포팅 매뉴얼 : YOUFOOT

맞춤형 산책로 추천 서비스

> ### SSAFY 10기 특화 프로젝트 구미 2반 7팀

0. 목차

1. 개발 환경

1. 프로젝트 기술 스택

- **Frontend**
  - Visual Studio Code(IDE) `1.81.1`
  - HTML5, CSS3, Javascript(ES6)
  - React : `18.2.0`
  - Vite `5.1.6`

    - Typescript `5.2.2`
    - Zustand `4.5.2`
    - Tanstack Query `5.28.4`
    - emotion CSS `11.11.4`
    - Nodejs `20.12.0`

- **Backend**
    - Intellij : `2023.3.2`
    - JVM OpenJDK : `17`
    - JWT : `0.11.5`
    - Spring Boot : `3.0.13`
    - JAVA Spring Data JPA
    - Spring Security
    - SSEEmitter
    - OAuth : `6.0.8`
    - Lettuce : `6.2.7`
    - Gradle
    - ORM : JPA
    - Python `3.8.10`
    - Fastapi `0.110.0`
    - Skilearn `1.3.2`
- **Bigdata**
    - Hadoop `3.3.6`
    - Spark `3.4.2`
    - pymysql : `1.4.6`
    - pandas : `2.2.1`
- **CI/CD**
    - AWS EC2

        - Nginx : `1.18.0`
        - Ubuntu : `20.04 LTS`
        - Docker : `25.0.4`
        - Jenkins : `2.443`
    - Docker Hub

2. 환경변수 설정

Frontend : .env

```
# "VITE_"를 붙여야 합니다!

# https://developers.kakao.com/console/app -> JavaScript 키
# 도메인 추가: http://localhost:5173 ,  http://j10d207.p.ssafy.
VITE_KAKAO_REST_API_KEY='387a65778152cfac269066d65fc23ab8'
VITE_KAKAO_JAVASCRIPT_API_KEY='224134797224c8b5cb93562eb0157d

#####################################################################
VITE_KAKAO_REDIRECT_URI='https://j10d207.p.ssafy.io/oauth/cal
VITE_API_BASE_URL='https://j10d207.p.ssafy.io'
VITE_API_BASE_NEXT_URL='/api/oauth/callback/kakao/token/d-t-d
VITE_OPEN_WEATHER_API_KEY='5b96a8599ca7541b43cc6a600cf99787'
API_BASE_URL='https://j10d207.p.ssafy.io'
```

Backend : application 파일

- application.yml

```
spring:
  profiles:
    active: dev
  servlet:
    multipart:
      enabled: true # 멀티파트 업로드 지원여부 (default: true)
```

```
      max-file-size: 10MB
      max-request-size: 10MB
  data:
    redis:
      host: 127.0.0.1
      port: 6379
      lettuce:
        pool:
          max-active: 100 # pool에 할당될 수 있는 최대 커넥션 수
          max-idle: 10   #  pool에 할당 될 수 있는 최대 idle 커넥션
          min-idle: 2    #  pool에서 관리하는 최소 idle 커넥션 대상

logging:
  level:
    org:
      apache:
        http: DEBUG
    httpclient:
      wire: DEBUG
kakao:
  clientId : ${spring.kakao.client-id}
  secret: ${spring.kakao.client-secret}

cloud:
  aws:
    credentials:
      access-key: ${aws.s3.accesskey}
      secret-key: ${aws.s3.secretkey}
    region:
      static: ap-northeast-2
    stack:
      auto: false
    s3:
      bucket: ssafys3
```

- application-dev.yml

```
spring:
  jpa:
    database: mysql
    show-sql: true
    hibernate:
      ddl-auto: validate
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://database-1.ctee4gmysdpo.ap-northeast-2.
    username: ${rds.username}
    password: ${rds.password}
server:
  servlet:
    encoding:
      force: 'true'
      enabled: 'true'
      charset: UTF-8
    context-path: /
  port: '8080'
```

3. 설정 파일

Nginx : nginx.conf

- 설정 파일 위치

```
/etc/nginx/nginx.conf
/etc/nginx/sites-enabled/default
```

- nginx.conf

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
```

```
events {
        worker_connections 768;
}

http {

        client_max_body_size 10M;
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        server {
                listen 80;
                server_name temp;

                return 301 https://$host$request_uri;
        }

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Droppi
        ssl_prefer_server_ciphers on;

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        gzip on;


        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

- sites-enabled/default

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /var/www/html;

        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
                proxy_connect_timeout 300s;
                proxy_read_timeout 600s;
                proxy_send_timeout 600s;
                proxy_buffers 8 16k;
                proxy_buffer_size 32k;

                proxy_pass http://localhost:5173;
                proxy_http_version 1.1;
                proxy_set_header Host $host;
        }

        location /api/ {
                proxy_pass http://localhost:8080/api/;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x
                proxy_set_header X-Forwarded-Proto $scheme;
                proxy_set_header X-Forwarded-Port $server_por
                proxy_set_header X-Forwarded-Prefix /api;
        }
}


server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name j10d207.p.ssafy.io;
```

```
    ssl_certificate /etc/letsencrypt/live/j10d207.p.ssafy.io/
    ssl_certificate_key /etc/letsencrypt/live/j10d207.p.ssafy
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://localhost:5173;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
    }

    location /api/ {
        proxy_pass http://localhost:8080;  # 백엔드 서비스로 리다
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Port $server_port;
    }
}
server {
        listen 80 ;
        listen [::]:80;
        server_name j10d207.p.ssafy.io;
        return 301 https://$host$request_uri;
}
~
```

Jenkins : Jenkinsfile

- 설정 파일 위치 : 프로젝트 최상단
- Jenkinsfile

```
pipeline {
    agent any

    stages {
```

```
        stage('MM-Alarm'){
            steps{
                script {
                    def Author_ID = sh(script: "git show -s -
                    def Author_Name = sh(script: "git show -s
                    mattermostSend (
                        color: '#D0E0E3',
                        icon: "https://jenkins.io/images/logo
                        message: "파이프라인 시작: ${env.JOB_NAM
                    )
                }
            }
        }


        stage('Clone') {
            steps {
                echo '클론을 시작!'
                git branch: 'dev', credentialsId: 'youfoot',
                echo '클론을 완료!'
            }
        }

        stage('BE-Build') {
            steps {
                echo '백엔드 빌드 및 테스트 시작!'
                dir("./backend/ssafy_sec_proj") {
                    sh "ls"
                    sh "chmod +x ./gradlew"

                    // sh "touch ./build.gradle"

                    // application properties 파일 복사
                    // sh "echo $BuildGradle > ./build.gradle

                    sh "./gradlew clean build --exclude-task

                }
```

```
                echo '백엔드 빌드 및 테스트 완료!'
            }
        }

        stage('Build Back Docker Image') {
            steps {
                echo '백엔드 도커 이미지 빌드 시작!'
                dir("./backend/ssafy_sec_proj") {
                    // 빌드된 JAR 파일을 Docker 이미지로 빌드
                    sh "docker build -t gungssam/youfootbe:la
                }
                echo '백엔드 도커 이미지 빌드 완료!'
            }
        }

        stage('Push to Docker Hub-BE') {
            steps {
                echo '백엔드 도커 이미지를 Docker Hub에 푸시 시작!'
                withCredentials([usernamePassword(credentials
                    sh "docker login -u $DOCKER_USERNAME -p $
                }
                dir("./backend/ssafy_sec_proj") {
                    sh "docker push gungssam/youfootbe:latest
                }
                echo '백엔드 도커 이미지를 Docker Hub에 푸시 완료!'
            }
        }

        stage('Deploy to EC2-BE') {
            steps {
                echo '백엔드 EC2에 배포 시작!'
                // 여기에서는 SSH 플러그인이나 SSH 스크립트를 사용하여
                sshagent(['aws-key']) {
                    sh "docker rm -f backend"
                    sh "docker rmi gungssam/youfootbe:latest"
                    sh "docker image prune -f"
                    sh "docker pull gungssam/youfootbe:latest
                }
```

```
                echo '백엔드 EC2에 배포 완료!'
            }
        }

        stage('FE-Build') {
            steps {
                echo '프론트 빌드 및 테스트 시작!'
                dir("./frontend") {
                    sh "npm install --legacy-peer-deps"
                    sh "npm run build"
                }
                echo '프론트 빌드 및 테스트 완료!'
            }
        }

        stage('Build Front Docker Image') {
            steps {
                echo '프론트 도커 이미지 빌드 시작!'
                dir("./frontend") {
                    // 빌드된 파일을 Docker 이미지로 빌드
                    sh "docker build -t gungssam/youfootfe:la
                }
                echo '프론트 도커 이미지 빌드 완료!'
            }
        }

        stage('Push to Docker Hub-FE') {
            steps {
                echo '프론트 도커 이미지를 Docker Hub에 푸시 시작!'
                withCredentials([usernamePassword(credentials
                    sh "docker login -u $DOCKER_USERNAME -p $
                }
                dir("frontend") {
                    sh "docker push gungssam/youfootfe:latest
                }
                echo '프론트 도커 이미지를 Docker Hub에 푸시 완료!'
            }
        }
```

```
        stage('Deploy to EC2-FE') {
            steps {
                echo '프론트 EC2에 배포 시작!'
                // 여기에서는 SSH 플러그인이나 SSH 스크립트를 사용하여
                sshagent(['aws-key']) {
                    sh "docker rm -f frontend"
                    sh "docker rmi gungssam/youfootfe:latest"
                    sh "docker image prune -f"
                    sh "docker pull gungssam/youfootfe:latest
                }
                echo '프론트 EC2에 배포 완료!'
            }
        }

    }

    post {
        success {
            echo '파이프라인이 성공적으로 완료되었습니다!'
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend (
                    color: '#D0E0E3',
                    icon: "https://jenkins.io/images/logos/je
                    message: "빌드 성공: ${env.JOB_NAME} #${env
                )
            }
        }
        failure {
            echo '파이프라인이 실패하였습니다. 에러를 확인하세요.'
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend (
                    color: '#D0E0E3',
                    icon: "https://4.bp.blogspot.com/-52EtGjE
```

```
                    message: "빌드 실패: ${env.JOB_NAME} #${env
                )
            }
        }
    }
}
~
```

Dockerfile : Backend

- 설정 파일 위치

```
/backend/Dockerfile
/frontend/Dockerfile
```

- dockerfile

```
FROM openjdk:17-jdk-alpine
EXPOSE 8080
VOLUME /tmp
ADD ./build/libs/ssafy_sec_proj-0.0.1-SNAPSHOT.jar app.jar
ENV JAVA_OPTS=""
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Dockerfile : Frontend

```
FROM node:alpine as builder
WORKDIR /frontend
COPY package.json .
# RUN npm install
COPY ./ ./
RUN npm run build

# 3000번 포트 노출
EXPOSE 5173
```

```
# npm start 스크립트 실행
CMD ["npm", "run", "dev"]:
```

2. 배포 방법

1. EC2 내부 방화벽 설정

포트 허용

```
sudo ufw allow 8080/tcp # ssh는 tcp 프로토콜만 허용해야함
sudo ufw status  # 방화벽 포트 상태 확인
sudo ufw deny 8080/tcp
```

2. 도커 설치

업데이트 및 HTTP 패키지 설치

```
sudo apt update
sudo apt-get install -y ca-certificates \
 curl \
 software-properties-common \
 apt-transport-https \
 gnupg \
 lsb-release
```

GPG키 및 저장소 추가

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | s
udo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
keyrings/docker.gpg] https://download.docker.com/linux/ubun
tu \
```

```
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.lis
t.d/docker.list > /dev/null
```

설치 가능 버전 확인

```
apt-cache madison docker-ce
```

도커 설치

```
sudo apt update
sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-c
li=<VERSION_STRING> containerd.io
```

도커 확인

```
sudo docker run hello-world # 또는
sudo docker version
```

3. 젠킨스 설치

Docker 컨테이너에 마운트 할 볼륨 디렉토리 설치

```
cd /home/ubuntu && mkdir jenkins-dat a
```

포트 설정

```
sudo ufw allow 9090/tcp
sudo ufw reload
sudo ufw status
```

도커로 젠킨스 컨테이너 생성 및 구동

```
sudo docker run -d -p 9090:8080 -v /home/ubuntu/jenkins-dat
a:/var/jenkins_home -v /var/run/docker.sock:/var/run/docke
r.sock --name jenkins jenkins/jenkins:lts
```

- Docker out of Docker (DooD) 설정을 위한 볼륨 지정

초기 패스워드 확인

```
sudo docker logs jenkins
```

- 중간에 나오는 패스워드 확인

```
************************************************************
************************************************************
************************************************************
************************************************************
****************************
casjije2ij10ue9qwdjsvogh4893uq2ejoadncvhw23y89q
************************************************************
************************************************************
************************************************************
************************************************************
****************************
```

4. Git 설치 및 프로젝트 셋팅

Git 설치

```
sudo apt-get install git
sudo apt install git
```

Git 버전 확인

```
sudo git --version
```

Git 계정 설정

```
sudo git config --global user.name ${유저 이름}
sudo git config --global user.email ${유저 이메일}
```

프로젝트 Clone

```
sudo git clone ${클론 받을 깃 레포지토리 url} ${다운받아올 폴더명}
```

5. HTTPS 적용

80, 443 포트 방화벽 해제

```
sudo ufw allow 80
sudo ufw allow 80/tcp
sudo ufw allow 443
sudo ufw allow 443/tcp
sudo ufw enable
sudo ufw status
```

기본 라이브러리 설치

```
sudo apt-get install -y build-essential
sudo apt-get install curl
```

Nginx 설치 및 conf 파일 작성

```
sudo apt-get install nginx
sudo vi /etc/nginx/conf.d/nginx.conf
```

위에 쓴 Nginx.conf 파일 작성

Certbot 설치 및 SSL 인증서 발급

```
sudo apt-get remove certbot
sudo snap install --classic certbot
# nginx 자동 설정
sudo certbot --nginx
```

6. 자바 설치

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install openjdk-11-jdk
```

환경 변수 설정

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin
```

7. 하둡 설치

설치 과정

```
wget https://www.apache.org/dyn/closer.cgi/hadoop/common/hado
tar zxvf hadoop-3.3.6.tar.gz
mv hadoop-3.3.6 hadoop
```

설정 파일 위치

```
${HADOOP_Home}/etc/hadoop/core-site.xml
${HADOOP_Home}etc/hadoop/hdfs-site.xml
```

설정 파일

- core-site.xml

```
<configuration>
    <property>
```

```
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

- hdfs-site.xml

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

하둡 실행

```
${HADOOP_HOME}/sbin/start-dfs.sh
${HADOOP_HOME}/sbin/start-yarn.sh
```

하둡에 데이터 넣기

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import java.io.IOException;

public class MoveFileToHDFS {
    public static void main(String[] args) throws IOException
        // HDFS에 접속하기 위한 Configuration 객체 생성
        Configuration configuration = new Configuration();
        // HDFS의 기본 파일 시스템을 설정
        configuration.set("fs.defaultFS", "hdfs://localhost:9

        // HDFS에 연결하기 위해 파일 시스템 객체 생성
        FileSystem fs = FileSystem.get(configuration);
```

```
        // 로컬 파일 시스템의 파일 경로
        String localFilePath = "/home/ubuntu/hadoop/src/{cate
        // HDFS로 옮기기 위한 목적지 경로
        String hdfsFilePath = "/home/ubuntu/src/data/{categor

        // 로컬 파일을 HDFS로 복사
        Path localPath = new Path(localFilePath);
        Path hdfsPath = new Path(hdfsFilePath);
        fs.copyFromLocalFile(localPath, hdfsPath);

        // 파일 복사 후 파일 시스템 닫기
        fs.close();
    }
}
```

위 파일 카테고리 바꿔가면서 아래 코드 6번 실행

```
["toilet", "police", "restaurant", "cctv", "cafe", "convenien
javac -cp $HADOOP_HOME/share/hadoop/common/hadoop-common-3.3.
java -cp $HADOOP_HOME/share/hadoop/common/hadoop-common-3.3.6
```

8. 스파크 설치

설치 과정

```
wget https://downloads.apache.org/spark-3.4.2.bin-hadoop3.tgz
tar zxvf spark-3.4.2.bin-hadoop3.tgz
ln -s spark-3.4.2.bin-hadoop3 spark

혹은
pip3 install pyspark
```

환경 변수 및 스파크 설정

```
$SPARK_HOME/conf에서
cp spark-env.sh.template spark-env.sh

이후 spark-env.sh에서
SPARK_HOME=/home/ubuntu/spark
PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH
export PYSPARK_PYTHON=python

export JAVA_HOME=/usr/lib/jvm/jdk
export HADOOP_HOME=/home/ubuntu/hadoop
export SPARK_HOME=/home/ubuntu/spark
export SPARK_CONF_DIR=$SPARK_HOME/conf
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
PYSPARK_PYTHON=/usr/bin/python3
PYSPARK_DRIVER_PYTHON=/usr/bin/python3
SPARK_MASTER_IP=3.36.69.13
SPARK_MASTER_PORT=7077
```

스파크 실행

```
$SPARK_HOME/sbin/start-master.sh
$SPARK_HOME/sbin/start-worker.sh
```

9. python 코드

설정 파일 위치

```
mkdir $SPARK_HOME/code
$SPARK_HOME/code
sudo vi mainmain.py (아래 mainmain.py 파일 내용 입력)
sudo vi tempmain.py (아래 tempmain.py 파일 내용 입력)
```

fastapi 코드(mainmain.py)

```
from fastapi import FastAPI, HTTPException, Query
from pydantic import BaseModel
from typing import List
from pyspark.sql import SparkSession
from pyspark.sql.functions import lit

class Coordinate(BaseModel):
    latitude: float
    longitude: float

class CoordinatesData(BaseModel):
    data: List[Coordinate]

app = FastAPI()

# PySpark 세션 생성
spark = SparkSession \
    .builder \
    .appName("FacilityFinder") \
    .config("spark.hadoop.fs.defaultFS", "hdfs://j10d207a.p.s
    .config("spark.shuffle.service.enabled", "false") \
    .config("spark.dynamicAllocation.enabled", "false") \
    .getOrCreate()

categories = ["toilet", "police", "restaurant", "cctv", "cafe

@app.get("/data/{longitude}/{latitude}")
async def read_datas(latitude: float, longitude: float):
    try:
        results = {}
        print(latitude, longitude)
        for category in categories:
            results[category] = []

        for category in categories:
            file_path = f'hdfs://j10d207a.p.ssafy.io:9000/hom
            df = spark.read.csv(file_path, encoding='EUC-KR',
            filtered_df = df.filter((df['lat'] - latitude).be
```

```
            results[category] += filtered_df.toJSON().collect
            print(category, results[category])
        # 중복 값 제거: 각 카테고리의 결과를 집합으로 변환한 후 리스트로
        return results
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

fastapi코드(tempmain.py)

```
import joblib
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from fastapi.middleware.cors import CORSMiddleware
from fastapi.encoders import jsonable_encoder
from fastapi.responses import JSONResponse
import warnings
from sklearn.exceptions import ConvergenceWarning

# 경고를 무시하도록 설정
warnings.filterwarnings("ignore", category=UserWarning)


class UserData(BaseModel):
    cafe_num : int
    cctv_num :  int
    convenience_num : int
    police_num : int
    restaurant_num : int
    gender : str
    age_range :str
    prefer_duration_e : int
    prefer_duration_s : int
    sum_num : int

app = FastAPI()
```

```
# CORS 미들웨어 설정
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # 모든 출처 허용
    allow_credentials=True,
    allow_methods=["*"],  # 모든 HTTP 메서드 허용
    allow_headers=["*"],  # 모든 헤더 허용
)

@app.post("/data/predict-cluster")
async def predict_cluster(request_body : UserData):
    try:
        # 저장된 kmeans 모델 로드
        loaded_model = joblib.load('kmeans_model.pkl')

        # 스케일링 모델 로드
        loaded_scaler = joblib.load('kmeans_scaler.pkl')


        # 나이 라벨인코딩
        def trans_age(age_range):
            if age_range == "1~9":
                return 0
            elif age_range == "10~14":
                return 1
            elif age_range == "15~19":
                return 2
            elif age_range == "20~29":
                return 3
            elif age_range == "30~39":
                return 4
            elif age_range == "40~49":
                return 5
            elif age_range == "50~59":
                return 6
            elif age_range == "60~69":
                return 7
            elif age_range == "70~79":
```

```
                return 8
            elif age_range == "80~89":
                return 9
            elif age_range == "90~":
                return 10

        cate_age = trans_age(request_body.age_range)

        gender = 0 if request_body.gender == "male" else 1

        # 범위 지정해주어야 하는 컬럼 : 'prefer_duration_e', 'prefe
        def minmax_scaling(min, max, v):
            return (int(v) - min) / (max - min)

        scale_age = minmax_scaling(0, 10, cate_age)
        scale_prefer_duration_e = minmax_scaling(0, 10, reque
        scale_prefer_duration_s = minmax_scaling(0, 10, reque

        cafe_num, cctv_num, convenience_num, police_num, rest
            request_body.cafe_num / request_body.sum_num, req
            request_body.police_num / request_body.sum_num, r

        data_prediction = [[cafe_num, cctv_num,convenience_nu
scale_prefer_duration_s]]

        cluster = loaded_model.predict(data_prediction).tolis

        return JSONResponse(content=jsonable_encoder({"cluste

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

모델 학습하는 파이썬 코드(배치 분석) - 실행하면 모델 나온다.

- 아래 학습 모델을 실행해 나온 kmeans_model.pkl 파일을 ~/spark/code에 추가한다.

```
import pandas as pd
import pymysql

# MySQL 연결 설정
def connect_to_mysql():
    return pymysql.connect(
        host="{db host값}",
        port="{db port값}",
        user="{db user값}",
        password="{db password값}",
        database="{db 이름}",
        charset="utf8",
        connect_timeout=60  # 연결 시간 초과 값 설정
    )

# MySQL에 연결
conn = connect_to_mysql()

## trail 데이터 생성
# 커서 생성
cursor = conn.cursor()

# SQL 쿼리 실행
query = "SELECT id, eup_myeon_dong, si_do, si_gun_go, users Fl
cursor.execute(query)

# 결과 가져오기
result = cursor.fetchall()

columns = ["id", "eup_myeon_dong", "si_do", "si_gun_go", "use
trails = pd.DataFrame(result, columns=columns)

## 산책로 편의시설 수 데이터 생성
# 커서 생성
cursor = conn.cursor()

# SQL 쿼리 실행
query = "SELECT id, cafe_num, cctv_num, convenience_num, poli
```

```
cursor.execute(query)

# 결과 가져오기
result = cursor.fetchall()

columns = ["id", 'cafe_num', 'cctv_num', 'convenience_num', '
trails_facility = pd.DataFrame(result, columns=columns)

## user 데이터
query = "SELECT id, age_range, gender, visited_location, pref
cursor.execute(query)

# 결과 가져오기
result = cursor.fetchall()

columns = ["id", "age_range", "gender", "visited_location", "

users = pd.DataFrame(result, columns=columns)

## 데이터 합치기
# 산책로 + 산책로 편의시설
data = pd.merge(left = trails , right = trails_facility, how =

# 산책로 + 유저
data = pd.merge(left = data , right = users, how = "inner", l

## 데이터 합치기
# 산책로 + 산책로 편의시설
data = pd.merge(left = trails , right = trails_facility, how =

# 산책로 + 유저
data = pd.merge(left = data , right = users, how = "inner", l

# 나이 라벨인코딩
def trans_age(age_range):
    if age_range == "1~9":
        return 0
    elif age_range == "10~14":
```

```
        return 1
    elif age_range == "15~19":
        return 2
    elif age_range == "20~29":
        return 3
    elif age_range == "30~39":
        return 4
    elif age_range == "40~49":
        return 5
    elif age_range == "50~59":
        return 6
    elif age_range == "60~69":
        return 7
    elif age_range == "70~79":
        return 8
    elif age_range == "80~89":
        return 9
    elif age_range == "90~":
        return 10

data['cate_age'] = data['age_range'].apply(trans_age)

# 성별 범주화
data['gender'].replace({'male':0, "fema" : 1}, inplace=True)


# 범위 지정해주어야 하는 컬럼 : 'prefer_duration_e', 'prefer_durati
# 0 ~ 10

def minmax_scaling(min, max, v):
    return (int(v) - min) / (max - min)

data['scale_age'] = data['cate_age'].apply(lambda x : minmax_
data['scale_prefer_duration_e'] = data['prefer_duration_e'].a
data['scale_prefer_duration_s'] = data['prefer_duration_s'].a

scaling_column = ['cafe_num', 'cctv_num','convenience_num', '
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(data[scaling_column])

data[scaling_column] = scaler.transform(data[scaling_column])

train_column = ['cafe_num', 'cctv_num','convenience_num', 'po
'scale_prefer_duration_s']

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# 최대 클러스터 개수 설정
max_clusters = 5

# 각 클러스터 개수에 대한 실루엣 스코어를 저장할 리스트
silhouette_scores = []

# 각 클러스터 개수에 대해 KMeans 알고리즘을 적용하고 실루엣 스코어를 계산
for k in range(2, max_clusters):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init="au
    cluster_labels = kmeans.fit_predict(data[train_column])
    silhouette_avg = silhouette_score(data[train_column], clu
    silhouette_scores.append(silhouette_avg)

# 실루엣 스코어를 시각화하여 최적의 K 값을 찾음
best_k = silhouette_scores.index(max(silhouette_scores)) + 2


kmeans = KMeans(n_clusters=best_k, random_state=42, n_init="a

data["cluster"] = kmeans.labels_

from datetime import datetime

# 추천 db 삭제
cursor.execute("TRUNCATE TABLE recommend_trails")
```

```
conn.commit()

for i in range(len(data)):
    cursor.execute(
                    "INSERT INTO recommend_trails (created_at
                    (datetime.now(), datetime.now(), data['eu
    conn.commit()
print("Data inserted successfully!")
conn.close()

import joblib
joblib.dump(kmeans, 'kmeans_model.pkl')
joblib.dump(scaler, 'kmeans_scaler.pkl')
```

10. fastapi 실행

```
EC2 환경에서 백그라운드 실행 코드
nohup uvicorn mainmain:app --host 0.0.0.0 --port 8000 --reloa
nohup uvicorn tempmain:app --host 0.0.0.0 --port 8001 --reloa
지속적 실행
uvicorn mainmain:app --host 0.0.0.0 --port 8000
uvicorn tempmain:app --host 0.0.0.0 --port 8001
```

3. 배포 시 특이사항

- 하둡, 스파크 설치 하고 주변 시설 데이터를 HDFS에 넣어야 한다.
- npm install --legacy-peer-deps, 그림판이 오래된 라이브러리라 npm install 대신 앞 명령어 사용해야 한다.
- 모델은 한 달에 한번 배치 분석 한 후 최신화한다.

4. DB 접속 정보 등 프로젝트(ERD)에 활용되는 주요 계정 및 프로퍼티가 정의된 파일 목록

- application.properties
- application.yml
- application-dev.yml

프로젝트에서 사용하는 외부 서비스 정보

- OpenWeatherMap
- Kakao Map
- Kakao Login