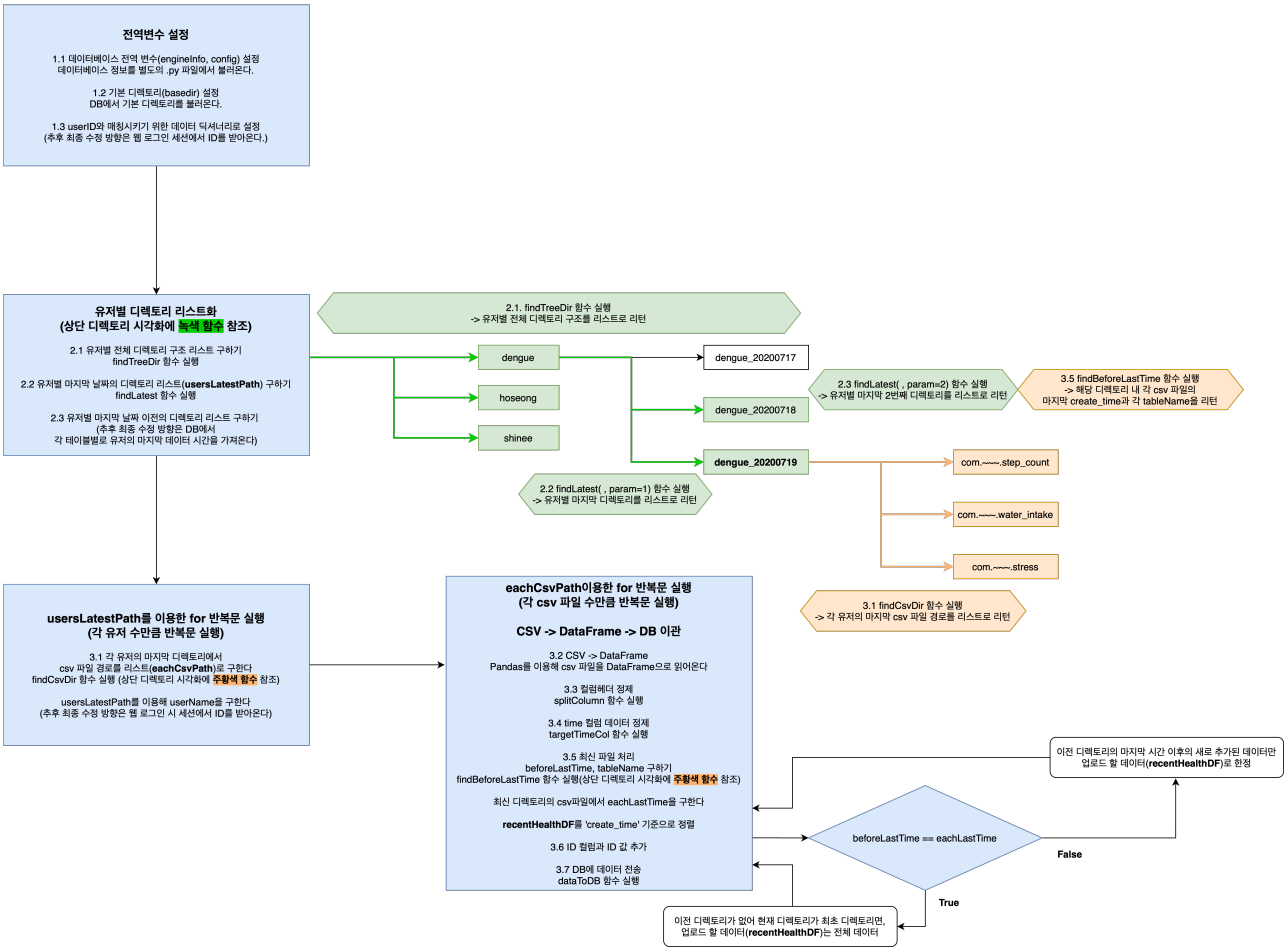
HealthCare ETL Python

워치, 밴드 데이터를 CSV 파일로 받아 DB에 적재하기

문제원형실습 2조 인수인계 문서 - 2020년 7월 21일

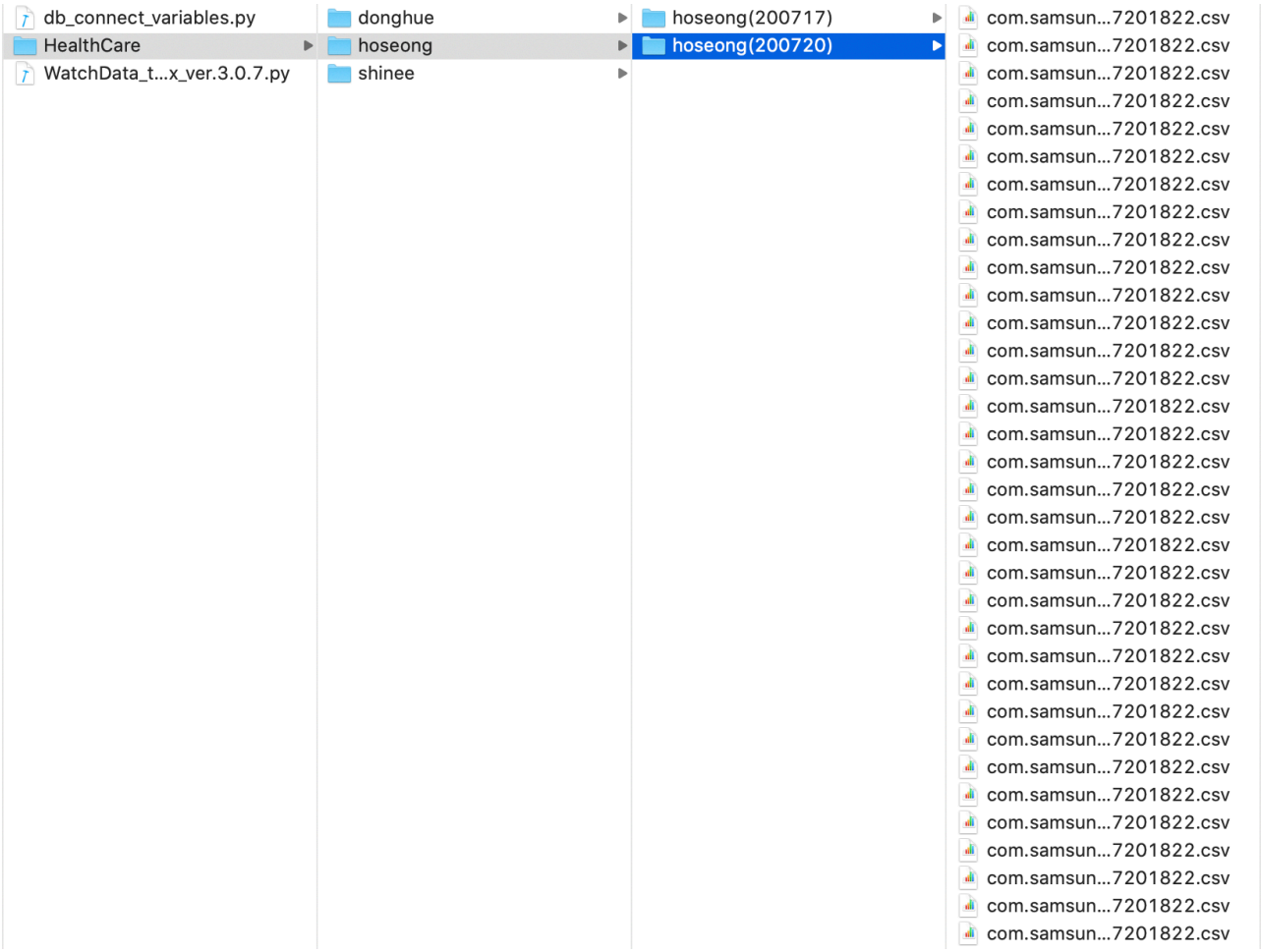
WatchData_to_DB 전개도



Introduction

삼성 스마트 워치, 밴드의 헬스 데이터를 읽어오고, 정제하고, DB에 적재하는 함수에 대해 소개합니다. 프로젝트의 디렉토리 구조에 대해 설명하고, 함수와 진행 로직에 대해 전체 흐름을 소개하도록 하겠습니다. 파이썬 코드의 로직에 대한 조직도는 위 그림을 참조하세요. (위 전개도는 확대해서 보거나 별도 첨부파일 참조)

프로젝트 디렉토리 구성 소개



프로젝트 디렉토리 구성은 다음과 같습니다.

1. Code(WatchData_to_DB_Final_Linux_ver.3.0.7.py)
   - 데이터 ETL(Extract(추출), Transform(변환), Load(적재)) 중 변환, 적재를 다룹니다.

2. DB 접속 정보를 따로 보관하는 파일(db_connect_variables.py)
   - 기밀성을 위해 서버의 정보를 코드에 저장하지 않고 별도의 파일에 보관합니다.

3. 사용자 데이터를 보관하는 디렉토리(HealthCare)
   - 하위 디렉토리로는 사람별로 나뉘고,
     각 사람의 하위 디렉토리에 사람의 이름과 날짜,
     그 하위 디렉토리에 CSV 파일을 저장

코드 전개도

코드의 전체적인 전개도는 다음과 같습니다.

1. 라이브러리 선언

...

2. 함수 정의

2-1. 디렉토리를 리스트로 추출하는 함수

...

2-2. 데이터 정제 함수

...

2-3. 최신 데이터만 가져오는 함수(이전 파일 마지막 create_time, table이름 리턴)

...

2-4. 데이터 베이스 이관 함수

...

3. 실행 로직

3-1. 전역변수 설정

...

3-2. 로직에 사용할 유저별 디렉토리 리스트화

...

3-3. CSV 파일을 읽어 데이터를 정제하고, 이전 파일과 비교해 최신 데이터만 DB에 보내는 함수

...

I. 라이브러리 선언

해당 코드를 사용하기 위해 필요한 라이브러리 모음입니다. 설치가 되어 있지 않다면 설치를 먼저 해야합니다.

```
import pandas as pd
import os
import re

from sqlalchemy import create_engine
import pymysql
pymysql.install_as_MySQLdb()
import MySQLdb
from datetime import datetime
from glob import glob
import time
import math
from db_connect_variables import connect_lnfo as db_login
```

- 디렉토리를 다루기 위해 os, glob 라이브러리를 사용합니다.
- 데이터 정제를 위해 pandas, re, datetime, 라이브러리를 사용합니다.
- DB 적재를 위해 sqlalchemy, pymysql, MySQLdb, math 라이브러리를 사용합니다.
- do_connect_variables는 기밀성을 위해 서버의 정보를 메인 코드와 분리해 사용합니다.

II. 함수 모듈화

2-1. 디렉토리를 리스트로 추출하는 함수

```
# 캐시 파일 등은 제외하고 디렉토리만 리스트로 추출하는 함수
def findDir(pwd):
    directory = []
    try:
        totalDir = os.listdir(pwd)
    except Exception as e:
        print(e)

    for eachDir in totalDir:
        fullDir = os.path.join(pwd, eachDir)
        if os.path.isdir(fullDir):
            directory.append(eachDir)

    return directory

# 입력 받은 디렉토리(pwd)와 그 하위 디렉토리 목록(subDir)을 입력 받아 경로를 merge 시켜 리스트로 반환
def mergeSubDirList(pwd, subDir):
    reDefinedDir = []

    for eachDir in subDir:
        reDefinedDir.append(os.path.join(pwd, eachDir))

    return reDefinedDir

# 대상 경로를 입력해주면 해당 경로의 하위의 하위 디렉토리까지 리스트로 반환
def findTreeDir(pwd):
    # 인풋 파라미터의 디렉토리 내의 디렉토리를 리스트로 만들어 병합
    subDir = findDir(pwd)
    subDirList = mergeSubDirList(pwd, subDir)

    # 위 디렉토리 리스트 내의 디렉토리를 다시 한 번 리스트로 만들어 병합
    treeDirList = []
    for each in subDirList:
        subDirList = findDir(each)
        treeDirList.append(mergeSubDirList(each, subDirList))

    return treeDirList
```

- 로직을 기능별로 함수화하고, 각 함수는 하나의 기능을 포함하도록 쪼갰습니다. 'findTreeDir' 함수가 해야하는 기능을 'findDir', 'mergeSubDirList'로 쪼개서 함수의 기능과 크기를 줄여 재사용성을 높이고, 디버깅이 쉽도록 하였습니다.

```
# 최신 디렉토리만 리스트로 반환
def findLatest(categories, index):
    result = []

    for eachList in categories:
        eachList.sort()
        try:
            result.append(eachList[-index])
        except Exception as e:
            result.append(eachList[0])
            print(e)

    return result
```

- 또한 함수에 파라미터를 추가해 하나의 함수로 여러 기능을 할 수 있도록 재사용성을 높였습니다.

```
# 입력 받은 디렉토리에서 CSV 파일만 전체 디렉토리 리스트로 추출
def findCsv(pwd):
    try:
        fileList = os.listdir(pwd)
        csvList = [file for file in fileList if (file.endswith(".csv")) \
                   and (file.split("\\")[-1].split(".")[0] == "com")\
                    and (re.search("(food|breath|reward|hist|recommend)", file) is None)]
    except Exception as e:
        print(e)

    return csvList

# 입력 받은 디렉토리 내에 csv 파일을 찾아 전체 디렉토리를 리스트로 반환
def findCsvDir(pwd):
    # 디렉토리 내 csv 파일을 리스트로 반환
    try:
        csvList = findCsv(pwd)
    except Exception as e:
        print(e)
    try:
        csvPathList = mergeSubDirList(pwd, csvList)
    except Exception as e:
        print(e)

    return csvPathList
```

- 전체 CSV 파일 중 필요한 파일만 가져옵니다.

```
# 컬럼명 정제 : 컬럼명 유형이 com.samsung.shealth.calories_burned.active_calorie인 경우 active_calorie만 추출
def splitColumn(columnList):
    for idx, val in enumerate(columnList):
        if re.search("com.", val) is not None:
            try:
                columnList[idx] = val.split(".")[-1]
            except Exception as e:
                print(e)
    return columnList
```

- 아래와 같이 com.~~ 형태로 들어오는 컬럼명 다른 테이블과 동일하게 정제합니다.

com.samsung.health.heart_rate.end_time	com.samsung.health.heart_rate.heart_rate
1593756678884	82.0
1594256641000	76.0
1594271483000	75.0
1594278191000	82.0
1594521618000	84.0
1594009747108	78.0
1594267199000	97.0

```python
# time 데이터 형식 일치 (2020. 07. 14 형식이나 timestamp 형식을 2020-07-14 형식으로 통일)
def dateTodatetime(date):
    date = str(date)
    result = None
    if re.search("\d{4}. \d{2}. \d{2}", date) is not None: # 시간 형식이 2020. 07. 08. ~인 경우에 대해서만
        splitDate = [each.strip() for each in date.split('.')]
        splitDateLen = len(splitDate)

        yearMonthDay = "-".join(splitDate[:splitDateLen - 1])
        hourMinSec = [each for each in splitDate[-1].split(":")]

        if "오후" in hourMinSec[0]:
            hourMinSec[0] = str(int(hourMinSec[0].split(" ")[-1]) + 12)
        elif ("오전" or "오후 12") in hourMinSec[0]:
            hourMinSec[0] = str(int(hourMinSec[0].split(" ")[-1]))

        hourMinSec = ":".join(hourMinSec)
        result = yearMonthDay + " " + hourMinSec
    elif re.search("\d{4}-\d{2}-\d{2}", str(date)) is not None:
        #splitDate = [each.strip() for each in date.split('.')]
        #print(splitDate)
        #splitDateLen = len(splitDate)
        #if splitDateLen == 2:
        result = str(date).split('.')[0]
        #else:
            #print(splitDate)
            #result = date
    elif len(str(date)) <= 8 and re.search("UTC", date) is None: # time_offset / longest_idle_time(-1 일때 ㅊ
        if not math.isnan(float(date)):
            result = convertMillis(int(float(date)))
    elif len(str(date)) == 13: # 13 digits unixtime / day_time / set_time / start_date / date / update_time
        result = datetime.fromtimestamp(int(date)/1000).strftime("%Y-%m-%d %H:%M:%S")
    else: # UTC+0900. time_offset
        result = date

    return result
```

- 시간 데이터가 3가지 종류로 들어와 하나의 형태로 정제하는 함수입니다.

타겟	정제할 타입 1	정제할 타입 2
create_time	create_time	com.samsung.shealth.calor
2017-11-10 03:02:52.442	2026. 06. 08. 오후 11:56:54	1569058792662
2017-11-10 03:02:52.443	2020. 07. 07. 오후 11:57:14	1569058794300
2017-11-10 03:02:52.444	2020. 07. 08. 오후 1:00:06	1569058794613
2017-11-10 03:02:52.445	2026. 05. 13. 오후 11:22:18	1569058794576

```
## 기존 데이터베이스 테이블에 없는 컬럼값을 입력했을 때 발생되는 에러를 반영하여 데이터베이스 테이블에 컬럼을 추가하는 함수
def alterColumnToDB(error, tableName, config):
    try:
        conn = MySQLdb.connect(**config)
        cur = conn.cursor()
        query = ""
        if re.search("1054", error) is not None:
            column = "".join(re.findall("\w", error.split(' ')[4]))
            query = "alter table {0} add column {1} text".format(tableName, column)
            cur.execute(query)
        elif re.search("1366", error) is not None:
            column = error.split(" ")[8].split("`")[-2]
            query = "alter table {0} modify column {1} text".format(tableName, column)
            cur.execute(query)
        elif re.search("1265", error) is not None:
            column = error.split(" ")[6].split("'")[1]
            query = "alter table {0} modify column {1} mediumtext".format(tableName, column)
            cur.execute(query)
        conn.close()
    except Exception as e:
        print(e)

## 데이터베이스 테이블에 데이터를 추가하는 함수
def dataToDB(engineInfo, recentHealthDF, tableName):
    error = ""
    try:
        engine = create_engine(engineInfo)
#         engine = create_engine("mysql+pymysql://root:1234@13.125.210.149:3306/health")
        recentHealthDF.to_sql(name="{}".format(tableName), con=engine, if_exists="append", index=False)
#         recentHealthDF.to_sql(name="{}".format(tableName), con=engine, if_exists="replace", index=False)
    except Exception as e:
        print(e)
        error = str(e)
        print("failed to transfer healthData to DB")

        if re.search("(1054|1366|1265)", error) is not None:
            alterColumnToDB(error, tableName, config)
            dataToDB(engineInfo, recentHealthDF, tableName)
```

- DB에 적재하는 함수입니다. 테이블에 컬럼이 없어서 에러가 발생하는 경우 에러코드를 받아 컬럼을 생성하고, 다시 적재하는 함수를 호출하도록 기능별로 모듈화 하였습니다.

III. 실행 로직

실행 로직 역시 변수와 엔진을 설정(1단계)하고, 디렉토리를 리스트화 하는 작업(2단계), 마지막으로 CSV파일을 읽어 정제, DB에 적재하는 작업(3단계)로 이루어집니다.

1단계) 기밀성을 위해 서버 정보를 별도 파일에서 불러옵니다.

```
# 1.1 데이터베이스 관련 전역변수. create_engine / MySQLdb.connect 시 사용
db_name = db_login['db_name']
db_api_name = db_login['db_api_name']
db_user = db_login['db_user']
db_password = db_login['db_password']
db_host = db_login['db_host']
db_port = db_login['db_port']
db_schema = db_login['db_schema']

engineInfo = "{0}+{1}://{2}:{3}@{4}:{5}/{6}".format(db_name, db_api_name, db_user, db_password, db_host, db_port, db_schema)
#print(engineInfo)
config = {"user":db_user, "password":db_password, "host":db_host, "port":db_port, "db":db_schema}
#print(config)

# 1.2 기본 디렉토리 설정
engine = create_engine(engineInfo)
query = "select * from meta_variables_dir"
variables = pd.read_sql(query, engine)

basedir = variables.basedir.values[0]
# print(basedir)

# 1.3 userID와 매칭시키기 위한 ID 데이터 딕셔너리로 설정 (최종적으로는 웹 로그인 시 세션에서 ID를 받아와야 함)
userIDDic = {"shinee" : "shinee", "donghue" : "dong2", "hoseong" : "hocastle"}
```

2단계) 로직에 사용할 유저별 디렉토리를 리스트로 만듭니다.

```
# 2.1 유저별 전체 디렉토리 구조를 리스트로 추출
usersDirList = findTreeDir(basedir)

# 2.2 유저별 마지막 날짜의 디렉토리 리스트 (usersDirList의 유저별 하위 디렉토리의 마지막 디렉토리를 리스트로 추출)
usersLatestPath = findLatest(usersDirList, 1)

# 2.3 유저별 마지막 날짜 이전의 디렉토리 리스트 (각 유저의 최신 데이터를 구별하기 위해 이전 날짜의 하위 디렉토리를 찾는다)
usersBeforePath = findLatest(usersDirList, 2)
```

3단계) ETL에서 T와 L을 실행하는 코드입니다. 대부분의 코드는 간단한 작업을 제외하고 전부 함수를 통해 실행하도록 되어있습니다. 따라서 코드의 가독성이 높을 뿐 아니라 문제가 발생한 단계의 함수만 디버깅을 함으로써 유지보수에 신경을 썼습니다.

```
for val in usersLatestPath:
    # 3.1 각 사람의 마지막 디렉토리에서 csv 파일 경로를 리스트로 추출
    eachCsvPath = findCsvDir(val)

    # DB에 넣을 때 ID컬럼에 값을 추가하기 위한 로직 (최종적으로는 웹 로그인 시 세션에서 ID를 받아와야 함)
    userName = val.split("/")[2]
    for eachCsv in eachCsvPath:
        print(eachCsv)   # 디버깅용
        # 3.2 CSV 파일 읽어오기
        eachHealthDF = pd.read_csv(eachCsv, skiprows=1, index_col=False)

        # 3.3 컬럼헤더 정제
        columnsList = eachHealthDF.columns.tolist()   # 컬럼헤더를 리스트로 바꾼다
        refinedColumnList = splitColumn(columnsList)  # com.~~을 정제하는 함수 로직 실행
        eachHealthDF.columns = refinedColumnList      # 정제된 리스트를 다시 컬럼헤더로

        # 3.4 2007. 02. 03 형식 또는 timestamp를 2007-02-03 형식으로 정제
        columns = eachHealthDF.columns.values
        targetColumns = targetTimeCol(columns)

        for each in targetColumns:
            eachHealthDF[each] = eachHealthDF[each].apply(dateTodatetime)

        # 3.5 최신 파일 처리
        beforeLastTime, tableName = findBeforeLastTime(eachCsv)

        eachLastTime = eachHealthDF.create_time.max()

        if eachLastTime == beforeLastTime:
            recentHealthDF = eachHealthDF
        else:
            recentHealthDF = eachHealthDF.loc[eachHealthDF.create_time > beforeLastTime]
#         recentHealthDF = eachHealthDF if eachLastTime == beforeLastTime else eachHealthDF.loc[eachLastTim

        sortKey = ['create_time']
        recentHealthDF.sort_values(by=sortKey, inplace=True)

        # 3.6 ID 컬럼과 ID 값 추가
        recentHealthDF['id'] = userIDDic[userName]

        # 3.7 DB에 데이터 전송
        dataToDB(engineInfo, recentHealthDF, tableName)

        time.sleep(0.5)
```

바깥 for문은 처음 소개된 디렉토리에서 각 사람별로 반복되는 로직이고, 안쪽 for문은 각 사람의 최신 날짜 디렉토리(마지막 디렉토리) 내의 모든 CSV 파일을 반복하는 로직입니다.