빅데이터 전처리

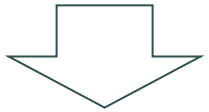
2회차 – 데이터 구성 전처리

1. **필요한 데이터만 추출하기**
2. 분석 단위를 손실 없이 변경하기
3. 여러 테이블 합치기
4. 학습용과 검증용 데이터 나누기
5. 불균형한 데이터를 보정용 데이터로 생성하기
6. 집계 데이터를 표 형식으로 바꾸기

지정된 데이터 열 추출

➢ 예약 테이블에서 reserve_id, hotel_id, customer_id, reserve_datetime을 선택하여 추출한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000



reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27

지정된 데이터 열 추출

- 방법1 : Reserve_tb의 배열에 추출할 열 이름을 문자열로 지정한다.

```
reserve_tb[['reserve_id', 'hotel_id', 'customer_id',
            'reserve_datetime', 'checkin_date', 'checkin_time',
            'checkout_date']]
```

여러 개의 컬럼을 선택할때

- 방법2 : Loc함수의 2차원 배열의 2차원 항목에 추출할 열 이름을 배열로 지정하여 열을 추출한다.

```
reserve_tb.loc[:, ['reserve_id', 'hotel_id', 'customer_id',
                   'reserve_datetime', 'checkin_date',
                   'checkin_time', 'checkout_date']]
```

- 방법3 : Drop 함수로 불필요한 열을 제거한다. axis=1은 열을 의미하고, inplace는 reserve_tb를 갱신한다.

```
reserve_tb.drop(['people_num', 'total_price'], axis=1, inplace=True)
```

NOTE> 방법1/2는 데이터가 업데이트 되지 않음.

조건을 부여되어 있는 데이터 행 추출

- 방법1 : DataFrame 배열에 지정한 조건의 결과값인 True/False를 가지는 행의 배열을 지정하여 추출한다.

```
reserve_tb[(reserve_tb['checkout_date'] >= '2016-10-13') &
           (reserve_tb['checkout_date'] <= '2016-10-14')]
```

- 방법2 : loc 함수의 2차원 배열의 1차원 항목에 지정한 조건의 결과값인 True/False를 지정하여 추출한다.

```
reserve_tb.loc[(reserve_tb['checkout_date'] >= '2016-10-13') &
               (reserve_tb['checkout_date'] <= '2016-10-14'), :]
```

- 방법3 : Query함수와 조건식을 이용하여 행을 추출한다.

```
reserve_tb.query('"2016-10-13" <= checkout_date <= "2016-10-14"')
```

중복성을 고려하지 않은 랜덤 샘플링

- 호텔 예약 레코드를 활용하여 약 50%의 랜덤 샘플링을 수행한다.
- Reserve_tb에서 50% 샘플링하는 코드

```
reserve_tb.sample(frac=0.5)
```

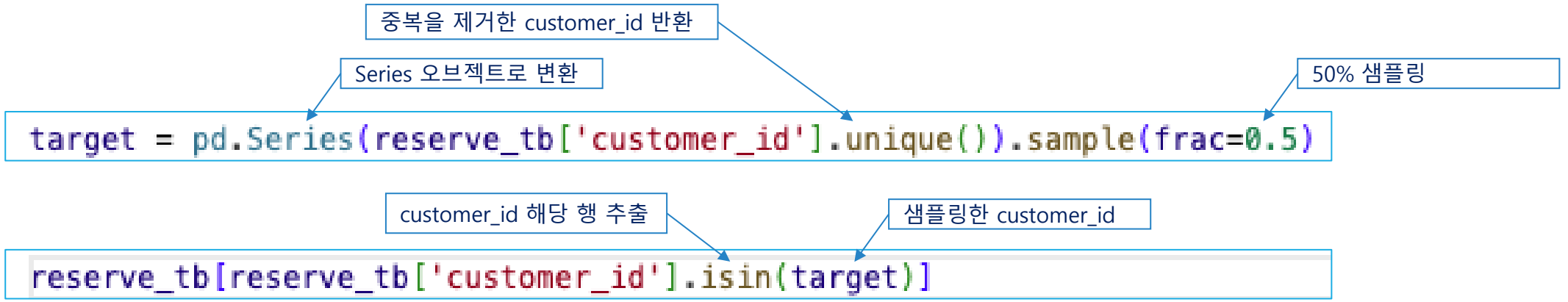
awesomebook-master/preprocess/002_selection/03/python_awesome.py

고객 ID에 기반한 샘플링 – 중복성 제거

- 어느 한 쪽으로 편향된 샘플링을 하게되면 이후 분석에서 잘못된 결과를 도출할 수 있다.
- 공평하게 샘플링하려면 분석 대상의 단위와 샘플링 단위를 서로 맞춰야 한다.
  - 연간 예약 횟수별 고객 수의 비율 계산시 샘플링 전후 분석 결과의 차이가 발생함
  - 이유 → 분석 대상 단위가 고객 한 명인데, 샘플링 단위는 예약한 건수가 됨.
  - 분석 대상의 단위와 샘플링 단위가 서로 달라서 발생한 현상.
- 해결 방법 : 예약 테이블의 고객 ID를 대상으로 랜덤 샘플링을 실행하고 샘플링한 고객 ID의 예약 레코드만 추출

집약 ID에 기반한 샘플링

- unique 함수로 중복값을 제거하여 pandas.Series를 얻고 sample로 50%를 추출한다.
- isin 함수로 매개변수로 전달된 리스트 값 중 일치하는 열의 값만을 추출한다.



NOTE> print(target)으로 target의 내용을 확인!

awesomebook-master/preprocess/002_selection/04/python_awesome.py

1. 필요한 데이터만 추출하기
2. **분석 단위를 손실 없이 변경하기**
3. 여러 테이블 합치기
4. 학습용과 검증용 데이터 나누기
5. 불균형한 데이터를 보정용 데이터로 생성하기
6. 집계 데이터를 표 형식으로 바꾸기

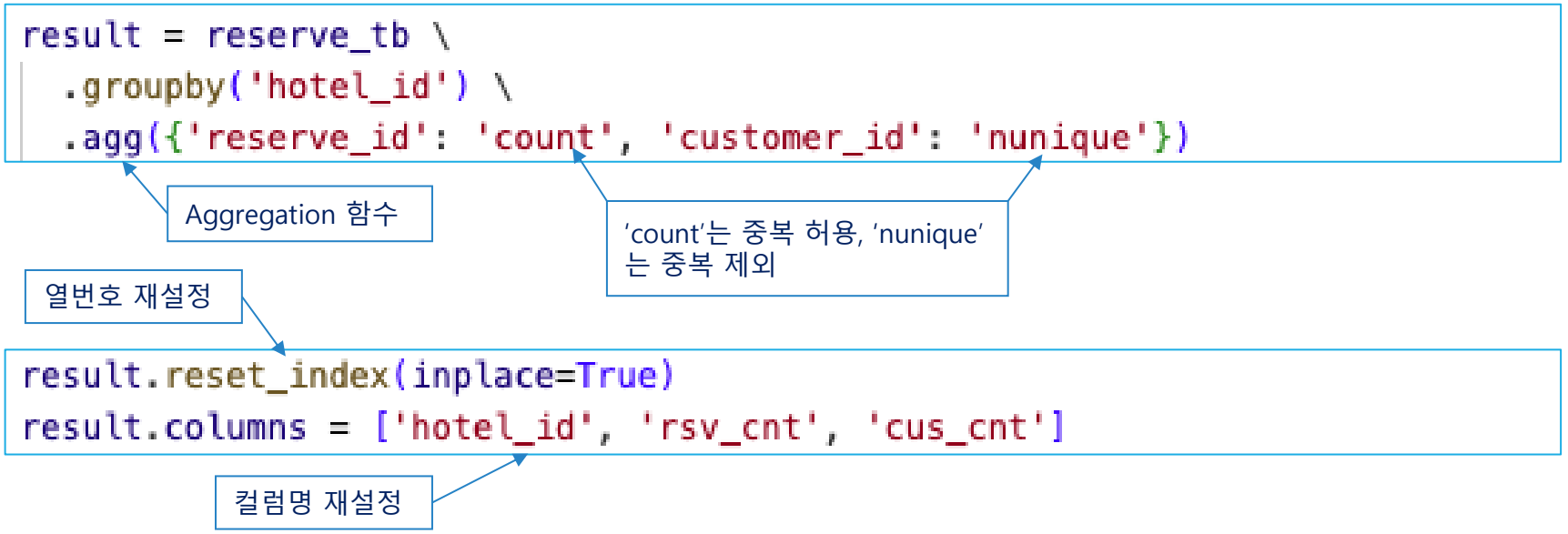
손실없이 데이터 분석 단위 변경 : Aggregation

- Aggregation → 데이터의 가치 손실 없이 분석 단위를 변경할 수 있는 방법
- 데이터의 가치를 되도록 손실없이 압축하여 데이터의 단위(행의 의미)를 변환할 수 있는 처리

  예> 시험 과목별로 점수의 평균값을 계산하면 시험 과목의 난이도를 쉽게 파악할 수 있음.
- 의미 : 데이터의 전체적인 경향을 파악하는데 도움이 되고 정보 손실이 적은 변환 처리가 가능
- 방법 : GROUP_BY로 집약할 단위를 지정하여 count, sum 함수를 이용하는 방법

agg() 함수를 활용한 개수 산출

- agg 함수의 매개변수에 dictionary오브젝트를 지정하여 Aggregation 처리를 한꺼번에 지정할 수 있다.
- dictionary 오브젝트는 key에 열의 이름, value에 집약함수 이름을 지정한다.



sum()을 이용한 매출 합계 산출

- 분석대상의 값이 숫자일때 데이터의 합을 계산하는 경우에 사용한다.

```
result = reserve_tb \
  .groupby(['hotel_id', 'people_num'])['total_price'] \
  .sum().reset_index()
```

2단계로 Grouping

매출 합계 산출

```
result.rename(columns={'total_price': 'price_sum'}, inplace=True)
```

컬럼명 변경

테이블에 변경 내용 반영

max, min, mean, median, percentile(백분위수) 산출

➢ 분석대상의 값이 숫자일때 최대값, 최소값, 평균, 중간값, 백분위수를 계산하는 경우에 사용한다.

```
result = reserve_tb \
  .groupby('hotel_id') \
  .agg({'total_price': ['max', 'min', 'mean', 'median',
                        lambda x: np.percentile(x, q=20)]}) \
  .reset_index()
result.columns = ['hotel_id', 'price_max', 'price_min', 'price_mean',
                  'price_median', 'price_20per']
```

하위 20% 가격중 제일 높은 점수

Variance(분산)과 Standard deviation(표준편차) 산출

- 분산값과 표준편차값은 데이터의 분포 정도를 나타낸다.

```
result = reserve_tb \
  .groupby('hotel_id') \
  .agg({'total_price': ['var', 'std']}).reset_index()
result.columns = ['hotel_id', 'price_var', 'price_std']
```

분산

표준편차

```
result.fillna(0, inplace=True)
```

분산이나 표준편차가 na로 나올경우 0으로 대체

NOTE> 출력값이 지수형태로 나올 경우 아래 코드를 추가하면 됨.

pd.options.display.float_format = '{:,.0f}'.format

mode(최빈값) 계산

- 최빈값은 가장 많이 나타나는 수치를 의미한다.
- Reserve_tb에서 최빈값 계산

```
reserve_tb['total_price'].round(-3).mode()
```

가장 자주 등장하는 값 추출

천 단위로 반올림. 비슷한 가격대 묶기 위함

rank()를 이용한 순위 계산

➢ 그룹별로 순서를 정렬하고 순위를 매겨서 새로 추가된 열에 기록한다.

```
reserve_tb['reserve_datetime'] = pd.to_datetime(
  reserve_tb['reserve_datetime'], format='%Y-%m-%d %H:%M:%S'
)
```

문자열에서 Timestamp 형으로 변환

```
reserve_tb['log_no'] = reserve_tb \
  .groupby('customer_id')['reserve_datetime'] \
  .rank(ascending=True, method='first')
```

''reserve_datetime 컬럼 선택

오름차순으로 순위를 매김

'first' → 동일한 시간이 있을때 DataFrame에 먼저 등장한 순서대로 순위를 매김

rank()를 이용한 순위 계산

- 예약 테이블을 이용하여 호텔별 예약 건수에 따른 순위를 메겨본다.

```
rsv_cnt_tb = reserve_tb.groupby('hotel_id').size().reset_index()
rsv_cnt_tb.columns = ['hotel_id', 'rsv_cnt']
```

groupby().size() → hotel_id별로 묶은 후 행 개수, 즉 예약회수를 센다.

```
rsv_cnt_tb['rsv_cnt_rank'] = rsv_cnt_tb['rsv_cnt'] \
  .rank(ascending=False, method='min')
```

내림차순으로 지정

동점이 나오면 그 다음 순위는 건너뛴다.

```
rsv_cnt_tb.drop('rsv_cnt', axis=1, inplace=True)
```

불필요한 'rsv_cnt'는 제거

1. 필요한 데이터만 추출하기
2. 분석 단위를 손실 없이 변경하기
3. **여러 테이블 합치기**
4. 학습용과 검증용 데이터 나누기
5. 불균형한 데이터를 보정용 데이터로 생성하기
6. 집계 데이터를 표 형식으로 바꾸기

레코드 테이블과 마스터 테이블의 결합

- 예약 테이블과 호텔 테이블을 결합해 숙박 인원수가 한 명인 비지니스 예약 레코드를 추출해본다.
- Step1. 예약 테이블에서 people_num이 1인 레코드를 추출

reserve_id	hotel_id	customer_	reserve_datetime	checkin_da	checkin_ti	checkout_da	people_nu	total_price
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27	3	68400
r10	h_240	c_2	2016.6.25 9:12	2016.7.14	11:00:00	2016.7.17	4	320400
r11	h_183	c_2	2016.11.19 12:49	2016.12.8	11:00:00	2016.12.11	1	29700
r12	h_268	c_2	2017.5.24 10:06	2017.6.20	9:00:00	2017.6.21	4	81600
r13	h_223	c_2	2017.10.19 3:03	2017.10.21	9:30:00	2017.10.23	1	137000
r14	h_133	c_2	2018.2.18 5:12	2018.3.12	10:00:00	2018.3.15	2	75600
r15	h_92	c_2	2018.4.19 11:25	2018.5.4	12:30:00	2018.5.5	2	68800

people_num이 1인 레코드 추출

reserve_id	hotel_id	customer_	reserve_datetime	checkin_da	checkin_ti	checkout_da	people_nu	total_price
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000
r11	h_183	c_2	2016.11.19 12:49	2016.12.8	11:00:00	2016.12.11	1	29700
r13	h_223	c_2	2017.10.19 3:03	2017.10.21	9:30:00	2017.10.23	1	137000
r18	h_132	c_3	2016.10.22 2:18	2016.11.12	12:00:00	2016.11.13	1	20400
r23	h_61	c_3	2017.12.16 23:31	2018.1.9	9:00:00	2018.1.12	1	224400
r25	h_277	c_4	2016.3.28 7:17	2016.4.7	10:30:00	2016.4.10	1	39300
r26	h_132	c_4	2016.5.11 17:48	2016.6.5	11:30:00	2016.6.6	1	20400
r32	h_287	c_4	2017.11.2 19:00	2017.11.5	10:00:00	2017.11.7	1	29000
r34	h_273	c_5	2016.11.25 15:44	2016.12.19	12:00:00	2016.12.22	1	134700
r35	h_90	c_5	2017.3.30 13:38	2017.4.5	11:30:00	2017.4.7	1	16000
r42	h_63	c_7	2016.11.11 12:42	2016.11.26	9:00:00	2016.11.29	1	44700
r45	h_104	c_7	2017.9.28 11:44	2017.10.5	9:00:00	2017.10.7	1	84400

레코드 테이블과 마스터 테이블의 결합

- 예약 테이블과 호텔 테이블을 결합해 숙박 인원수가 한 명인 비지니스 예약 레코드를 추출해본다.
- Step2. 호텔 테이블에서 is_business가 TRUE인 레코드를 추출

hotel_id	base_price	big_area_n	small_area	hotel_latitu	hotel_long	is_business
h_1	26100	D	D-2	43.0645686	141.511397	TRUE
h_2	26400	A	A-1	35.7153197	139.939446	TRUE
h_3	41300	E	E-4	35.2815717	136.988565	FALSE
h_4	5200	C	C-3	38.4312931	140.795615	FALSE
h_5	13500	G	G-3	33.5972915	130.533872	TRUE
h_6	49500	A	A-3	35.9127637	139.731281	TRUE
h_7	18900	C	C-2	38.3287016	140.894969	FALSE
h_8	12400	B	B-2	35.5433183	139.798737	FALSE
h_9	31400	C	C-1	38.2326736	140.795693	FALSE
h_10	5600	A	A-3	35.9138742	139.931003	FALSE

is_business가 TRUE인 레코드 추출

hotel_id	base_price	big_area_n	small_area	hotel_latitu	hotel_long	is_business
h_1	26100	D	D-2	43.0645686	141.511397	TRUE
h_2	26400	A	A-1	35.7153197	139.939446	TRUE
h_5	13500	G	G-3	33.5972915	130.533872	TRUE
h_6	49500	A	A-3	35.9127637	139.731281	TRUE
h_18	27800	G	G-1	33.4916764	130.536058	TRUE
h_22	10700	D	D-3	43.1648754	141.408229	TRUE
h_24	4600	E	E-1	35.1868207	136.883244	TRUE
h_25	10100	A	A-1	35.8121611	139.739566	TRUE
h_26	49900	C	C-2	38.3326553	140.897438	TRUE
h_29	9300	C	C-1	38.2366101	140.891594	TRUE
h_30	7700	A	A-1	35.714166	139.83767	TRUE

레코드 테이블과 마스터 테이블의 결합

- 예약 테이블과 호텔 테이블을 결합해 숙박 인원수가 한 명인 비지니스 예약 레코드를 추출해본다.
- Step3. hotel_id를 키로 결합

reserve_id	hotel_id	customer_	reserve_datetime	checkin_da	checkin_ti	checkout_da	people_nu	total_price
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000
r11	h_183	c_2	2016.11.19 12:49	2016.12.8	11:00:00	2016.12.11	1	29700
r13	h_223	c_2	2017.10.19 3:03	2017.10.21	9:30:00	2017.10.23	1	137000
r18	h_132	c_3	2016.10.22 2:18	2016.11.12	12:00:00	2016.11.13	1	20400
r23	h_61	c_3	2017.12.16 23:31	2018.1.9	9:00:00	2018.1.12	1	224400
r25	h_277	c_4	2016.3.28 7:17	2016.4.7	10:30:00	2016.4.10	1	39300
r26	h_132	c_4	2016.5.11 17:48	2016.6.5	11:30:00	2016.6.6	1	20400
r32	h_287	c_4	2017.11.2 19:00	2017.11.5	10:00:00	2017.11.7	1	29000
r34	h_273	c_5	2016.11.25 15:44	2016.12.19	12:00:00	2016.12.22	1	134700
r35	h_90	c_5	2017.3.30 13:38	2017.4.5	11:30:00	2017.4.7	1	16000
r42	h_63	c_7	2016.11.11 12:42	2016.11.26	9:00:00	2016.11.29	1	44700
r45	h_104	c_7	2017.9.28 11:44	2017.10.5	9:00:00	2017.10.7	1	84400

hotel_id를 키로 결합한다.

hotel_id	base_price	big_area_n	small_area	hotel_latitu	hotel_long	is_business
h_1	26100	D	D-2	43.0645686	141.511397	TRUE
h_2	26400	A	A-1	35.7153197	139.939446	TRUE
h_5	13500	G	G-3	33.5972915	130.533872	TRUE
h_6	49500	A	A-3	35.9127637	139.731281	TRUE
h_18	27800	G	G-1	33.4916764	130.536058	TRUE
h_22	10700	D	D-3	43.1648754	141.408229	TRUE
h_24	4600	E	E-1	35.1868207	136.883244	TRUE
h_25	10100	A	A-1	35.8121611	139.739566	TRUE
h_26	49900	C	C-2	38.3326553	140.897438	TRUE
h_29	9300	C	C-1	38.2366101	140.891594	TRUE
h_30	7700	A	A-1	35.714166	139.83767	TRUE

레코드 테이블과 마스터 테이블의 결합

➢ 예약 테이블과 호텔 테이블을 결합해 숙박 인원수가 한 명인 비지니스 예약 레코드를 추출해본다.

```
pd.merge(reserve_tb.query('people_num == 1'),
         hotel_tb.query('is_business'),
         on='hotel_id', how='inner')
```

숙박객수 1명인 데이터 추출

영업중인 호텔 데이터 추출

hotel_id를 key 로 두개 테이블을 결합함

공통된 hotel_id가 있는 행만 결합

awesomebook-master/preprocess/004_join/01/python_2_awesome.py

마스터 테이블을 조건에 따라 변경하기

➢ 결합을 위한 새로운 열을 생성한다.

hotel_id	base_price	big_area_nam	small_area_na	hotel_latitude	hotel_longitu	is_business
h_1	26100	D	D-2	43.0645686	141.511397	TRUE
h_2	26400	A	A-1	35.7153197	139.939446	TRUE
h_3	41300	E	E-4	35.2815717	136.988565	FALSE
h_4	5200	C	C-3	38.4312931	140.795615	FALSE
h_5	13500	G	G-3	33.5972915	130.533872	TRUE
h_6	49500	A	A-3	35.9127637	139.731281	TRUE
h_7	18900	C	C-2	38.3287016	140.894969	FALSE

small_area_name 기준 호텔 수 계산

big_area_nam	small_area_na	hotel_cnt
A	A-1	33
A	A-3	11
B	B-1	23
B	B-2	17
B	B-3	18
C	C-1	29
C	C-2	33
C	C-3	22
D	D-1	5
D	D-2	5

마스터 테이블을 조건에 따라 변경하기

- 결합을 위한 새로운 열을 생성한다.

hotel_cnt가 20이상이면 small_area_name
20미만이면 big_area_name 지정

small_area_na	join_area_id
A-1	A-1
A-3	A
B-1	B-1
B-2	B
B-3	B
C-1	C-1
C-2	C-2
C-3	C-3
D-1	D
D-2	D

small_area_name을 키로 사용하여
hotel_id/Join_area_id 지정

hotel_id	join_area_id
h_1	A-1
h_2	D
h_3	C-1
h_4	C-2
h_5	A
h_6	C-3
h_7	D
h_8	A
h_9	B
h_10	D

마스터 테이블을 조건에 따라 변경하기

➢ 공통열(join_area_id)를 제작한다.

hotel_id	base_price	big_area_nam	small_area_na	hotel_latitude	hotel_longitu	is_business
h_1	26100	D	D-2	43.0645686	141.511397	TRUE
h_2	26400	A	A-1	35.7153197	139.939446	TRUE
h_3	41300	E	E-4	35.2815717	136.988565	FALSE
h_4	5200	C	C-3	38.4312931	140.795615	FALSE
h_5	13500	G	G-3	33.5972915	130.533872	TRUE
h_6	49500	A	A-3	35.9127637	139.731281	TRUE
h_7	18900	C	C-2	38.3287016	140.894969	FALSE
h_8	12400	B	B-2	35.5433183	139.798737	FALSE
h_9	31400	C	C-1	38.2326736	140.795693	FALSE

join_area_id를
small_area_name에서 추출

join_area_id	rec_hotel_id
D-2	h_1
A-1	h_2
E-4	h_3
C-3	h_4
G-3	h_5
A-3	h_6
C-2	h_7
B-2	h_8
C-1	h_9
A-3	h_10

두 테이블 결합

join_area_id를 big_area_name에서 추출

join_area_id	rec_hotel_id
D	h_1
A	h_2
E	h_3
C	h_4
G	h_5
A	h_6
C	h_7
B	h_8
C	h_9
A	h_10

=

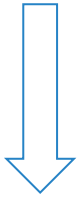
join_area_id	rec_hotel_id
D	h_1
A	h_2
E	h_3
C	h_4
G	h_5
D-2	h_1
A-1	h_2
E-4	h_3
C-3	h_4
G-3	h_5

마스터 테이블을 조건에 따라 변경하기

- 공통열(join_area_id)를 제작한다.

hotel_id	join_area_id
h_1	A-1
h_2	D
h_3	C-1
h_4	C-2
h_5	A
h_6	C-3
h_7	D
h_8	A
h_9	B
h_10	D

join_area_id	rec_hotel_id
D	h_1
A	h_2
E	h_3
C	h_4
G	h_5
D-2	h_1
A-1	h_2
E-4	h_3
C-3	h_4
G-3	h_5



join_area_id를 키로
hotel_id와 rec_hotel_id를 결합한다.

hotel_id	rec_hotel_id
h_1	h_14
h_2	h_22
h_3	h_27
h_4	h_40
h_5	h_45
h_6	h_77
h_7	h_79

마스터 테이블을 조건에 따라 변경하기

```
import gc

small_area_mst = hotel_tb \
  .groupby(['big_area_name', 'small_area_name'], as_index=False) \
  .size().reset_index()
print(small_area_mst)

small_area_mst.columns = ['index', 'big_area_name', 'small_area_name', 'hotel_cnt']

small_area_mst['join_area_id'] = \
  np.where(small_area_mst['hotel_cnt'] - 1 >= 20,
           small_area_mst['small_area_name'],
           small_area_mst['big_area_name'])

small_area_mst.drop(['hotel_cnt', 'big_area_name'], axis=1, inplace=True)

base_hotel_mst = pd.merge(hotel_tb, small_area_mst, on='small_area_name') \
                   .loc[:, ['hotel_id', 'join_area_id']]

del small_area_mst
gc.collect()
```

Garbage collection 라이브러리

small_area_name 기준 호텔 수 계산

group기준 컬럼을 인덱스로 사용하지 안ㅎ고 일반 컬럼으로 둠

20건이상 → small_area_name
20건이하 → big_area_name

hotel_id와 join_area_id 로 구성된 마스터 테이블 제작

불필요한 메모리 정리

마스터 테이블을 조건에 따라 변경하기

```
recommend_hotel_mst = pd.concat([
  hotel_tb[['small_area_name', 'hotel_id']] \
    .rename(columns={'small_area_name': 'join_area_id'}, inplace=False),

  hotel_tb[['big_area_name', 'hotel_id']] \
    .rename(columns={'big_area_name': 'join_area_id'}, inplace=False)
])

recommend_hotel_mst.rename(columns={'hotel_id': 'rec_hotel_id'}, inplace=True)

pd.merge(base_hotel_mst, recommend_hotel_mst, on='join_area_id') \
  .loc[:, ['hotel_id', 'rec_hotel_id']] \
  .query('hotel_id != rec_hotel_id')
```

추천 후보 마스터 테이블 제작

추천 후보 테이블 이름 변경

join_area_id를 key로 테이블 merge

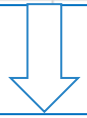
Base 호텔과 피추천 호텔 열만 남기기

Base호텔과 피추천 호텔이 동일한 경우 제외

과거 데이터를 사용하기 위한 준비

➢ customer_id 별로 정렬 후 특정 열 데이터를 n칸 아래로 이동한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27	3	68400
r10	h_240	c_2	2016.6.25 9:12	2016.7.14	11:00:00	2016.7.17	4	320400



데이터 특정 열을 두 칸 아래로 이동시킴

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price	before_price
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200	NA
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600	NA
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600	97200
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400	20600
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100	33600
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000	194400
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500	68100
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000	36000
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27	3	68400	103500
r10	h_240	c_2	2016.6.25 9:12	2016.7.14	11:00:00	2016.7.17	4	320400	6000

과거 데이터를 사용하기 위한 준비

➢ 예약 테이블의 모든 행에 고객이 이전에 예약했던 두 번의 금액 정보를 첨부해본다.

```
result = reserve_tb \
  .groupby('customer_id') \
  .apply(lambda group:
         group.sort_values(by='reserve_datetime', axis=0, inplace=False))
```

예약시간을 기준으로 정렬

행 기준 정렬

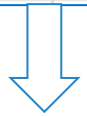
```
result['before_price'] = \
    pd.Series(result['total_price'].shift(periods=2))
```

데이터 행을 아래로 2행 옮김

과거 n건의 합계 구하기

- 예약 테이블의 모든 행에 자신의 행에서 같은 고객의 이전 예약 세건의 예약 금액 정보를 추출하여 합계를 구한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27	3	68400
r10	h_240	c_2	2016.6.25 9:12	2016.7.14	11:00:00	2016.7.17	4	320400



본 건과 과거 두 건의 합계 계산

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price	price_sum
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200	NA
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600	NA
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600	151400
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400	248600
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100	296100
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000	298500
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500	207600
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000	145500
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27	3	68400	NA
r10	h_240	c_2	2016.6.25 9:12	2016.7.14	11:00:00	2016.7.17	4	320400	NA

과거 n건의 합계 구하기

- 예약 테이블의 모든 행에 자신의 행에서 같은 고객의 이전 예약 세 건의 예약 금액 정보를 추출하여 합계를 구한다.

```
result = reserve_tb.groupby('customer_id') \
  .apply(lambda x: x.sort_values(by='reserve_datetime', ascending=True)) \
  .reset_index(drop=True)
result['price_sum'] = pd.Series(
    result.loc[:, ["customer_id", "total_price"]]
    .groupby('customer_id')
    .rolling(center=False, window=3, min_periods=3).sum()
    .reset_index(drop=True)
    .loc[:, 'total_price']
)
```

인덱스 초기화

예약 시간 기준으로 정렬

3칸짜리 이동 윈도우내에 숫자 합계 구해서 total_price에 저장

현재행과 그 앞 2개행 기준 롤링

최소3건이 모여야 합을 계산함

과거 n건의 평균값 구하기

➢ 예약 테이블의 모든 행에 한 건 이전의 데이터에서 세 건 이전까지의 평균예약금액을 구한다.

```
result = reserve_tb.groupby('customer_id') \
  .apply(lambda x: x.sort_values(by='reserve_datetime', ascending=True)) \
  .reset_index(drop=True)

result['price_avg'] = pd.Series(
  result
    .groupby('customer_id')
    ['total_price'].rolling(center=False, window=3, min_periods=1).mean()
    .reset_index(drop=True)
)

result['price_avg'] = \
  result.groupby('customer_id')['price_avg'].shift(periods=1)
```

3칸짜리 이동 윈도우내에 평균을 구해서 price_avg에 저장

직전 예약들의 평균만 반영하기 위해 한 칸씩 아래로 이동

awesomebook-master/preprocess/004_join/03_c/python_not_awesome.py

과거 n일의 합계값 구하기

- 예약 테이블의 모든 데이터에 자신을 포함하지 않으면서 같은 고객의 지난 90일간의 합계 예약 금액 정보를 첨부한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27	3	68400
r10	h_240	c_2	2016.6.25 9:12	2016.7.14	11:00:00	2016.7.17	4	320400

과거 90일의 total_price 합계 계산

reserve_id	total_price_90d
r3	20600
r6	68100
r7	36000
r15	75600
r16	68800

과거 n일의 합계값 구하기

➢ 예약 테이블의 모든 데이터에 자신을 포함하지 않으면서 같은 고객의 지난 90일간의 합계 예약 금액 정보를 첨부한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27	3	68400
r10	h_240	c_2	2016.6.25 9:12	2016.7.14	11:00:00	2016.7.17	4	320400

과거 90일의 total_price 합계 계산

reserve_id	total_price_90d
r3	20600
r6	68100
r7	36000
r15	75600
r16	68800

reserve_id 키로 결합

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_dat	people_num	total_price	total_price_90d
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200	0
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	20600	0
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	9:00:00	2016.10.22	2	33600	20600
r4	h_214	c_1	2017.3.8 3:20	2017.3.29	11:00:00	2017.3.30	4	194400	0
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	10:30:00	2017.9.23	3	68100	0
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000	68100
r7	h_256	c_1	2017.12.29 10:38	2018.1.25	10:30:00	2018.1.28	1	103500	36000
r8	h_241	c_1	2018.5.26 8:42	2018.6.8	10:00:00	2018.6.9	1	6000	0
r9	h_217	c_2	2016.3.5 13:31	2016.3.25	9:30:00	2016.3.27	3	68400	0
r10	h_240	c_2	2016.6.25 9:12	2016.7.14	11:00:00	2016.7.17	4	320400	0

과거 n일의 합계값 구하기

➢ 예약 테이블의 모든 데이터에 자신을 포함하지 않으면서 같은 고객의 지난 90일간의 합계 예약 금액 정보를 첨부한다.

```
import pandas.tseries.offsets as offsets
import operator

reserve_tb['reserve_datetime'] = \
  pd.to_datetime(reserve_tb['reserve_datetime'], format='%Y-%m-%d %H:%M:%S')

sum_table = pd.merge(
  reserve_tb[['reserve_id', 'customer_id', 'reserve_datetime']],
  reserve_tb[['customer_id', 'reserve_datetime', 'total_price']]
            .rename(columns={'reserve_datetime': 'reserve_datetime_before'}),
  on='customer_id')

sum_table = sum_table[operator.and_(
  sum_table['reserve_datetime'] > sum_table['reserve_datetime_before'],
  sum_table['reserve_datetime'] + offsets.Day(-90) <= sum_table['reserve_datetime_before']
)].groupby('reserve_id')['total_price'].sum().reset_index()

sum_table.columns = ['reserve_id', 'total_price_sum']

pd.merge(reserve_tb, sum_table, on='reserve_id', how='left').fillna(0)
```

일시형으로 변환

customer_id를 기반으로 선택된 필드를 Self-join 수행

오른쪽 reserve_datetime을 reserve_datetime_before 로 변경

90일 이전 데이터 필터링

합계 산출

산출된 데이터를 reserve_tb에 병합

Cross Join

- 결합하는 양쪽 테이블을 모두 조합하여 생성하는 결합이다.
- 주로 집계나 학습 데이터를 만들기 위한 전처리에서 사용된다.

customer_id	age	sex	home_latitude	home_longitude	year_month
c_1	41	man	35.092193	136.512347	201701
c_2	38	man	35.325076	139.410551	201702
	49	woman	35.120543	136.511179	201703

동일 customer_id의 과거 90일간 예약 레코드 생성

customer_id	year_month
c_1	201701
c_1	201702
c_1	201703
c_2	201701
c_2	201702
c_2	201703

Cross Join

➢ 결합하는 양쪽 테이블을 모두 조합하여 생성하는 결합이다.

```
import datetime
from dateutil.relativedelta import relativedelta

month_mst = pd.DataFrame({
  'year_month':
    [(datetime.date(2017, 1, 1) + relativedelta(months=x)).strftime("%Y%m")
     for x in range(0, 3)]
})

customer_tb['join_key'] = 0
month_mst['join_key'] = 0

customer_mst = pd.merge(
  customer_tb[['customer_id', 'join_key']], month_mst, on='join_key'
)
```

2017sus 1월부터 3월까지의 year_month를 가지는 DataFrame생성

cross join용 결합키 준비

customer_id와 month_mst로 Cross join

1. 필요한 데이터만 추출하기
2. 분석 단위를 손실 없이 변경하기
3. 여러 테이블 합치기
4. **학습용과 검증용 데이터 나누기**
5. 불균형한 데이터를 보정용 데이터로 생성하기
6. 집계 데이터를 표 형식으로 바꾸기

개요

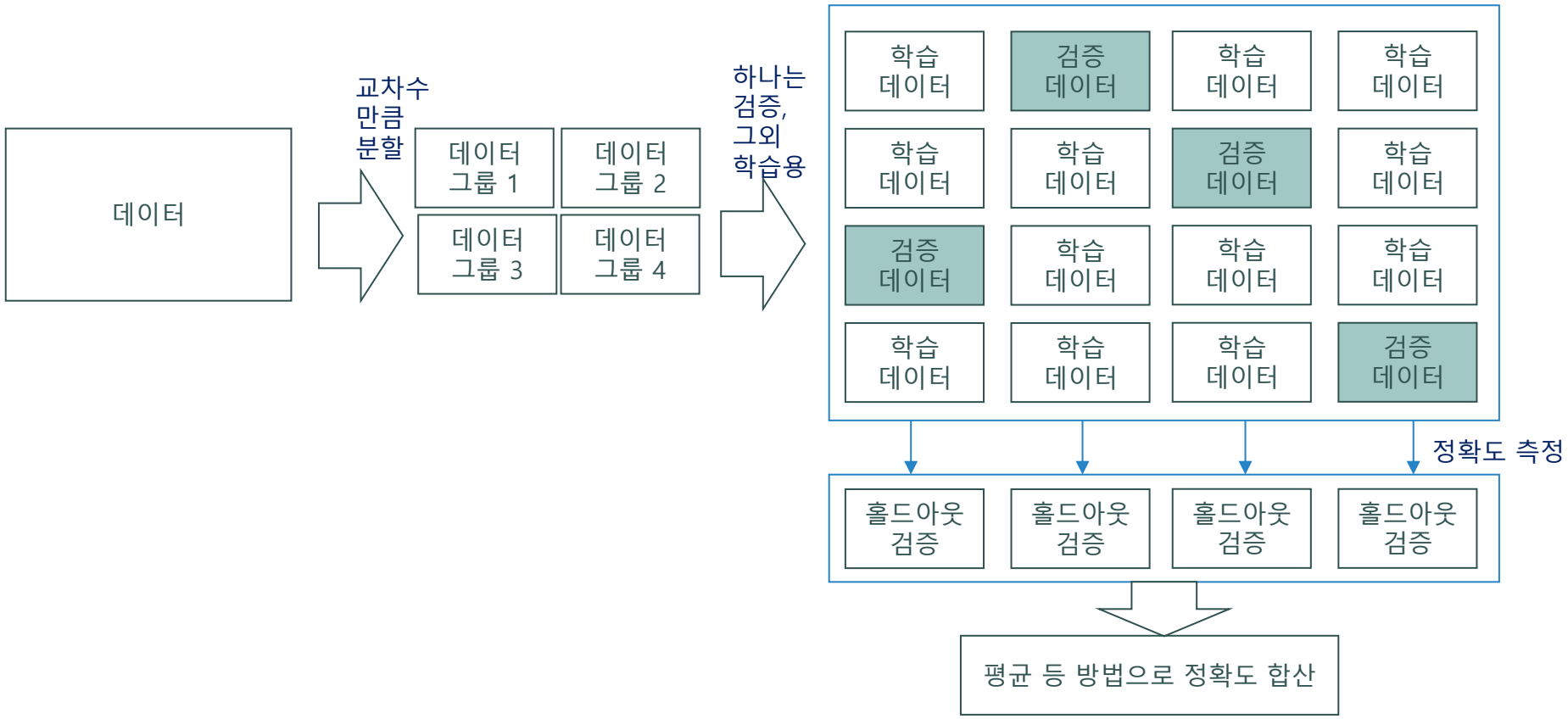
- 데이터 분할은 예측 모델을 평가할 때 필요한 전처리 이다.
- 주로 학습 데이터와 검증 데이터의 분할에 사용된다.
- 학습 데이터와 검증 데이터는 같은 전처리를 적용하며, 되도록 같은 데이터로 묶어서 다루고,

  예측모델에 입력하기 직전 분할하는 것이 적절하다.

- 적용데이터의 경우 답을 알 수 없는 상태에서 사용하는 데이터로 데이터를 얻는 방법과 흐름, 타이밍도 다르다.
- 따라서 적용 데이터의 경우 데이터를 분할할 필요가 없다.

머신러닝 검증을 위한 데이터 레코드 분할

➢ 교차 검증은 데이터를 검증용 데이터와 학습용 데이터로 구분한다.



➢ 홀드아웃 검증 → 교차 검증용 데이터와 별개로 최종 정확도 검증을 위한 데이터를 미리 준비후 검증한다.

머신러닝 검증을 위한 데이터 레코드 분할 - 교차검증

➢ 제조 레코드를 사용하여 예측모델 구축을 위한 데이터를 분할한다.

type	length	thickness	fault_flg
B	-34.743311	-1.5865954	TRUE
E	-10.789816	-0.2620702	FALSE
E	9.228733	0.433382	FALSE
C	147.110538	26.6938774	FALSE
D	1.36317	0.166149	FALSE
E	-7.896265	-0.9626665	FALSE
B	-50.787817	-4.4510008	TRUE
B	-28.373147	-1.3936702	FALSE
E	-43.342262	-5.8571695	FALSE
C	72.473747	12.4748105	FALSE
C	108.399187	15.5815358	TRUE
B	15.131973	2.2487379	FALSE
B	-66.839579	-11.473178	FALSE
E	-82.598978	-5.2011388	FALSE
E	-30.580873	-3.0772432	FALSE
E	-74.312738	-9.7427552	FALSE
B	58.865154	2.2261563	FALSE
A	-62.484811	-8.8444766	FALSE
B	50.858233	1.9775838	FALSE
B	-73.465397	-1.8891682	FALSE

분할

홀드아웃 검증의 검증 데이터

type	length	thickness	fault_flg
D	72.473747	12.4748056	FALSE
C	108.399189	15.5815358	TRUE
B	-66.839579	-11.473178	FALSE
E	58.865155	2.2261563	FALSE

홀드아웃 검증의 학습 데이터

type	length	thickness	fault_flg
B	-34.743311	-1.5865954	TRUE
E	-10.789816	-0.2620702	FALSE
E	9.228733	0.433282	FALSE
C	147.110538	26.6938774	FALSE
D	1.36317	0.166149	FALSE
B	-7.896295	-0.9626656	TRUE
B	-50.878717	-4.4510008	FALSE
E	-28.377433	-1.3962072	TRUE
D	-43.324622	-5.8571695	FALSE
C	15.131973	2.2473578	FALSE
B	-82.759878	-6.0711859	FALSE
E	-30.580373	-3.0772432	FALSE
C	-60.202878	-6.0884156	FALSE
E	-62.484811	-8.8444766	FALSE
A	50.858233	1.9775838	FALSE
D	-73.465397	-1.8891682	FALSE

교차 검증의 검증 데이터

type	length	thickness	fault_flg
E	9.228733	0.433328	FALSE
E	-30.580873	-3.077243	FALSE
C	147.110538	26.693877	FALSE
E	-28.377433	-1.396028	TRUE

교차 검증의 학습 데이터

type	length	thickness	fault_flg
C	15.113713	2.248736	FALSE
D	1.36317	0.166149	FALSE
B	-73.465397	-1.889168	FALSE
E	-60.622768	-5.038815	FALSE
D	-7.879625	-0.962665	FALSE
B	50.858233	1.977583	FALSE
C	-43.324622	-5.857169	FALSE
B	-60.787177	-4.451	FALSE
B	-34.743311	-1.586596	TRUE
E	-82.739878	-5.071859	FALSE
B	-86.539575	-11.473178	FALSE
E	-10.789816	-0.26207	FALSE

머신러닝 검증을 위한 데이터 레코드 분할 - 교차검증

➢ 제조 레코드를 사용하여 예측모델 구축을 위한 데이터를 분할한다.

```
production_tb = load_production()
```

production 데이터 로드

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

train_data, test_data, train_target, test_target = \
  train_test_split(production_tb.drop('fault_flg', axis=1),
                   production_tb[['fault_flg']],
                   test_size=0.2)

train_data.reset_index(inplace=True, drop=True)
test_data.reset_index(inplace=True, drop=True)
train_target.reset_index(inplace=True, drop=True)
test_target.reset_index(inplace=True, drop=True)

row_no_list = list(range(len(train_target)))

k_fold = KFold(n_splits=4, shuffle=True)

for train_cv_no, test_cv_no in k_fold.split(row_no_list):
  train_cv = train_data.iloc[train_cv_no, :]
  test_cv = train_data.iloc[test_cv_no, :]
```

홀드아웃 검증을 위한 분할.
모델 학습시 정답이 없어야 하므로
fault_flg 컬럼 제거

테스트 데이터 20%

정답부분만 추출

교차 검증을 위한 분할

4조각으로 나눠 학습3, 검증1로 사용

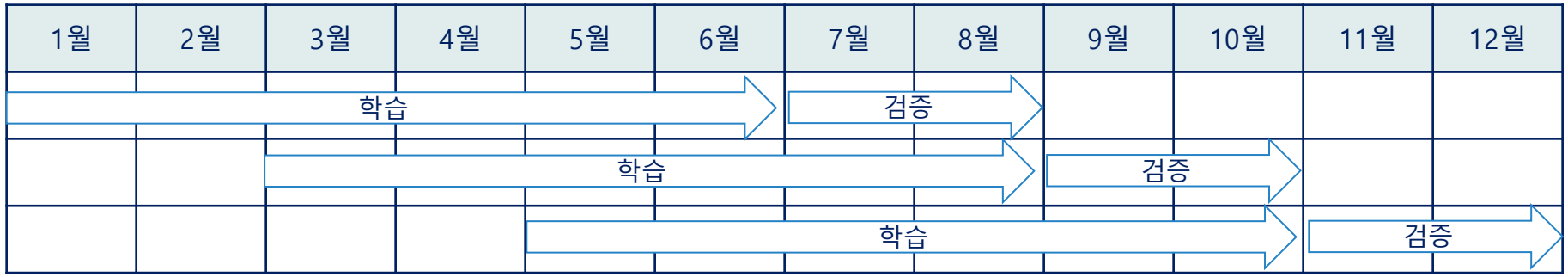
4개로 나뉜 데이터 인덱스중 학습용/
검증용 인덱스를 반환

awesomebook-master/preprocess/005_split/01/python_awesome.py

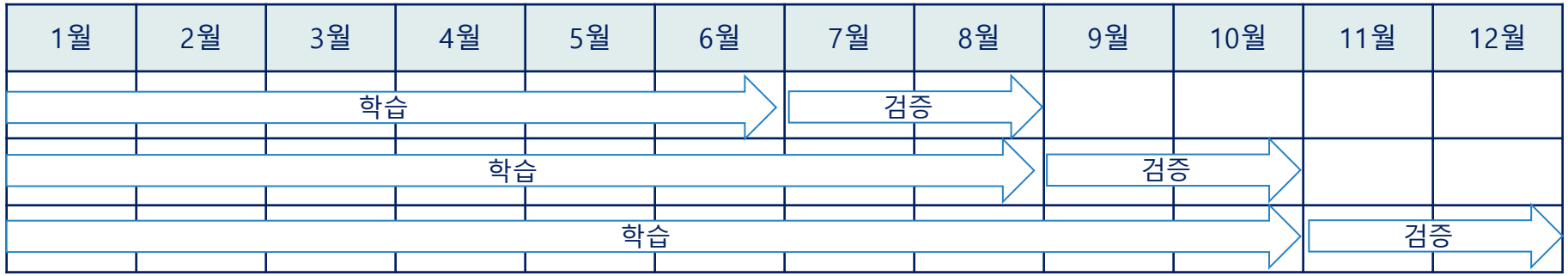
머신러닝 검증을 위한 시간 데이터 분할

➢ 교차 검증시 시간 데이터를 이용할 경우 과거데이터로 학습하고 미래 데이터로 검증하여야 한다.

학습 기간이 일정한 경우



학습 기간이 늘어나는 경우



머신러닝 검증을 위한 시간 데이터 분할

➢ 교차 검증시 시간 데이터를 이용할 경우 과거데이터로 학습하고 미래 데이터로 검증하여야 한다.

```
monthly_index_tb = load_monthly_index()

train_window_start = 1
train_window_end = 24
horizon = 12
skip = 12

monthly_index_tb.sort_values(by='year_month')

while True:
  test_window_end = train_window_end + horizon
  train = monthly_index_tb[train_window_start:train_window_end]
  test = monthly_index_tb[(train_window_end + 1):test_window_end]
  if test_window_end >= len(monthly_index_tb.index):
    break
  train_window_start += skip
  train_window_end += skip
```

학습데이터 시작 시점

학습데이터 종료 시점

검증 기간

윈도우 이동 기간

년월로 정렬

검정 종료 시기 설정

학습 데이터 추출

테스트 데이터 추출

데이터 소모시 종료

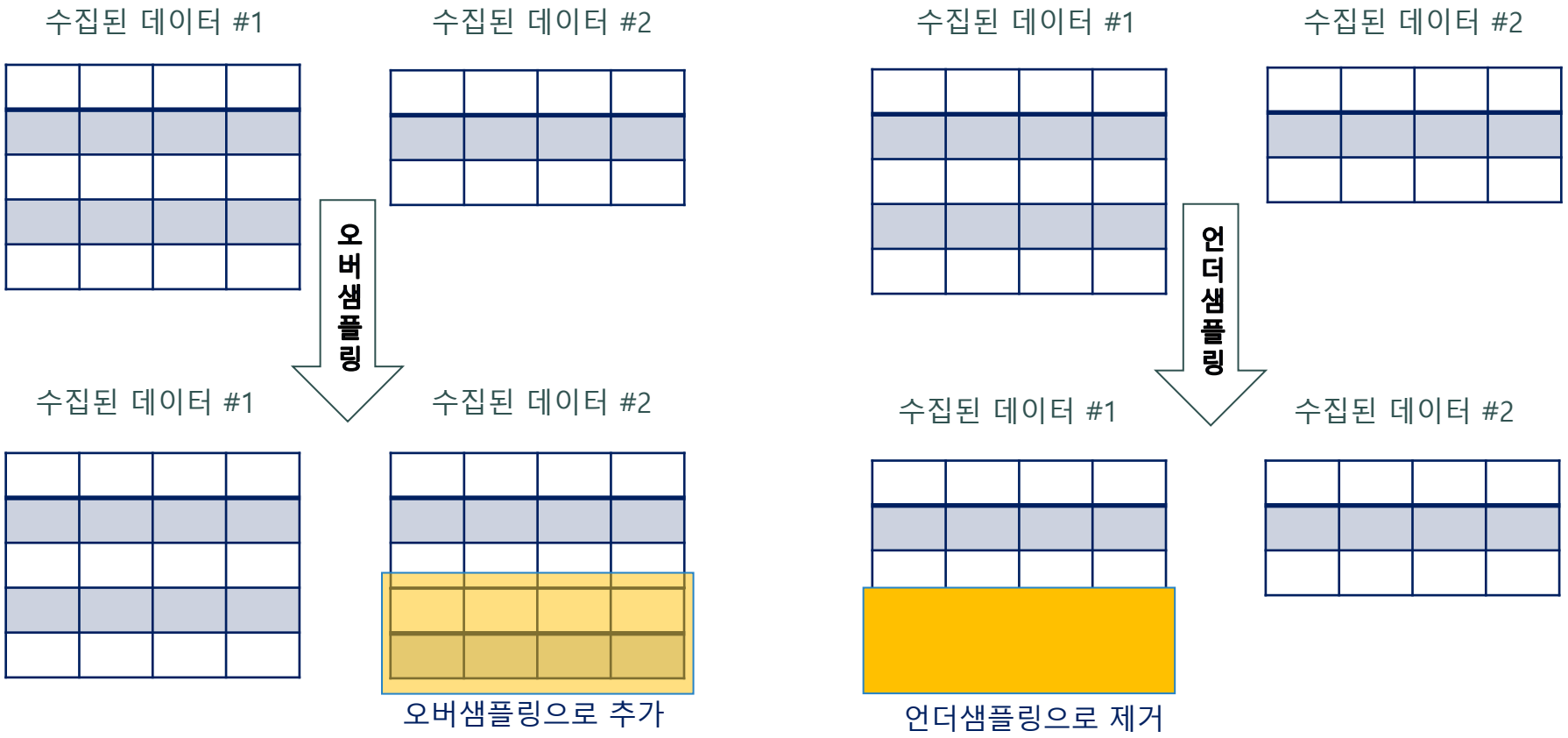
검증기간만큼 윈도우 이동

awesomebook-master/preprocess/005_split/02/python_awesome.py

1. 필요한 데이터만 추출하기
2. 분석 단위를 손실 없이 변경하기
3. 여러 테이블 합치기
4. 학습용과 검증용 데이터 나누기
5. **불균형한 데이터를 보정용 데이터로 생성하기**
6. 집계 데이터를 표 형식으로 바꾸기

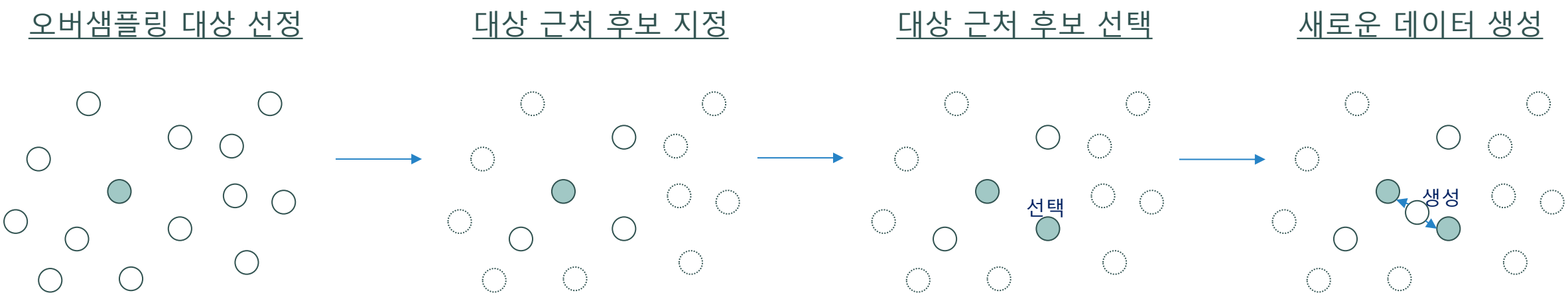
오버 샘플링과 언더샘플링

- 데이터를 억지로 늘려도 데이터의 가치는 전과 거의 차이는 없다.
- 그럼에도 불균형한 데이터를 조정할 때는 데이터를 생성해야 한다.
- 데이터 생성에는 오버샘플링과 언더샘플링의 방법이 있다.



데이터 불균형 조정을 위해 오버샘플링 사용하기

- 오버샘플링은 원본 데이터에서 새로운 데이터를 생성하는 것이다.
- 과학습을 방지하기 위해 SMOTE 기법을 사용한다.
- SMOTE로 생성된 데이터는 원본 데이터와 같은 특성을 유지하지만 약간의 노이즈를 더한 데이터이다.



오버샘플링으로 데이터 불균형 조정하기

➢ 아래 그림과 같이 fault_flg의 불균형을 SMOTE로 해소한다.

type	length	thickness	fault_fg
E	-72.14877	-10.831815	FALSE
E	-92.94186	-11.567282	FALSE
C	144.10377	8.399005	FALSE
C	33.82072	2.489825	FALSE
A	105.2964	7.906002	FALSE
E	-53.85227	-9.150479	FALSE
B	-73.63264	-10.356727	FALSE
A	24.41563	1.391697	FALSE
E	-76.58208	-8.47837	FALSE
C	114.92684	1.56777	TRUE

SMOTE를 활용하여 length/thickness 대비 falut_flg 불균형성 해소

length	thickness	fault_fg
114.92684	1.5677796	TRUE
-72.14877	-10.831815	FALSE
-92.94186	-11.567282	FALSE
144.10377	8.399005	FALSE
33.82072	2.4898248	FALSE
105.2964	7.906002	FALSE
-53.85227	-9.1504794	FALSE
24.41563	1.391697	FALSE
-76.58208	-8.47837	FALSE
24.41583	1.3916971	TRUE
21.91676	1.9785185	TRUE
-53.73975	-9.150479	TRUE
116.41273	6.396213	TRUE
71.51077	3.032397	TRUE
-24.56531	-11.27385	TRUE
84.11853	0.9420974	TRUE
26.11059	2.030735	TRUE
-57.36879	-8.9145347	TRUE
69.12726	1.1039784	TRUE
98.44214	10.307	TRUE

오버샘플링으로 데이터 불균형 조정하기

➢ 아래 그림과 같이 fault_flg의 불균형을 SMOTE로 해소한다.

주의1> imbalancelearn 패키지를 추가 설치해야 함
pip install imbalanced-learn
주의2> 원본의 ratio는 sampling_strategy 로 변경
주의3> 원본의 fit_sample은 fit_resample로 변경

```
production_tb = load_production()

from imblearn.over_sampling import SMOTE

sm = SMOTE(sampling_strategy='auto', k_neighbors=5, random_state=71)

blance_data, balance_target = \
  sm.fit_resample(production_tb[['length', 'thickness']],
                  production_tb['fault_flg'])
```

후보수

무작위 샘플링 을 위한 seed

length와 thickness 데이터 생성

fault_flg가 균형 잡히도록 데이터 생성

awesomebook-master/preprocess/006_generate/02/python_awesome.py

1. 필요한 데이터만 추출하기
2. 분석 단위를 손실 없이 변경하기
3. 여러 테이블 합치기
4. 학습용과 검증용 데이터 나누기
5. 불균형한 데이터를 보정용 데이터로 생성하기
6. **집계 데이터를 표 형식으로 바꾸기**

개요

- 전개(Spread)는 데이터 집계 결과를 표 형식으로 변환하는 것이다.
- 전개를 통해 간단한 집계 처리의 결과를 쉽게 하거나 추천 항목에 사용할 데이터를 준비할때 이용한다.
- 참고: 가로 데이터와 세로 데이터

세로 데이터

연령대	성별	인원수
20	남성	50
20	여성	37
30	남성	64
30	여성	68
40	남성	57
40	여성	49

가로 데이터

연령대	남성 수	여성 수
20	50	37
30	64	68
40	57	49

세로 데이터에서 가로데이터로 변환

➢ 예약 테이블(세로데이터)에서 행을 고객ID, 열을 투숙객 수, 값을 예약 건수인 표(가로데이터)로 변환한다.

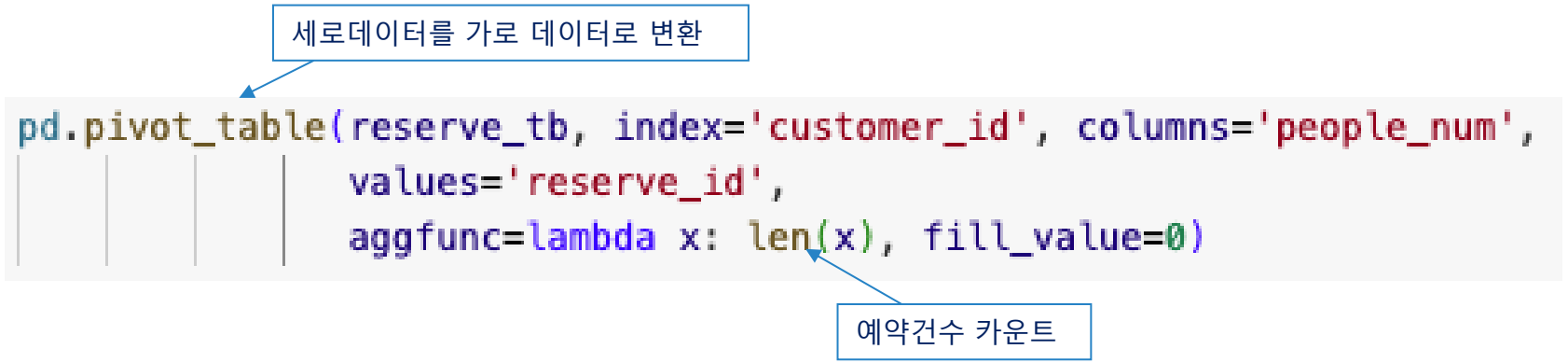
reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_dat	people_num	total_price
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	13:00:00	2016.7.21	2	32400
r3	h_179	c_1	2017.3.5 9:36	2017.3.19	15:00:00	2017.3.20	2	32400
r4	h_214	c_1	2017.3.8 3:00	2017.3.29	12:00:00	2017.3.30	3	48600
r5	h_221	c_1	2017.10.12 12:47	2017.10.22	15:00:00	2017.10.23	2	32400
r6	h_191	c_1	2017.11.27 8:39	2017.12.2	16:00:00	2017.12.3	1	16200
r7	h_231	c_1	2018.5.6 23:16	2018.5.8	13:00:00	2018.5.9	1	16200
r8	h_241	c_2	2016.3.26 19:05	2016.3.31	13:00:00	2016.4.1	2	32400
r9	h_240	c_2	2017.8.13 10:00	2017.8.16	15:00:00	2017.8.18	3	48600
r10	h_183	c_2	2017.10.19 3:06	2017.10.21	14:00:00	2017.10.22	2	32400
r11	h_188	c_2	2018.1.6 4:31	2018.1.8	12:00:00	2018.1.9	1	16200
r12	h_223	c_2	2018.2.15 10:44	2018.2.16	12:00:00	2018.2.17	2	32400
r13	h_223	c_2	2018.2.16 9:18	2018.2.17	12:00:00	2018.2.18	1	16200
r14	h_92	c_2	2018.4.19 11:25	2018.4.20	12:00:00	2018.4.21	4	64800
r15	h_92	c_2	2018.7.6 4:18	2018.7.8	12:00:00	2018.7.9	4	64800

people_num 값에 따른 customer별 예약 건수로 변환

customer_id	1	2	3	4
c_1	2	2	1	1
c_2	2	2	1	2

가로데이터로 변환

➢ 예약 테이블(세로데이터)에서 행을 고객ID, 열을 투숙객 수, 값을 예약 건수인 표(가로데이터)로 변환한다.



희소 행렬로의 변환

- 희소 행렬 → 대부분 요소의 값이 0이고 극히 일부만 값을 가지는 거대한 행렬(표)
- 데이터 특성에 따라서 가로 데이터로 변환하면 희소 행렬로 되는 경우가 발생한다.
- 가로 데이터를 데이터 크기가 커지지 않도록 하려면 세로 데이터의 표현을 유지한채 표로 만들어야 한다.
- scipy.sparse 에서 제공하는 다양한 자료형의 희소 행렬
  - lil_matrix : matrix의 값을 갱신하는 것이 빠르고 연산처리가 느린 형식
  - csr_matrix : 행에 대한 접근이 빠르고 연산처리가 빠른 형식
  - csc_matrix : 열에 대한 접근이 빠르고 연산 처리가 빠른 형식
- 순차적인 데이터를 갱신할때는 lil_matrix, 연산처리를 다룬다면 csr_matrix 또는 csc_matrix 사용한다.
- 각 matrix는 상호간 변환이 가능하다 → tolil, tocsr, tocsc 함수 사용

희소 행렬로의 변환

```
from scipy.sparse import csc_matrix

cnt_tb = reserve_tb \
  .groupby(['customer_id', 'people_num'])['reserve_id'].size() \
  .reset_index()
cnt_tb.columns = ['customer_id', 'people_num', 'rsv_cnt']

customer_id = pd.Categorical(cnt_tb['customer_id'])
people_num = pd.Categorical(cnt_tb['people_num'])

sparse_matrix = csc_matrix((cnt_tb['rsv_cnt'], (customer_id.codes, people_num.codes)),
                           shape=(len(customer_id.categories), len(people_num.categories)))


print(sparse_matrix.toarray())
```

카테고리형으로
행과 열 준비

카테고리형개수에 따른
행열을 가지는 행렬생성