when]][df\_temp\_min.지점명 == city\_input].values[0][0]  temp\_max = df\_temp\_max.loc[:,[when]][df\_temp\_max.지점명 == city\_input].values[0][0]  wind = df\_wind.loc[:,[when]][df\_wind.지점명 == city\_input].values[0][0]  rain = df\_rain.loc[:,[when]][df\_rain.지점명 == city\_input].values[0][0]  rhumid = df\_rhumid.loc[:,[when]][df\_rhumid.지점명 == city\_input].values[0][0]  weather = df\_weather.loc[:,[when]][df\_weather.지점명 == city\_input].values[0][0]   print(city\_input +'의 '+when+' 평년 기온은 '+str(temp)+'도 입니다.')   # 불러온 과거 데이터를 기반으로 체감기온 계산 및 옷차림 추천 코드 실행  recommend\_clothes(month, temp, temp\_min, temp\_max, wind, rain, rhumid, weather) |
| --- |

* 1. 날씨정보에 따른 옷차림 추천

| ### 옷차림 추천   ########################### 단기예보 추가 버전 ####################수정 중#################### def recommend\_clothes(month, temp, temp\_min, temp\_max, wind, rain, rhumid, weather, type, \*args):   fcsttime=args  tempplist=[]  wind2=[]  tempp=[]  tempw=[]   if type==0: #단기예보 체감기온 계산   if month not in range(5,10): #겨울철 = 10 ~ 4월  for wind in wind:  wind2.append(wind \* 3.6) # 1 m/s = 3.6 km/h  for temp,wind2 in zip(temp, wind2):  tempp.append(round(13.12 + 0.6215\*temp - 11.37\*(wind2\*\*0.16) + 0.3965\*(wind2\*\*0.16)\*temp, 1)) # 계산시의 풍속 단위는 km/h 임!!!   for i in range(len(temp)):  if abs(temp[i]-tempp[i]) >= 1:  print(str(fcsttime[i])+'시에 풍속이'+str(wind[i])+'m/s로 강해 체감 기온이 낮습니다. 보온 악세사리를 잊지 마세요.\n')   else: #여름철 = 5 ~ 9월   for rhumid,temp in zip(rhumid, temp):  tempw.append(temp\*math.atan(0.151977\*((rhumid+8.313659)\*\*0.5)) + math.atan(temp+rhumid) - math.atan(rhumid-1.67633) + 0.00391838\*(rhumid\*\*(3/2))\*math.atan(0.023101\*rhumid) - 4.686035)    tempp = list(np.multiply(tempw, temp))   for i in range(len(tempp)):  if tempp[i] >= 33:  print(str(fcsttime[i])+'시에 체감온도가'+str(wind[i])+'도로, 폭염이 예상됩니다. 야외 활동 시 주의하세요.\n')   # 온도와 체감온도 평균값으로 바꾸기   temp=np.mean(np.array(temp))  tempp=np.mean(np.array(tempp))   elif type==1: #과거 기후일때 체감온도   if month not in range(5,10): #겨울철 = 10 ~ 4월  wind2 = wind \* 3.6 # 1 m/s = 3.6 km/h  tempp = round(13.12 + 0.6215\*temp - 11.37\*(wind2\*\*0.16) + 0.3965\*(wind2\*\*0.16)\*temp, 1) # 계산시의 풍속 단위는 km/h 임!!!   if abs(temp - tempp) >= 1: # 바람이 강해 기온과 체감온도에 1도 이상 차이가 있는 경우  print('풍속이 '+str(wind)+' m/s로 강해 체감 기온이 낮습니다. 보온 악세서리를 잊지 마세요.')   else: #여름철 = 5 ~ 9월  tempw = temp\*math.atan(0.151977\*((rhumid+8.313659)\*\*0.5)) + math.atan(temp+rhumid) - math.atan(rhumid-1.67633) + 0.00391838\*(rhumid\*\*(3/2))\*math.atan(0.023101\*rhumid) - 4.686035  tempp = round(-0.2442 + 0.55399\*tempw + 0.45535\*temp - 0.0022\*(tempw\*\*2) + 0.00278\*tempw\*temp + 3.5, 1)   if tempp >= 33: # 일최고체감기온 33도 넘으면 폭염에 해당함  print('일최고체감온도가 '+str(tempp)+'도로, 폭염이 예상됩니다. 야외 활동 시 주의하세요.\n')     ## 단기예보/과거예보 공통 부분   difftemp = temp\_max - temp\_min #일교차  print('여행일의 최고기온은 '+str(temp\_max)+'도, 최저기온은 '+str(temp\_min)+'도 입니다.\n')    # 기온별 옷차림 df에서, 내 기온에 해당하는 행의 인덱스를 저장  clothes\_maxind = clothes[(clothes.tmin <= temp\_max) & (clothes.tmax +1 >= temp\_max)].index  clothes\_minind = clothes[(clothes.tmin <= temp\_min) & (clothes.tmax +1 >= temp\_min)].index   if np.isnan(tempp) == True: # 결측값으로 인해 체감온도가 nan인 경우  clothes\_avgind = clothes[(clothes.tmin <= temp) & (clothes.tmax +1 >= temp)].index  else: # 체감온도가 잘 계산된 경우  clothes\_avgind = clothes[(clothes.tmin <= tempp) & (clothes.tmax +1 >= tempp)].index    # 큰 일교차 == 최고기온의 range와 최저기온의 range index가 3 이상 차이 날 때 (약 10도 차이)  if abs(clothes\_maxind.values[0] - clothes\_minind.values[0]) >= 3 :  print('일교차가 크므로, 얇게 여러 겹을 껴입는 것을 추천해요.\n최저 및 최고기온에 따른 옷차림을 추천합니다.')   print('\n최고기온: '+str(temp\_max)+'도')  # 추천 옷차림 출력  for i in range(len(clothes\_col)):  if str(clothes\_np[clothes\_maxind[0]][i+2]) != 'nan': # 저장된 옷차림 정보가 있는 경우만 출력   print(str(clothes\_col[i])+' : '+str(clothes\_np[clothes\_maxind[0]][i+2]))  else:  continue # nan값인 경우 출력하지 않고 넘어감   print('\n최저기온: '+str(temp\_min)+'도')  for i in range(len(clothes\_col)):  if str(clothes\_np[clothes\_minind[0]][i+2]) != 'nan': # 저장된 옷차림 정보가 있는 경우만 출력   print(str(clothes\_col[i])+' : '+str(clothes\_np[clothes\_minind[0]][i+2]))  else:  continue # nan값인 경우 출력하지 않고 넘어감    elif np.isnan(tempp) == True: # 체감기온 계산 불가 --> 평균기온 이용해 추천 옷차림 출력  print('평균기온 '+str(round((temp\_min+temp\_max)/2,1))+'도에 적절한 옷차림을 추천합니다.\n')  for i in range(len(clothes\_col)):  if str(clothes\_np[clothes\_avgind[0]][i+2]) != 'nan':  print(str(clothes\_col[i])+' : '+str(clothes\_np[clothes\_avgind[0]][i+2]))  else:  continue   else: # 체감기온 이용해 추천 옷차림 출력  print('체감기온 '+str(tempp)+'도에 적절한 옷차림을 추천합니다.\n')  for i in range(len(clothes\_col)):  if str(clothes\_np[clothes\_avgind[0]][i+2]) != 'nan': # 저장된 옷차림 정보가 있는 경우만 출력   print(str(clothes\_col[i])+' : '+str(clothes\_np[clothes\_avgind[0]][i+2]))  else:  continue # nan값인 경우 출력하지 않고 넘어감 |
| --- |

* 1. 선호 테마 선택

| # 선호 여행테마 선택  def q\_theme():   # 내가 선택한 지역의 테마별 관광지 개수  area = spot[spot.City == trip\_city]  theme\_num = np.arange(6)  for i in range(6):  theme\_num[i] = len(area[area['ThemeCode'] == i+1])   theme\_types = ['종교/역사/전통', '체험/학습/산업', '자연/힐링', '문화/예술', '쇼핑/놀이', '캠핑/스포츠']  dic = { y:x for x, y in zip(np.arange(6), theme\_num) }  plt.figure(figsize=(10,6))  plt.rc('font', family = 'NanumBarunGothic')  plt.title(trip\_city + "의 테마별 관광지 개수")  plt.bar(np.arange(6), theme\_num) # 선택한 지역의 테마별 관광지 개수 막대 그래프   for i in np.arange(6): # 그래프에 값 추가  plt.text(i, theme\_num[i], str(theme\_num[i]),   horizontalalignment='center',  verticalalignment='bottom')   plt.xticks(np.arange(6), theme\_types) # x축 값 이름 변경  plt.show()    # 테마 선택을 위한 보기 출력  print(''' <<테마를 선택해주세요>>  [1] 종교/역사/전통 [2] 체험/학습/산업 [3] 자연/힐링 [4] 문화/예술 [5] 쇼핑/놀이 [6] 캠핑/스포츠\n''')    #선호 테마 저장을 위한 배열 선언  theme\_input = []  flag = 1  # 선호 테마 3순위까지 선택  for i in range(3):  while(flag):  theme\_i = int(input("\n%d번째로 선호하는 테마 번호를 입력해주세요 : \n" %(i+1))) # 선호 테마 입력 받기  if theme\_i not in range(0,7):  print("잘못된 입력입니다. 다시 입력해주세요") # 1~6 이외의 값을 넣으면 다시 입력 받기  elif theme\_i in theme\_input:  print('이미 선택한 테마입니다. 다른 테마를 선택해주세요. 더이상 선호테마가 없으면 0을 입력해주세요. 2번째 테마 선택 시 0을 입력하면 1순위 테마만 적용됩니다.') # 이미 선택한 테마를 입력하면 다시 입력 받기  elif theme\_i == 0:  if i == 0: # 선호 테마를 한 개도 고르지 않으면 다시 입력 받기  print('선호 테마는 최소 1가지 이상이어야 합니다.')  continue  else:  flag = False # 0을 입력하면 그만 입력 받기  for j in range(i,3):  theme\_input.append(theme\_input[i-1])   break  elif theme\_num[theme\_i-1] == 0:  print('해당 테마의 추천 관광지가 없습니다. 다른 테마를 선택해주세요.') # 선택한 테마에 추천 관광지가 없으면 다시 입력 받기  else:  theme\_input.append(theme\_i) # theme\_input에 선호 테마 저장  break  print('테마 선택을 종료합니다.\n')    return theme\_input |
| --- |

* 1. 선호 테마 내 여행지 시각화

| def select\_theme():  global spot\_fav\_first  global spot\_fav\_second  global spot\_fav\_third   # 선호 테마 순으로 3개 입력받아오기  theme\_input = q\_theme()   # 선호테마 순으로 여행갈 지역의 여행지 출력  spot\_fav\_first = [] # 1순위 여행테마의 여행지  spot\_fav\_second = [] # 2순위 여행테마의 여행지  spot\_fav\_third = [] # 3순위 여행테마의 여행지   for i in range(3):   # print('%s 지역의 %d번째 선호 테마 관광지 정보를 출력할게요' % (city\_input, i+1))  spot\_fav\_i = list(spot[(spot['City']==city\_input) & (spot['ThemeCode'] == theme\_input[i])]['Spot']) # 사용자가 선택한 지역명와 테마의 여행지를 리스트로 저장하기  if i == 0:   spot\_fav\_first = spot\_fav\_i # 1순위 여행테마의 여행지를 spot\_fav\_first 변수에 저장  elif i == 1:  spot\_fav\_second = spot\_fav\_i # 2순위 여행테마의 여행지를 spot\_fav\_second 변수에 저장  elif i == 2:  spot\_fav\_third = spot\_fav\_i # 3순위 여행테마의 여행지를 spot\_fav\_third 변수에 저장  # print(spot\_fav\_i, '\n')    ### 입력받은 관광지 지도에 출력   # 시작위치를 사용자 선택 지역으로 설정 ## 지도에 색, 순위(숫자) 표시  if spot\_fav\_first != []: # spot\_fav\_first가 비어있지 않은 경우  for i in range(len(spot)):   if spot['Spot'][i] in spot\_fav\_first: # spot\_fav\_first의 spot DataFrame의 요소와 같다면 그 행 뽑아 위경도 저장한 후 지도에 저장  lat = spot['Lat'][i]  lon = spot['Lon'][i]  m = folium.Map(location = [lat, lon], zoom\_start = 11)  break # spot의 첫번째 원소의 위경도 대표로 뽑은 후 break  else: # spot\_fav\_first가 비어있는 경우, spot\_fav\_second로 대체  for i in range(len(spot)):  if spot['Spot'][i] in spot\_fav\_second:  lat = spot['Lat'][i]  lon = spot['Lon'][i]  m = folium.Map(location = [lat, lon], zoom\_start = 11)  break   for i in range(len(spot)):  if (spot['Spot'][i] in spot\_fav\_first): # spot\_fav\_first이 있는 행 == i행  lat = spot['Lat'][i] # spot\_fav\_first 원소의 위도, 경도 값, 이름 각각 저장 후 지도에 추가  lon = spot['Lon'][i]  place\_map = spot['Spot'][i]  folium.Marker([lat,lon],tooltip = place\_map, icon=plugins.BeautifyIcon(icon="arrow-down",icon\_shape="marker", border\_color='#FAFCFD', background\_color='#C92519', number=1)).add\_to(m)   # 1순위 테마의 여행지 빨간색,'1'으로 표시  # m 지도에 추가   elif (spot['Spot'][i] in spot\_fav\_second): # spot\_fav\_second이 있는 행 -> i행  lat = spot['Lat'][i] # spot\_fav\_second 원소의 위도, 경도 값, 이름 각각 저장 후 지도에 추가  lon = spot['Lon'][i]  place\_map = spot['Spot'][i]  folium.Marker([lat,lon],tooltip = place\_map, icon=plugins.BeautifyIcon(icon="arrow-down",icon\_shape="marker", border\_color='#FAFCFD', background\_color='#FA7A35', number=2)).add\_to(m)   # 2순위 테마의 여행지 주황색,'2'으로 표시  # m 지도에 추가   elif (spot['Spot'][i] in spot\_fav\_third): # spot\_fav\_third이 있는 행 -> i행  lat = spot['Lat'][i] # spot\_fav\_third 원소의 위도, 경도 값, 이름 각각 저장 후 지도에 추가  lon = spot['Lon'][i]  place\_map = spot['Spot'][i]  folium.Marker([lat,lon],tooltip = place\_map, icon=plugins.BeautifyIcon(icon="arrow-down",icon\_shape="marker", border\_color='#FAFCFD', background\_color='#FAE500', number=3)).add\_to(m)   # 3순위 테마의 여행지 노란색,'3'으로 표시  # m 지도에 추가     return m # 1,2,3 순위 테마의 여행지를 m 지도의 marker로 나타내기 |
| --- |

* 1. 선호 테마별 관광지 선택

| def select\_spot():   place\_final\_first = [] # 1번째 선호 테마의 관광지 중 사용자가 선택한 관광지 저장할 리스트  place\_final\_second = [] # 2번째 선호 테마의 관광지 저장할 리스트  place\_final\_third = [] # 3번째 선호 테마의 관광지 저장할 리스트  place\_final = [] # 사용자가 선택한 최종 여행지를 저장할 리스트   print("\n1번째 선호 테마의 관광지 :") # 1번째 선호 테마의 관광지 출력  for i in range(len(spot\_fav\_first)): # 1번째 선호 테마의 관광지 개수 만큼  print("%d. %s" % (i+1, spot\_fav\_first[i]), end = ' ') # 각 관광지명에 번호 대응  print("\n")   if len(spot\_fav\_first) > 0: # spot\_fav\_first의 원소가 1개 이상일 때 입력받기  flag = True  while(flag):  place\_temp = [] # place\_final\_first에 포함되지 않는 수 하나라도 있을 시 리스트 초기화 후 다시 실행  place\_final\_input = input("1번째 선호 테마의 관광지 중 여행 희망 장소 번호를 선택하세요. 띄어쓰기로 구분하시고, 끝내려면 0을 입력하세요. ( ex: 1 2 )\n")  place\_final\_first = place\_final\_input.split(' ') # [4,5,8]  for i in place\_final\_first:  if int(i) not in range(len(spot\_fav\_first)+1): # int(i)가 spot\_fav\_first에 존재하지 않는 경우  print("[ %d ]는 존재하지 않는 장소입니다." % int(i)) # 문구 출력  flag = True # while문 다시 실행   elif int(i) == 0: # 사용자가 0을 입력했을 경우  flag = False # while문 끝내기  else: # int(i)가 spot\_fav\_first에 존재할 경우  place\_temp.append(spot\_fav\_first[int(i)-1]) # place\_temp 리스트에 임시저장  flag = False # while문 끝내기  place\_final.extend(place\_temp) # 모두 알맞은 수를 입력한 경우(모두 place\_final\_first의 원소인 경우) place\_temp에 있던 요소 place\_final 리스트에 저장    if spot\_fav\_first != spot\_fav\_second:  print("\n2번째 선호 테마의 관광지 :") # 2번째 선호 테마의 관광지 출력  for i in range(len(spot\_fav\_second)):  print("%d. %s" % (i+1,spot\_fav\_second[i]), end = ' ')  print("\n")   if len(spot\_fav\_second) > 0: # spot\_fav\_second의 원소가 1개 이상일 때 입력받기  flag = True  while(flag):  place\_temp = []  place\_final\_input = input("2번째 선호 테마의 관광지 중 여행 희망 장소 번호를 선택하세요. 띄어쓰기로 구분하시고, 끝내려면 0을 입력하세요. ( ex: 1 2 )\n")  place\_final\_second = place\_final\_input.split(' ') # [4,5,8]  for i in place\_final\_second:  if int(i) not in range(len(spot\_fav\_second)+1):  print("[ %d ]는 존재하지 않는 장소입니다." % int(i))  flag = True  elif int(i) == 0:  flag = False  else:  place\_temp.append(spot\_fav\_second[int(i)-1])  flag = False  place\_final.extend(place\_temp)   if spot\_fav\_second != spot\_fav\_third:  print("\n3번째 선호 테마의 관광지 :") # 3번째 선호 테마의 관광지 출력  for i in range(len(spot\_fav\_third)):  print("%d. %s" % (i+1,spot\_fav\_third[i]), end = ' ')  print("\n")    if len(spot\_fav\_third) > 0: # spot\_fav\_third의 원소가 1개 이상일 때 입력받기  flag = True  while(flag):  place\_temp = []  place\_final\_input = input("3번째 선호 테마의 관광지 중 여행 희망 장소 번호를 선택하세요. 띄어쓰기로 구분하시고, 끝내려면 0을 입력하세요. ( ex: 1 2 )\n")  place\_final\_third = place\_final\_input.split(' ') # [4,5,8]  for i in place\_final\_third:  if int(i) not in range(len(spot\_fav\_third)+1):  print("[ %d ]는 존재하지 않는 장소입니다." % int(i))  flag = True  elif int(i) == 0:  flag = False  else:  place\_temp.append(spot\_fav\_third[int(i)-1])  flag = False  place\_final.extend(place\_temp)  place\_final = list(set(place\_final))  print("최종 선택 여행지 : ", place\_final) # 1,2,3번째 선호 테마의 관광지 중 여행 희망 장소 place\_final 리스트에 저장하여 출력   return place\_final |
| --- |

* 1. 주차장 정보 필요 여부 입력

| global parking\_input, place\_final  def parking\_system():  global place\_final   while(1):  parking\_input = int(input("\n주차장 정보가 필요하십니까?\n [1] 네\n [2] 아니오\n" ))   if parking\_input not in (1,2):  print('잘못된 입력입니다. 다시 입력해 주세요')  elif parking\_input == 1 :   # 관광지와 주차장 정보가 저장된 지도 받아오기  parking\_info(place\_final)  break  else:  print('프로그램을 종료합니다.\n')  break |
| --- |

* 1. 주차장 정보 출력, 시각화

| #주차장 정보 불러오는 함수 def parking\_info(place\_final):   ##전국 주차장 파일에서 입력받은 도시와 같은 도시만 출력##  arrival\_city = parking\_lot.loc[parking\_lot['city'] == city\_input]  parking\_spot = []  parking\_show = []    ##관광지의 이름과 위도, 경도 데이터프레임으로 뽑아오기##  for loc\_parking in place\_final:  for i in spot.index[spot.iloc[:, 6] == loc\_parking]: # 6,7,8열  parking\_spot.append(spot.iloc[i, 6:9])  parking\_spot = pd.DataFrame(parking\_spot)   ######## 주차장 지도 출력  spot\_map = spot[spot['City'] == city\_input].iloc[0] # 사용자가 선택한 지역의 위경도 받은 후 지도 저장  lat = spot\_map[-3]   lon = spot\_map[-2]  m = folium.Map(location = [lat, lon], width = 800, height = 600, zoom\_start = 12)    ##관광지와 주차장 간 거리 계산##  for i in range(len(parking\_spot)):  templist = []  tempdis = []  min\_dis = 999.0  min\_info = []   for j in range(len(arrival\_city)):  temp2 = parking\_spot.iloc[i]['Lat'], parking\_spot.iloc[i]['Lon']  temp1 = (arrival\_city.iloc[j]['위도'],arrival\_city.iloc[j]['경도'])  dis = haversine(temp1, temp2)   if min\_dis > dis:  min\_info.append(arrival\_city.iloc[j])  min\_dis = dis  if dis < 1 :  templist.append(arrival\_city.iloc[j])  tempdis.append(round(dis,2))   print("\n [", parking\_spot.iloc[i]['Spot'], "인근의 주차장 정보]")   ##계산된 거리가 1km보다 멀면 근처에 주차장이 없음을 출력한다##  if len(templist) == 0:  print("※1km 이내에 주차장이 없어서, 가장 가까운 주차장 정보를 출력합니다.")  min\_parking = pd.DataFrame(min\_info)[['주차장명', '요금정보', '소재지지번주소', '평일', '토요일', '공휴일']].iloc[-1]  min\_parking['거리[km]'] = round(min\_dis,2)  min\_parking = pd.DataFrame(min\_parking).T  print(tabulate(min\_parking, headers='keys', tablefmt='psql', showindex=False))  parking\_show.append(min\_info[-1])   ##계산된 거리가 1km보다 작으면 해당 주차장을 출력한다##   else:  final\_parking = pd.DataFrame(templist)[['주차장명', '요금정보', '소재지지번주소', '평일', '토요일', '공휴일']]  final\_parking['거리[km]'] = tempdis  final\_parking = final\_parking.sort\_values(by=['거리[km]'], ascending=True)  if len(final\_parking) > 5:  print\_len = 5  else:  print\_len = len(final\_parking)  print(tabulate(final\_parking[:print\_len], headers='keys', tablefmt='psql', showindex=False))  parking\_show.extend(templist)   parking\_latlon = pd.DataFrame(templist)#[['주차장명','위도','경도']]  parking\_latlon['거리[km]'] = tempdis  parking\_latlon = parking\_latlon.sort\_values(by=['거리[km]'], ascending=True)[:print\_len]   for k2 in range(len(parking\_latlon)): # 인근 주차장 목록  lat = parking\_latlon['위도'].iloc[k2]   lon = parking\_latlon['경도'].iloc[k2]  place\_map = parking\_latlon['주차장명'].iloc[k2]  parkinglot\_info = '평 일: '+parking\_latlon['평일'].iloc[k2]+'\n토요일: '+parking\_latlon['토요일'].iloc[k2]+'\n공휴일: '+parking\_latlon['공휴일'].iloc[k2]  folium.Marker([lat,lon], tooltip = place\_map, popup=parkinglot\_info,  icon=folium.Icon(color='blue', icon = 'car', prefix='fa')).add\_to(m) # 주차장 - 파란색 자동차 마커로 지도에 저장   for k1 in range(len(parking\_spot)): # 관광지 목록  lat = parking\_spot['Lat'].iloc[k1]  lon = parking\_spot['Lon'].iloc[k1]  place\_map = parking\_spot['Spot'].iloc[k1]  folium.Marker([lat,lon], tooltip = place\_map, icon=folium.Icon(color='red', icon = 'star')).add\_to(m) # 최종 선택 관광지 - 빨간색 별 마커로 지도에 저장  print('')   display(m) # 최종 관광지 및 인근 주차장 지도 출력 |
| --- |

1. ‘메인 프로그램’ 을 실행

| 1. 출발지 입력 2. 여행 날짜 입력 3. 여행지 선택 여부 입력 4. 여행지 입력 5. 선호 테마 입력 6. 원하는 관광지 선택 7. 주차장 정보 필요 여부 입력 8. 결과 확인 |
| --- |

| ## main flow  # start print('여행코스 추천 프로그램입니다.')  # 출발 지역 입력 area\_input = select\_spoint()  # 여행 날짜 입력, 기상데이터 불러오기 # 여행지 선택 여부 /지역 추천 trip\_city = where\_to\_trip()  # 날씨 정보 제공, 옷차림 추천 forecast(tripdate, diffdate)  # 선호 테마 입력, 해당 관광지 출력 select\_theme()  # 원하는 관광지 선택 place\_final = select\_spot()  # 주차장 정보 제공 parking\_system()  # end print('\n프로그램을 종료합니다. 즐거운 여행 되세요!') |
| --- |

1. 요구 조건 반영 여부
2. 정보 제공 메뉴 (5개 이상)
3. 여행 테마별 관광지 : 여행지를 정하지 못한 사용자를 위해 체험, 문화 등의 여행 테마를 제시하고 사용자가 선택한 테마에 해당하는 관광지를 안내한다.
4. 여행 테마별 관광지 개수 그래프 : 사용자가 선택한 테마에 해당하는 관광지 개수를 막대 그래프로 안내한다.
5. 여행 날의 날씨 정보 : 사용자가 선택한 날의 기온, 강수확률, 하늘 상태 등의 정보를 제공한다. 여행 날짜가 3일 이내라면 1시간 간격의 강수 확률 예보 그래프를 제공하여 상세한 일기 예보를 안내한다
6. 기상 데이터를 기반으로 한 추천 옷차림 정보 : 기상 데이터를 바탕으로 상의, 하의, 신발, 옷의 소재 등 다양한 옷차림 정보를 추천한다.
7. 여행지 근처의 공영주차장 정보 : 여행지 근처의 공영주차장 위치, 운영요일 등을 제공하여 사용자에게 편의를 제공한다.
8. pandas 기능 (2개 이상)
9. pandas.read\_excel
10. pandas.DataFrame.to\_numpy
11. pandas.DataFrame
12. numpy 기능 (2개 이상)
13. numpy.arange
14. numpy.isnan
15. numpy.unique
16. numpy.array
17. matplotlib 기능 (5개 이상)
18. plt.bar
19. plt.xticks
20. plt.figure
21. plt.rc
22. plt.title
23. plt.plot
24. plt.xlabel / plt.ylabel
25. plt.show
26. 그 외 사용한 주요 라이브러리 및 모듈 설명
27. csv : csv 파일을 읽고 쓰기 위해 사용하는 라이브러리로, csv 파일 데이터를 읽기 위해 사용한다.
28. math : 수학 관련 함수들이 들어 있는 라이브러리로, 위경도를 x, y 좌표로 변환하기 위해 사용한다.
29. geopy, haversine : 위경도를 이용하여 두 지역간의 거리차를 구하기 위해 사용하는 라이브러리로, 출발지에서 가까운 여행지 10개를 출력할 때, 관광지 인근의 주차장 정보를 출력할 때 사용한다.
30. time, datetime, pytz : 날짜와 시간을 계산할 수 있는 함수들을 제공하는 라이브러리로, 입력 받은 날짜가 얼마나 먼 미래인지를 계산하기 위해 사용한다.
31. requests : HTTP 통신에 사용되는 라이브러리로, 주로 API를 호출할 때 사용된다. 기상청 날씨 예보 API를 파싱하기 위해 사용한다.
32. json : json 형태의 데이터를 처리하기 위해 사용되는 라이브러리로, 기상청 날씨 예보 API 정보를 파싱하기 위해 사용한다.
33. folium : 지도 시각화 라이브러리로, 위경도 좌표를 기준으로 지도에 지점을 표시할 수 있다. 이를 이용하여 사용자가 선택한 테마의 관광지를 지도에 표시한다.
34. tabulate : 표 형식의 데이터를 가독성 좋게 출력하기 위해 사용하는 모듈이다.
35. 팀원별 담당 부분

| **팀원 이름** | **파이썬 프로그램 구현** | **Report** | **발표** | **데모 비디오** | **기타** |
| --- | --- | --- | --- | --- | --- |
| **문주원** | - 출발 지역 입력 코드 구현  - 여행지 선택 및 추천 코드 구현  - 여행 날짜 입력 코드 구현  - 날짜 분류 및 과거 날씨 정보 불러오는 코드 구현  - 날씨정보 제공 (체감기온 /일교차 등), 옷차림 추천 코드 구현 | - 문제 정의, 데이터 수집 방법 작성  - 레포트 수정 및 보강 | - 중간 발표 ppt 제작  - 최종 발표자  -최종 발표 대본 작성 | -PPT 녹화 | - .xlsx 파일 전처리 (엑셀 내장 함수 및 pandas 사용)  - 전체 코드 함수화 및 메인 프로그램 코드 작성 : 변수 통일, 전역변수 처리, 에러 수정 등  -flow 차트 제작 |
| **이하경** | - 주차장 정보 출력 함수 코드 구현 | - 레포트 수정 및 보강 |  |  | - readme파일 작성 |
| **염다빈** | - 선호 여행테마 입력하는 함수  -주차장 정보 출력 함수 코드 구현 | - 전체 레포트 작성 | - 중간 발표 발표자  - 최종 발표 ppt 제작 |  | - flow 차트 제작  - 주차장 정보 지도에 출력 코드 도움  - 에러 수정 |
| **민황선** | - 날씨 데이터 불러오는 부분 : 기상청 API 파싱, 데이터 전처리, 지역별, 날짜별 날씨 정보 출력 코드 구현 | - 레포트 수정 및 보강 | - 최종 발표 대본 작성  - 최종 시연 | - 시연 녹화  - 영상 편집 |  |
| **송은채** | - 테마 선택하고, 관광지 선택하는 함수 코드 구현 : 여행테마와 관광지 선택하고 지도에 출력, 주차장 정보 지도에 출력 | - 전체 레포트 작성 | -최종 발표 대본 작성 |  |  |