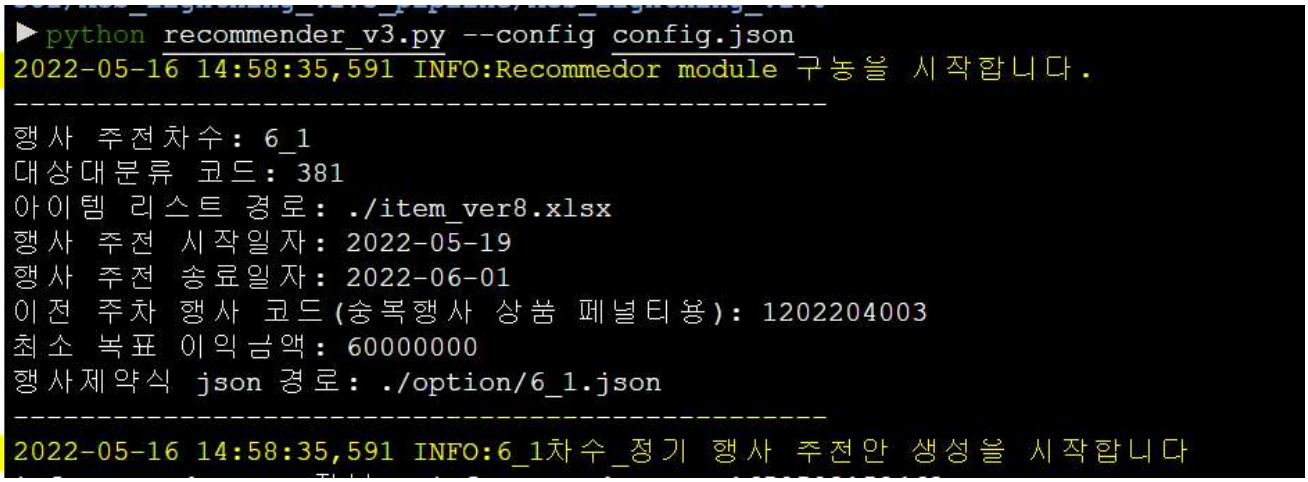
Recommender pipline을 위한 logging 작업 가이드 및 Format 논의 사항

이인환(알렉스파트너) - AI/ML담당Data Science Chapter 고자영(벨라파트너) - AI/ML담당Data Science Chapter

i) 간단한 logging 모듈 설명



- **Print 로 출력하는 것 보다는 logging 패키지를 활용하면, 각 모듈별, 단계별로 log 시간과 함께 출력이 가능합니다.**

1) log는 위험도 따라 level 별로 설정 가능합니다.

- **터미널 실행시, 주요 프로세스등은 log로 남겨 저장 & 출력하는 작업이 필요한데, 행사 추천 시작, 및 종료 정보저장 부분등은 INFO 단계로 출력. Fuction 오류시에는 ERROR 단계등으로 출력하는게 좋습니다.**

Level	Value	When to use
DEBUG	10	(주로 문제 해결을 할 때 필요한) 자세한 정보.
INFO	20	작업이 정상적으로 작동하고 있다는 확인 메시지.
WARNING	30	예상하지 못한 일이 발생하거나, 발생 가능한 문제점을 명시. (e.g. 'disk space low') 작업은 정상적으로 진행.
ERROR	40	프로그램이 함수를 실행하지 못 할 정도의 심각한 문제.
CRITICAL	50	프로그램이 동작할 수 없을 정도의 심각한 문제.

- config 파일이 없는 경우 if__name == '__main__' 부분이 오류날 것이며, 이는 프로그램 자체가 동작할 수 없는 문제기 때문에 CRTICAL 단계로 표시할 수 있습니다.

2) logging은 처음에 logger를 정의 해야하는데, 모듈별로 만들수 있으며, 각 모듈별로 출력 log level를 달리 가져갈 수 있습니다. (즉, option을 달리 가져갈 수 있음)

다음, 최상위(root) 로거에 이전 단계에서 생성한 핸들러 두 개를 연결해주도록 하겠습니다. 최상위 로거는 `WARNING` 심각도 이상의 로그만 남기도록 설정하였습니다.

```
root_logger = logging.getLogger()
root_logger.addHandler(console_handler)
root_logger.addHandler(file_handler)
root_logger.setLevel(logging.WARNING)
```

마지막으로, 특정 모듈을 위한 로거에 대한 설정을 해보겠습니다. `parent` 모듈은 `INFO` 레벨 이상의 로그만 남기도록, `parent.child` 모듈은 `DEBUG` 레벨 이상의 로그만 남기도록 하였습니다.

```
parent_logger = logging.getLogger("parent")
parent_logger.setLevel(logging.INFO)

child_logger = logging.getLogger("parent.child")
child_logger.setLevel(logging.DEBUG)
```

이런 식으로 모듈 별로 로깅 레벨을 다른 게 설정해줌으로써, 특정 모듈에 대해서는 다른 모듈에 비해서 좀 더 자세한 로그를 남길 수 있습니다. 여기서 `parent` 로거와 `parent.child` 로거에 별도로 핸들러 설정을 하지 않아도 되는 이유는 자식 로거의 로그 메세지는 부모 로거로 전파(propogate)되기 때문입니다.

즉, `parent.child` 로거의 로그 메세지는 `parent` 로거로 전파되고, 이는 다시 최종적으로 최상위 로거로 전파됩니다. 따라서 `parent` 로거와 `parent.child` 로거가 남기는 로그는 취상위 로거에 연결되어 있는 두 개의 핸들러에 의해서 처리될 것 입니다.

- 위 예시에서는 Root - Parent -Child 구조로 설명하였으며, 최상위를 root, childe를 parent에 부속되는 단위로 설명

3) log 출력시에는 Formatter에 담긴 옵션을 통해 진행합니다.

```
import logging

logging.basicConfig(
    format='%(asctime)s %(levelname)s:%(message)s',
    level=logging.DEBUG,
    datefmt='%m/%d/%Y %I:%M:%S %p',
)

logging.debug("Sample debug message")
```

***출력결과**

```
04/23/2022 04:33:13 PM DEBUG:Sample debug message
```

어트리뷰트 이름	포맷	설명
args	직접 포맷할 필요는 없습니다.	message 를 생성하기 위해 msg 에 병합되는 인자의 튜플. 또는 (인자가 하나뿐이고 딕셔너리일 때) 병합을 위해 값이 사용되는 딕셔너리.
asctime	%(asctime)s	사람이 읽을 수 있는, LogRecord 가 생성된 시간. 기본적으로 '2003-07-08 16:49:45,896' 형식입니다 (쉼표 뒤의 숫자는 밀리 초 부분입니다).
created	%(created)f	LogRecord 가 생성된 시간 (time.time() 이 반환하는 시간).
exc_info	직접 포맷할 필요는 없습니다.	예외 튜플 (sys.exc_info 에서 제공) 또는, 예외가 발생하지 않았다면, None.
filename	%(filename)s	pathname 의 파일명 부분.
funcName	%(funcName)s	로깅 호출을 포함하는 함수의 이름.
levelname	%(levelname)s	메시지의 텍스트 로깅 수준 ('DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL').
levelno	%(levelno)s	메시지의 숫자 로깅 수준 (DEBUG, INFO, WARNING, ERROR, CRITICAL).
lineno	%(lineno)d	로깅 호출이 일어난 소스 행 번호 (사용 가능한 경우).
message	%(message)s	로그 된 메시지. msg % args 로 계산됩니다. Formatter.format() 이 호출 될 때 설정됩니다.
module	%(module)s	모듈 (filename 의 이름 부분).
msecs	%(msecs)d	LogRecord 가 생성된 시간의 밀리 초 부분.
msg	직접 포맷할 필요는 없습니다.	원래 로깅 호출에서 전달된 포맷 문자열. args 와 병합하여 message 를 만듭니다. 또는 임의의 객체 (임의의 객체를 메시지로 사용하기 를 보세요).
name	%(name)s	로깅 호출에 사용된 로거의 이름.
pathname	%(pathname)s	로깅 호출이 일어난 소스 파일의 전체 경로명 (사용 가능한 경우).
process	%(process)d	프로세스 ID (사용 가능한 경우).
processName	%(processName)s	프로세스 이름 (사용 가능한 경우).
relativeCreated	%(relativeCreated)d	logging 모듈이 로드된 시간을 기준으로 LogRecord가 생성된 시간 (밀리 초).
stack_info	직접 포맷할 필요는 없습니다.	현재 스레드의 스택 바닥에서 이 레코드를 생성한 로깅 호출의 스택 프레임까지의 스택 프레임 정보 (사용 가능한 경우).
thread	%(thread)d	스레드 ID (사용 가능한 경우).
threadName	%(threadName)s	스레드 이름 (사용 가능한 경우).

4) logging parameter 파일 역시 json 파일로 관리합니다.

```
import logging
import logging.config

config = {
    "version": 1,
    "formatters": {
        "simple": {"format": "[%(name)s] %(message)s"},
        "complex": {
            "format": "%(asctime)s %(levelname)s [%(name)s] [%(filename)s:%(lineno)d] - %
        },
    },
    "handlers": {
        "console": {
            "class": "logging.StreamHandler",
            "formatter": "simple",
            "level": "DEBUG",
        },
        "file": {
            "class": "logging.FileHandler",
            "filename": "error.log",
            "formatter": "complex",
            "level": "ERROR",
        },
    },
    "root": {"handlers": ["console", "file"], "level": "WARNING"},
    "loggers": {"parent": {"level": "INFO"}, "parent.child": {"level": "DEBUG"},},
}

logging.config.dictConfig(config)
```

- logging.streamhandler는 터미널 창에 출력 되는 옵션, logging.FileHandler는 log 파일 저장에 사용되는 옵션으로 해당 예시를 보면 출력창에는 단순히 출력하고, 파일 저장시는 구체적으로 저장해라 라는 뜻입니다.

(streamhandler: simple formatter, filehandler: complex formatter)

- root logger에는 warning 이상, 단순하게 터미널에 출력해라. parent에는 info 레벨이상, child에는 debug lervel이상으로 출력해라 라는 뜻입니다.

ii) 그래서 logging을 Recommender에 적용한다면? (논의 사항)

현재까지 코드:http://10.147.13.249:8022/notebooks/dev-corca/wonwuk.lee/event_recommend/6_1_event/381/MGS_lightning_v1.3_pipline/MGS_lightning_v1.0/recommender_v3.py

1.현재 pipline 관련 recommeder.py는 크게 3부분으로 나뉘어져있습니다.

1) if __name == '__main__' 부분

-python script를 터미널에서 실행하기 위한 부분으로 써 config 파일을 load하고, config 파일 parameter에 없는 부분 (추천안 output 파일 저장경로 , log json 파일 경로등(없애도 될듯함)) 추가 하여 해당정보가 def main으로 들어감

2) def main: Recommender 모듈이 실제로 구동되는 부분

-input confing.json 파일에 입력 되지 않은 parameter에 default 값을 채워주며, recommend class가 실행되고 결과 저장 및 관련 정보가 파일에 저장되는 Function입니다.

3) class Recommender: Recommender 모듈. 관련된 fucnction 및 쿼리를 비롯하여 행사 추천안을 생성하고 결과에 맞도록 Format으로 만들어주는 부분

- Class 안에 init 부분에는 def main에서 받아지는 config 파일이 input으로 들어가서 관련인자를 받아서 생성됨

2.앞서 i-2)의 설명에 빗대어 보면 최상위 레벨인 root는 ii-1-1)인 if__name 부분 parent는 ii-1-2) main 부분 child는 ii-1-3) Recommender로 빗대어 볼 수 있습니다.(물론 정의방식에 따라 다를 수 있습니다)

-Root 부분은 config 파일의 존재 여부가 아니면 오류 날것이 없으므로 Critical 정보의 구분만 넣기

-Parent 부분은 핵심 부분으로 추천안 시작, 끝, log 정보 저장 등등의 중간 스텝마다 info, 그리고 예상가능 한 문제 작성후 warning, function 자체의 오류가 날 수 있으므로 error 단계 정도까지 작성

-child 부분은 때에따라 debug, 주요 필요 출력결과(행사 제약식, 매가 변경 시작일자) 등은 info, 그리고 waring이나 error 까지 적용

3. 적용방식은 i-4)와 같이 json 파일을 만들어서 load하는 방식으로 진행하고, 추후 구동이 안될시 slack noti 까지 하면 좋을 것 같은데, 이미 Airflow상 구현이 되어있는 것 같아 굳이 필요할지는 고민이 필요 것 같습니다.

→ 출력방식은: log 시간, 오류단계, 오류시 구문? 추천안 시작, 종류 같은?? 내용정도 생각하고 있습니다.

진행경과 (22/05/17)

***ii)-1-2) Recommender Class를 제외한 1-1), 1-3) 부분 logging 작업완료**

```
▶ python recommender_v3.py --config config.json --log_file log_config.json
2022-05-17 16:37:23:recommender_v3:INFO:최적의 행사주전안을 제안하는 모듈입니다
2022-05-17 16:37:23:recommender_v3:INFO:구동시작 시간 2022-05-17 07:37:23
--------------------------------------------------
행사 주전차수: 6_1
대상대분류 코드: 381
아이템 리스트 경로: ./item_ver8.xlsx
행사 주전 시작일자: 2022-05-19
행사 주전 송료일자: 2022-06-01
이전 주차 행사 코드(숭복행사 상품 페널티용): 1202204003
최소 복표 이익금액: 60000000
행사제약식 json 경로: ./option/6_1.json
--------------------------------------------------
2022-05-17 16:37:23:recommender_v3:INFO:6_1차수_정기 행사 주전안 생성을 시작합니다
inference_dataset 정보:  inference_dataset_1650593158463.csv
**************************************
최근6개월 가격의 시작일자: 2021-11-17
최근6개월 가격의 끝일자: 2022-05-16
price_df load complete!
price, prdt dataset 불러오기 완료
**************************************
이전 행사리스트 수 28
all_점포 주전결과
```

*logging 변환방식

- log_config.json 파일에 콘솔과, 파일에 저장하는 logging 단위를 나눠서 작성
- 콘솔: INFO level 부터 출력, File recommender_debug.log 파일과 recommender_error.log 파일로 나눠서 Debug는 Debug level 부터 Error는 Error level 부터 log를 남기도록 지정
- log printer format 방식은 구동시간, 모듈명, level, 출력 메세지까지 하도록 진행

```
{
"version": 1,
"disable_existing_loggers": false,
"formatters": {
"basic": {
"format": "%(asctime)s:%(module)s:%(levelname)s:%(message)s",
"datefmt": "%Y-%m-%d %H:%M:%S"
}
},
"handlers": {
"console": {
"class": "logging.StreamHandler",
"level": "INFO",
"formatter": "basic"
},
"file_debug": {
"class": "logging.FileHandler",
"level": "DEBUG",
"formatter": "basic",
"filename": "recommender_debug.log"
},
"file_error": {
"class": "logging.FileHandler",
"level": "ERROR",
"formatter": "basic",
"filename": "recommender_error.log"
}
},
"loggers": {
"__main__": {
"level": "DEBUG",
"handlers": ["console", "file_debug", "file_error"],
"propagate": true
}
}
}
```

*Error log 처리방식

- try, except 문 사용 단, 오류시 어떤 오류가 나는지 Tracking을 할 수 있도록, error log 시 어떤 부분이 문제인지 요약+ 해당코드의 Detail문제까지 같이 출력
- Recommender class도 rebuilding 작업이 끝나면 중간중간 해당 log를 추가 해야함

***관련코드**

```
try:
    recommend_df=Recommend.make_recommend_df()
    logger.info("{}_정기 행사 추천안 생성을 완료하였습니다".format(kwargs['number_of_events']))

except Exception as e:
    print("-------------------------------------------------")
    logger.error("{}차수_정기 행사 추천안 생성을 실패하였습니다".format(kwargs['number_of_events']))
    trace_back = traceback.format_exc()
    message = str(e)+ "₩n" + str(trace_back)
    logger.error('[FAIL] %s', message)
```

***출력결과**

```
-------------------------------------------------
6_1차수_정기 행사 추천안 생성을 실패하였습니다
2022-05-17 16:20:37:recommender_v3:ERROR:[FAIL] name 'df_predicte' is not defined
Traceback (most recent call last):
  File "/workspace/dev-corca/wonwuk.lee/event_recommend/6_1_event/381/MGS_lightning_v1.3_pipline/MGS_lightning_v1.0/recommen
der_v3.py", line 111, in main
    recommend_df=Recommend.make_recommend_df()
  File "/workspace/dev-corca/wonwuk.lee/event_recommend/6_1_event/381/MGS_lightning_v1.3_pipline/MGS_lightning_v1.0/recommen
der_v3.py", line 1264, in make_recommend_df
    result_df,df_all,exclude_prdt_list=self.recommend()
  File "/workspace/dev-corca/wonwuk.lee/event_recommend/6_1_event/381/MGS_lightning_v1.3_pipline/MGS_lightning_v1.0/recommen
der_v3.py", line 808, in recommend
    print(df_predicte)
NameError: name 'df_predicte' is not defined

-------------------------------------------------
```

*log 저장 파일 예시(error log 파일)