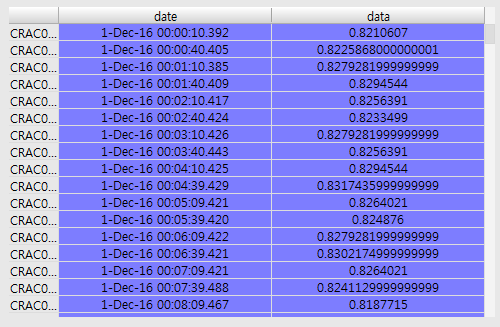
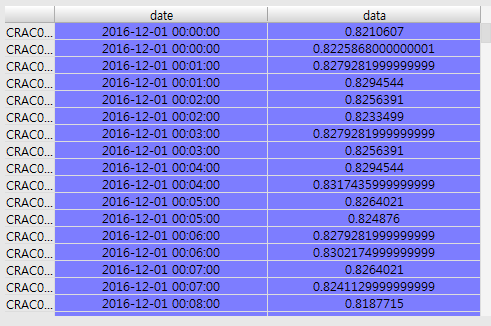
**SK**

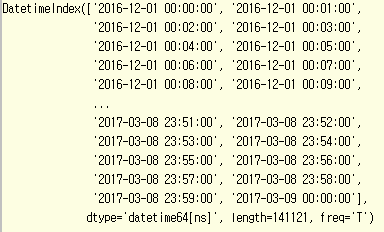
1. Python script 실행
   1. Argument 추가 및 사용자 옵션(Flag) 설정
      1. Raw 데이터 저장 디렉터리
      2. 데이터 처리 단위(1 sec, 1 min, 1 hour …)
      3. 처리 구간 설정(start date ~ end date)
2. Raw 데이터 파일(.csv) 불러오기
   1. Lab 데이터와 비 Lab 데이터 구분하여 파일 리스트(csv 파일 위치) 저장
   2. 분할되어 있는 파일(ex : ARPC014A.Y\_2017\_04\_1, ARPC014A.Y\_2017\_05\_1) 병합하여 파일 생성 후 비 Lab 데이터 파일리스트에 추가  
        
      사용자가 미리 파일 이름 설정 가능.
   3. 각 파일 이름(Tag 이름) 별도 저장.
3. csv 파일 읽기
   1. csv 파일 형식은 (date, value) 로 되어있음.
   2. Lab 데이터와 비 Lab 데이터를 구분하여 csv파일을 읽음.
      1. 읽는 동시에 행으로 붙임. = rbind,   
         한 열에 대해서 일괄 처리하기 위함.(행 또는 열이 복잡해질수록 연산량이 늘어날 가능성 증가하기 때문)

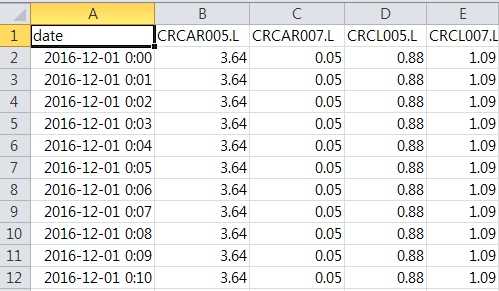
|  |  |  |
| --- | --- | --- |
|  | date | value |
| CRTI117B.Y | 20-Apr-17 12:00:29 | 358.4029 |
|  | 20-Apr-17 12:00:59 | 366.5465 |
|  | … |  |
| ARTI212.Y | 20-Apr-17 12:00:29 | 719.6918 |
|  | … | … |

* + 1. Dataframe  
       

1. 데이터 전처리
   1. String to Date factor 변환
      1. 사용자가 지정한 옵션에 따라 원하는 단위로 변환 가능(default : 분)
      2. Date 형식은 20-Apr-17 12:00:29
      3. datetime의 strptime 함수를 사용 시 쉽게 변환가능 <- 하지만 느림
         1. 빠른 변환을 위해 indexing 접근
         2. **lambda** x: dt.combine(date(int(**"20"** + x[7:9]), abbr\_to\_num[x[3:6]], int(x[:2])),  
             time(int(x[10:12]), int(x[13:15]), 1))  
            ex : x[7:9] = 17(년). X[3:6] = Apr(월), x[:2] = 20(일), x[10:12] = 12(시) …  
            마지막 초는 1초로 직접 설정 <- 분단위로 변경하기 위해서 같은 분 내에서는 모든 같은 초를 가짐.(후 과정에서 첫 번째 값만 유지)
         3. abbr\_to\_num 은 약어를 월(숫자)로 매핑하는 함수
      4. 결과로 (2016, 12, 01, 12, 00, 00) 형식의 factor 생성  
         출력은 2016-12-01 12:00:00 AM 형식  
         
   2. 분(초, 시) 단위 데이터로 변환
      1. 사용자가 지정한 옵션에 따라 원하는 단위로 변경 가능(default : 분)

|  |
| --- |
| DatetimeIndex |
| 2016-12-01 12:00:01 |
| 2016-12-01 12:01:01 |
| … |
| 2017-03-08 12:02:01 |
| 2017-03-08 12:03:01 |
| 2017-03-08 12:04:01 |

* + 1. 사용자가 지정한 start date와 end date factor 변경 후  
       해당 기간만큼 분 단위에 date time index 생성
    2. DatetimeIndex로 index 재설정
       1. 위의 date factor 과정에서 모든 시간을 1초로 바꾸었음.  
          중복 시간 60개 존재.(milli 초 단위에서는 더 많은 중복 가능)
       2. Drop\_duplicates에서 첫 번째 값만을 사용하게 하여 중복 제거  
          (사전에 1초씩 값을 빼 01초가 첫 번째에 놓이도록 설정)
       3. Date를 기준으로 DatetimeIndex에 병합.  
          각 데이터에서 비어있는 시간들은 Nan으로 채워짐  
          
       4. Nan을 처리하기 위해 가장 최근의 이전 값으로 Nan을 채움  
          사전에 Digital State라는 값들이 csv파일에 포함되어 있는 경우가 있는데 사전에 Digital State를 Nan으로 변경해 놓음.(Digital State로 되어있으면 산술 연산 불가하기 때문)  
          이전 데이터가 없을 경우에 대비해 가장 최근의 후의 값으로 Nan을 다시 채움.
  1. 일렬로 병합된 Dataframe을 transpose 및 Lab 데이터와 비 Lab 데이터 병합
     1. 사전에 masking 해놓은 tag name을 기준으로 행으로 병합한다.(cbind)
     2. 위의 과정은 Lab 데이터와 비 Lab 데이터에 동일하게 분리되어 적용  
        행으로 Lab 데이터와 비 Lab 데이터를 병합
     3. Index 설정이 되어있지 않아 index를 date로 설정
  2. Csv 파일로 최종 RTDB 저장



**DBDiscover**

1. Python script 실행
2. Raw 데이터 저장 디렉터리 진입
   1. 해당 디렉터리의 모든 파일 경로 저장
3. 반복문
   1. csv 파일 한 개 단위로 읽음.   
      읽으면서 parser를 이용해 date factor로 변환 (긴 실행시간)
   2. date를 기준으로 정렬
   3. Resample 함수를 이용하여 1초 단위로 reindex
      1. ffill()을 이용해 가장 최근의 data로 reindex 할 때 생성된 Nan 요소들을 이전 최근 데이터로 채움.
      2. 중복 값이 존재할 시(같은 초 단위 존재) groupby로 중복을 묶고 그 중 첫 번째 값만 유지  
         선 작업 후 resample
   4. 분 단위의 대표값 ‘01초’ 설정
      1. 데이터의 index를 탐색하면서 01초인 행에만 masking
      2. Masking 한 행들만 추출하여 최종 분 단위 데이터 생성
   5. 해당 Dataframe csv 파일로 저장
   6. 모든 파일에 대해 반복

DBDiscover 프로세스 : 개별 파일에 대한 처리 및 처리된 개별 파일 저장

SK 프로세스 : 개별 파일 처리 후 병합한 RTDB 파일 저장

속도 차이 영역 : 문자열의 date를 factor 변경  
DBDiscover의 parser의 느린 처리가 요인

|  |  |  |
| --- | --- | --- |
|  | SK (단위 : s) | DBDiscover (단위 : s) |
| 표준 실행 시간(129 파일) | 242 | 1892 |
| 분할파일 없는 경우 | 110 | 1892 |
| 분할 파일 처리 | 분할파일 병합 추가작업 | 추가 작업 x |
| 10만 row 이상 파일  (ARPC014A.Y) | 5.9 | 154.8 |