빅데이터 전처리

3회차 – 데이터의 세부내용 전처리

1. 수치 데이터 전처리 기법

2. 범주 데이터 전처리 기법

3. 일시 데이터 전처리 기법

4. 위치정보 데이터 전처리 기법

수치 데이터 형식으로의 변환

- 원칙적으로 수치형 데이터열은 수치형으로 자동 변환된다.
- 값에 문자가 섞이거나 정수형/부동소수점형의 변환이 필요한 경우 전처리 작업이 필요하다.

```
print(type(40000/3))

print(int(40000 / 3))

print(float(40000 / 3))
```

데이터 형 확인

정수형 변환

실수형 변환

```
df = pd.DataFrame({'value': [40000 / 3]})

print(df.dtypes)

print(df['value'].astype('int8'))
print(df['value'].astype('int16'))
print(df['value'].astype('int32'))
print(df['value'].astype('int64'))

print(df['value'].astype('float16'))
print(df['value'].astype('float32'))
print(df['value'].astype('float64'))
print(df['value'].astype('float128'))

print(df['value'].astype(int))
print(df['value'].astype(float))
```

다양한 사이즈의 정수형으로 변환

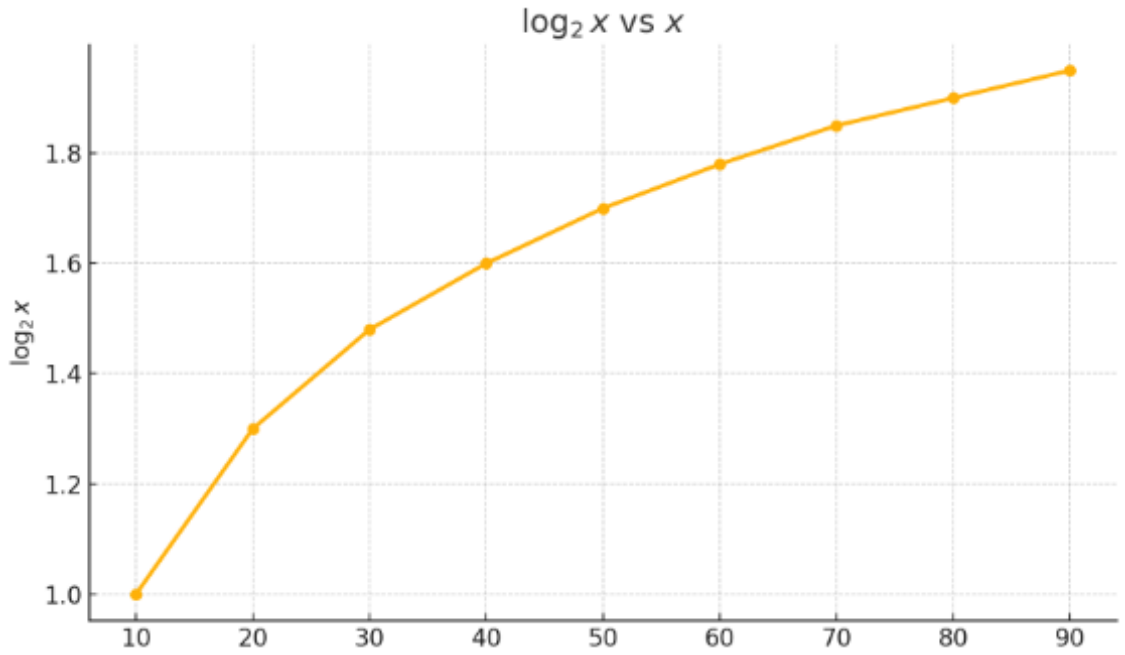
다양한 사이즈의 실수형으로 변환

데이터형 지정 가능

로그 변환 등 지수기반 스케일링

- 선형 모델로 표현할 수 없는 문제를 비선형 모델로 다루기 위해 수치를 변환하는 전처리가 필요하다.
- $b = \log_a x$ 식을 이용하여 b 값을 구하는 방법을 대수화라고 한다.
- 예> 10대와 20대의 키 차이와 60대와 70대의 키 차이는 달라야 한다.

x	$\log_2 x$
10	1
20	1.3
30	1.48
40	1.6
50	1.7
60	1.78
70	1.85
80	1.9
90	1.95



로그 변환 등 지수기반 스케일링

➢ hotel_price를 대수화 하여 전처리한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_da	people_num	total_price
r1	h_75	c_1	2016.3.6 13:00	2016.3.26	10:00:00	2016.3.29	2	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:00:00	2016.7.21	2	20600
r3	h_179	c_1	2016.9.26 14:00	2016.10.3	12:00:00	2016.10.5	3	33600
r4	h_214	c_1	2017.3.8 20:03	2017.3.22	12:00:00	2017.3.24	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.29	12:00:00	2017.10.2	4	194400
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000

total_price를 1000으로 나누고
1을 더해 대수화한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_da	people_num	total_price	total_price_log
r1	h_75	c_1	2016.3.6 13:00	2016.3.26	10:00:00	2016.3.29	2	97200	1.99211
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:00:00	2016.7.21	2	20600	1.33445
r3	h_179	c_1	2016.9.26 14:00	2016.10.3	12:00:00	2016.10.5	3	33600	1.52891
r4	h_214	c_1	2017.3.8 20:03	2017.3.22	12:00:00	2017.3.24	4	194400	2.28903
r5	h_16	c_1	2017.9.5 19:50	2017.9.29	12:00:00	2017.10.2	4	194400	2.28903
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000	1.5682

```
reserve_tb['total_price_log'] = \
  reserve_tb['total_price'].apply(lambda x: np.log10(x / 1000 + 1))

reserve_tb['total_price_log']
```

x가 0일경우 log(0)
를 피하기 위함

순위 또는 분위수를 활용한 스케일링

- 범주화란 수치를 다수의 플래그값(TRUE or FALSE)으로 변환하는 것을 뜻한다.

연령	0~9세 플래그	10~19세 플래그	20~59세 플래그	60세 이상 플래그
9세	TRUE	FALSE	FALSE	FALSE
15세	FALSE	TRUE	FALSE	FALSE
27세	FALSE	FALSE	TRUE	FALSE
39세	FALSE	FALSE	TRUE	FALSE
58세	FALSE	FALSE	TRUE	FALSE
64세	FALSE	FALSE	FALSE	TRUE

- 대수화만으로 복잡한 변화를 충분히 표현할 수 없을때 범주화를 사용한다.

순위 또는 분위수를 활용한 스케일링 – 수치형의 범주화

➢ 고객 테이블 나이를 10단위의 범주형으로 추가해 본다.

customer_id	age	sex	home_latituc	home_longitude
c_1	41	man	35.0219	136.5123
c_2	38	man	35.32508	139.4105
c_3	49	woman	35.12054	136.5112
c_4	43	man	43.01847	141.2403
c_5	31	man	35.10266	136.5233
c_6	52	man	34.44077	135.8905

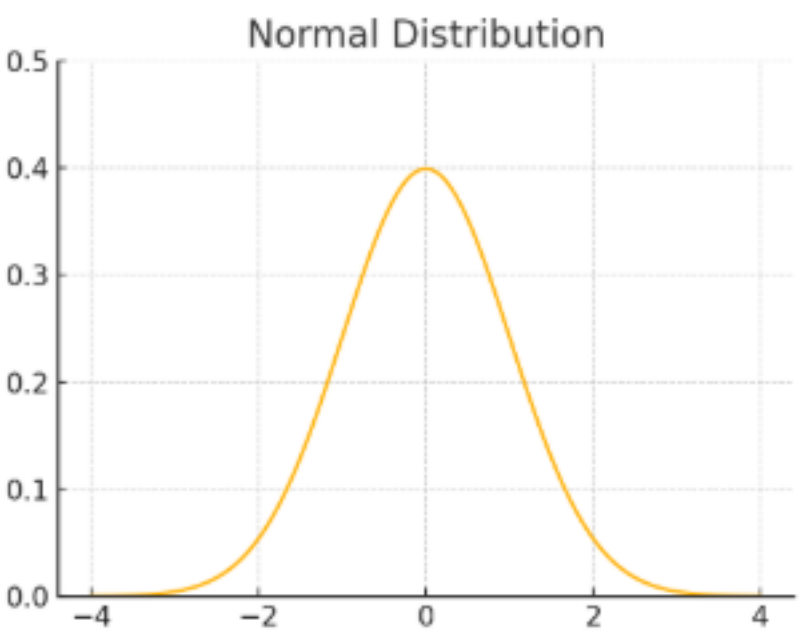
age를 10단위로 범주화한다.

customer_id	age	sex	home_latituc	home_longit	age_rank
c_1	41	man	35.0219	136.5123	40
c_2	38	man	35.32508	139.4105	30
c_3	49	woman	35.12054	136.5112	40
c_4	43	man	43.01847	141.2403	40
c_5	31	man	35.10266	136.5233	30
c_6	52	man	34.44077	135.8905	50

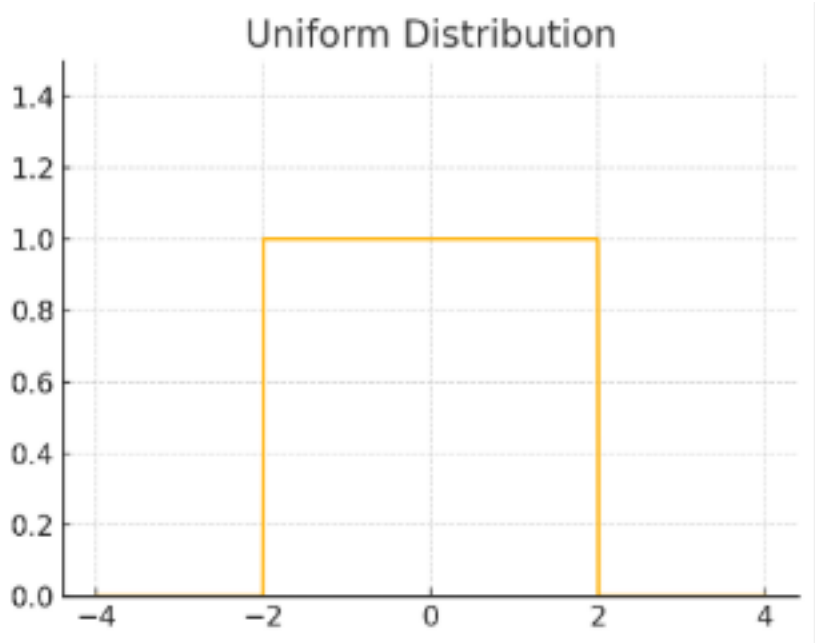
```
customer_tb['age_rank'] = \
  (np.floor(customer_tb['age'] / 10) * 10).astype('category')

print(customer_tb['age_rank'])
```

정규화

- 값이 가질수 있는 범위를 통일하는 변환처리를 의미한다.
- 머신러닝 모델의 입력값이 가질 수 있는 범위를 균일하게 유지하는 역할을 한다.
- 정규화의 두가지 방법
  - 평균 0, 분산 1로 변환하는 정규화
  - 최소값 0, 최대값 1로 변환하는 정규화



[평균1, 분산1로 변환하는 정규화]



[최소값0, 최대값1로 변환하는 정규화]

정규화

➢ 예약인수와 합계 금액을 평균0, 분산1의 분포로 변환하여 정규화한다.

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 13:00	2016.3.26	10:00:00	2016.3.29	2	97200
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:00:00	2016.7.21	2	20600
r3	h_179	c_1	2016.9.26 14:00	2016.10.3	12:00:00	2016.10.5	3	33600
r4	h_214	c_1	2017.3.8 2:01	2017.3.22	12:00:00	2017.3.24	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.29	12:00:00	2017.10.2	4	194400
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000

people_num과 total_price를 정규화한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price	people_num	total_price_normalized
r1	h_75	c_1	2016.3.6 13:00	2016.3.26	10:00:00	2016.3.29	2	97200	-1.300547	-0.0571874
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:00:00	2016.7.21	2	20600	-1.300547	-1.0833677
r3	h_179	c_1	2016.9.26 14:00	2016.10.3	12:00:00	2016.10.5	3	33600	0.04348487	-0.8483125
r4	h_214	c_1	2017.3.8 2:01	2017.3.22	12:00:00	2017.3.24	4	194400	1.387517	1.6800187
r5	h_16	c_1	2017.9.5 19:50	2017.9.29	12:00:00	2017.10.2	4	194400	1.387517	1.6800187
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	36000	0.04348487	-0.3711887

정규화

- 예약인수와 합계 금액을 평균0, 분산1의 분포로 변환하여 정규화한다.

```
from sklearn.preprocessing import StandardScaler

reserve_tb['people_num'] = reserve_tb['people_num'].astype(float)

ss = StandardScaler()

result = ss.fit_transform(reserve_tb[['people_num', 'total_price']])

reserve_tb['people_num_normalized'] = [x[0] for x in result]
reserve_tb['total_price_normalized'] = [x[1] for x in result]

reserve_tb['people_num_normalized']
```

소수점을 다루기 위해 실수형 변환

정규화 객체 생성

정규화 변환 함수

이상값 처리

- 대부분 값보다 극단적으로 크거나 작은 값, 즉 예외값의 영향으로 문제가 발생하기도 한다.
- 이는 정규화나 예측모델 구축시 나쁜 영향을 끼칠 때가 많으므로 전처리 단계에서 제거해야 한다.
- 정규분포를 전제로 한 예외값 검출방법
  - 표준편차의 일정 배수 이상 떨어진 값을 평균값에서 제거하는 방법
  - 정규분포에 따른 값은 평균값에서 표준편차의 세 배 이내 범위에 99.73%가 있으므로 0.27%를 예외값으로 봄

이상값 처리

➢ 예약 테이블의 예약 합계금액의 평균값에서 표준편차의 세 배 이외 값을 삭제한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price
r1001	h_6	c_244	2018.5.4 1:46	2018.5.11	10:00:00	2018.5.14	2	297000
r1002	h_290	c_244	2018.9.26 7:41	2018.10.20	11:00:00	2018.10.23	4	190800
r1003	h_176	c_244	2018.8.8 12:14	2018.10.20	11:00:00	2018.10.23	3	333600
r1004	h_214	c_245	2016.7.31 13:22	2016.8.3	12:00:00	2016.8.6	3	437400
r1005	h_176	c_246	2016.2.29 19:45	2016.3.5	12:00:00	2016.3.6	4	194400
r1006	h_172	c_247	2016.3.14 17:08	2016.3.28	12:00:00	2016.3.30	3	128000

total_price를 기준으로
이상값을 제거한다.

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price
r1001	h_6	c_244	2018.5.4 1:46	2018.5.11	10:00:00	2018.5.14	2	297000
r1002	h_290	c_244	2018.9.26 7:41	2018.10.20	11:00:00	2018.10.23	4	190800
r1005	h_176	c_246	2016.2.29 19:45	2016.3.5	12:00:00	2016.3.6	4	194400
r1006	h_172	c_247	2016.3.14 17:08	2016.3.28	12:00:00	2016.3.30	3	128000

```
reserve_tb = reserve_tb[
  (abs(reserve_tb['total_price'] - np.mean(reserve_tb['total_price'])) /
   np.std(reserve_tb['total_price']) <= 3)
].reset_index()
```

주성분 기반 차원 축소

- 입력값의 종류가 많을수록 머신러닝 모델이 경향을 학습하는데 필요한 데이터 양이 커진다.
- 주성분 분석을 이용하여 차원(입력값의 종류)을 줄이면 데이터 양이 줄어들어 쉽게 학습을 할 수 있다.
- 예> X와 Y의 상관관계를 이용하여 Z 로 압축하는 방법
  - x, y의 2차원을 z의 1차원으로 압축함.

Data No.	X	Y
1	3	6
2	2	4
3	5	9
4	9	18
5	7	13

Z
6.7
4.5
11.2
20.1
15.7

주성분 기반 차원 축소

- PCA 클래스를 이용하여 주성분 분석을 구현할 수 있다.

```
production_tb = load_production()

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

pca_values = pca.fit_transform(production_tb[['length', 'thickness']])

print('누적 기여율: {0}'.format(sum(pca.explained_variance_ratio_)))
print('각 차원의 기여율: {0}'.format(pca.explained_variance_ratio_))

pca_newvalues = pca.transform(production_tb[['length', 'thickness']])

pca_newvalues
```

2개 주성분으로 PCA 객체 생성

PCA 학습 및 변환

누적 기여율

각 주성분이 원본 데이터 정보를 얼마나 잘 설명하는지 여부

변환만 다시 한번 수행해봄

결측값 처리 기법 – 누락된 행 제거

➢ thickness에 결손이 발생한 제조 레코드 사용시 결손된 레코드를 제거한다.

type	length	thickness	fault_flg
E	142.0689	11.418895	FALSE
D	121.2394	NA	FALSE
B	273.6399	5.0000496	FALSE
E	177.5333	9.7220772	FALSE
A	128.4934	7.286805	FALSE
E	180.5049	29.648748	FALSE
A	222.9105	2.0628351	FALSE
C	246.9794	46.382952	FALSE
E	195.0203	15.556976	FALSE
D	165.4061	7.040498	FALSE
E	182.6259	21.818294	FALSE
A	193.0016	NA	FALSE

thickness의 NA 레코드를 제거

type	length	thickness	fault_flg
E	142.0689	11.418895	FALSE
B	273.6399	5.0000496	FALSE
E	177.5333	9.7220772	FALSE
A	128.4934	7.286805	FALSE
E	180.5049	29.648748	FALSE
A	222.9105	2.0628351	FALSE
C	246.9794	46.382952	FALSE
E	195.0203	15.556976	FALSE
D	165.4061	7.040498	FALSE
E	182.6259	21.818294	FALSE

```
production_miss_num = load_production_missing_num()

production_miss_num.replace('None', np.nan, inplace=True)

production_miss_num.dropna(subset=['thickness'], inplace=True)
```

None을 nan으로 교체

dropna로 nan 레코드 제거

결측값 처리 기법 – 고정값 대체

- thickness에 결손이 발생한 제조 레코드 사용시 1로 보완한다.

type	length	thickness	fault_flg
E	142.0689	11.418895	FALSE
D	121.2394	NA	FALSE
B	273.6399	5.0000496	FALSE
E	177.5333	9.7220772	FALSE
A	128.4934	7.286805	FALSE
E	180.5049	29.648748	FALSE
A	222.9105	2.0628351	FALSE
C	246.9794	46.382952	FALSE
E	195.0203	15.556976	FALSE
D	165.4061	7.040498	FALSE
E	182.6259	21.818294	FALSE
A	193.0016	NA	FALSE

thickness의 NA 값을 1로 채움

type	length	thickness	fault_flg
E	142.0689	11.418895	FALSE
D	121.2394	1	FALSE
B	273.6399	5.0000496	FALSE
E	177.5333	9.7220772	FALSE
A	128.4934	7.286805	FALSE
E	180.5049	29.648748	FALSE
A	222.9105	2.0628351	FALSE
C	246.9794	46.482952	FALSE
E	195.0203	15.556976	FALSE
D	165.4061	7.040498	FALSE
E	182.6259	21.818294	FALSE
A	193.0016	1	FALSE

```
production_miss_num.replace('None', np.nan, inplace=True)

production_miss_num['thickness'].fillna(1, inplace=True)
```

None을 nan으로 교체

nan 값을 1로 채움

결측값 처리 기법 – 평균값 대체

➢ thickness에 결손이 발생한 제조 레코드 사용시 평균값으로 보완한다.

type	length	thickness	fault_flg
E	142.0689	11.418895	FALSE
D	121.2394	NA	FALSE
B	273.6399	5.0000496	FALSE
E	177.5333	9.7220772	FALSE
A	128.4934	7.286805	FALSE
E	180.5049	29.648748	FALSE
A	222.9105	2.0628351	FALSE
C	246.9794	46.382952	FALSE
E	195.0203	15.556976	FALSE
D	165.4061	7.040498	FALSE
E	182.6259	21.818294	FALSE
A	193.0016	NA	FALSE

thickness의 NA 값을 평균으로 채움

type	length	thickness	fault_flg
E	142.0689	11.418895	FALSE
D	121.2394	21.18521	FALSE
B	273.6399	5.0000496	FALSE
E	177.5333	9.7220772	FALSE
A	128.4934	7.286805	FALSE
E	180.5049	29.648748	FALSE
A	222.9105	2.0628351	FALSE
C	246.9794	46.482952	FALSE
E	195.0203	15.556976	FALSE
D	165.4061	7.040498	FALSE
E	182.6259	21.818294	FALSE
A	193.0016	21.18521	FALSE

```
production_miss_num.replace('None', np.nan, inplace=True)

production_miss_num['thickness'] = \
  production_miss_num['thickness'].astype('float64')

thickness_mean = production_miss_num['thickness'].mean()

production_miss_num['thickness'].fillna(thickness_mean, inplace=True)
```

None을 nan으로 교체

실수값으로 변경

평균 구하기

nan을 평균값으로 교체

1. 수치 데이터 전처리 기법

2. 범주 데이터 전처리 기법

3. 일시 데이터 전처리 기법

4. 위치정보 데이터 전처리 기법

범주형 데이터로의 변환

- 범주형은 가질수 있는 값의 종류가 정해진 값을 의미한다.
- 호텔 예약 레코드를 사용하여 성별을 불리언형과 범주형으로 변환해 본다.

customer_id	age	sex	home_latit	home_long
c_1	41	man	35.09219	136.5123
c_2	38	man	35.32508	139.4106
c_3	49	woman	35.12054	136.5112
c_4	43	man	43.03487	141.2403
c_5	31	man	35.10266	136.5233
c_6	52	man	34.44077	135.3905

성별을 판별
범주화 함

customer_id	age	sex	home_latit	home_long	sex_is_man	sex_c
c_1	41	man	35.09219	136.5123	TRUE	man
c_2	38	man	35.32508	139.4106	TRUE	woman
c_3	49	woman	35.12054	136.5112	FALSE	man
c_4	43	man	43.03487	141.2403	TRUE	man
c_5	31	man	35.10266	136.5238	TRUE	man
c_6	52	man	34.44077	135.3905	TRUE	man

범주형 데이터로의 변환

- 범주형은 가질수 있는 값의 종류가 정해진 값을 의미한다.
- 호텔 예약 레코드를 사용하여 성별을 불리언형과 범주형으로 변환해 본다.

```
customer_tb[['sex_is_man']] = (customer_tb[['sex']] == 'man').astype('bool')

customer_tb['sex_c'] = \
  pd.Categorical(customer_tb['sex'], categories=['man', 'woman'])

customer_tb['sex_c'].cat.codes

customer_tb['sex_c'].cat.categories
```

성별이 man일때 TRUE가 되는 bool 타입 추가

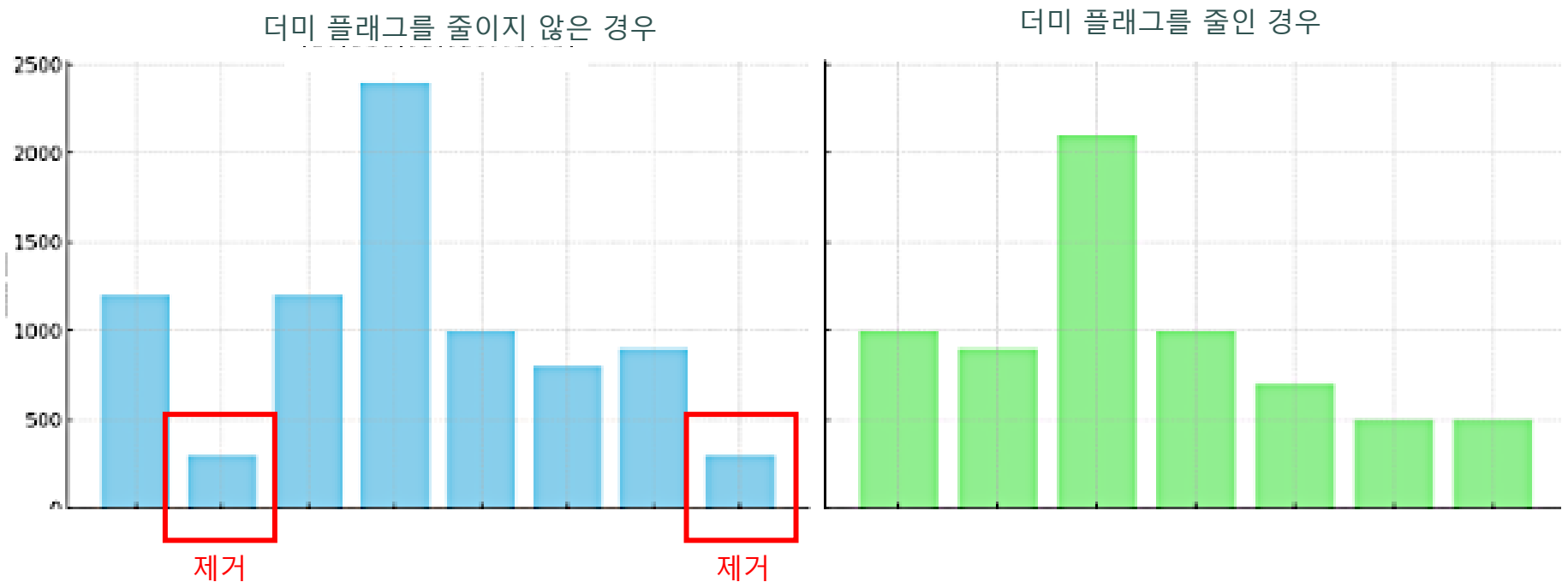
sex_c 라는 범주형 데이터 추가

인덱스 데이터는 codes에 저장됨

마스터 데이터는 categories 에 저장됨

더미 변수화

- 파이썬 머신러닝에서 사용되는 일부 함수에서 범주형을 지원하지 않는 경우가 있다.
- 이 경우 범주값을 플래그의 집합값으로 변환해야 하며 이를 더미 변수화라고 한다.
- 더미 변수를 줄이면 머신러닝 예측모델에서 학습데이터를 줄일수 있지만 성능에 영향을 줄 수 있다.



더미 변수화

➢ 고객 테이블의 성별을 더미 변수로 만들어 본다.

customer_id	age	sex	home_latitude	home_longitude
c_1	41	man	35.09219	136.5123
c_2	38	man	35.32508	139.4106
c_3	49	woman	35.12054	136.5112
c_4	43	man	43.03487	141.2403
c_5	31	man	35.10266	136.5238
c_6	52	man	34.44077	135.3905

sex를 더미 변수화한다

customer_id	age	sex	home_latitude	home_longitude	sexman	sexwoman
c_1	41	man	35.09219	136.5123	1	0
c_2	38	man	35.32508	139.4106	1	0
c_3	49	woman	35.12054	136.5112	0	1
c_4	43	man	43.03487	141.2403	1	0
c_5	31	man	35.10266	136.5238	1	0
c_6	52	man	34.44077	135.3905	1	0

```
customer_tb['sex'] = pd.Categorical(customer_tb['sex'])

dummy_vars = pd.get_dummies(customer_tb['sex'], drop_first=False)

dummy_vars
```

sex 를 범주형으로 전환

더미 변수화

모든 카테고리에 대해
각각의 더미 칼럼 생성

범주 그룹 단순화

➢ 데이터 수가 극단적으로 적은 범주값을 다른 범주값과 묶는 방법을 범주값의 집약이라고 한다.

customer_id	age	sex	home_latitude	home_longitude	age_rank
c_20	43	woman	34.44108	135.3925	40
c_21	62	woman	34.47108	135.3221	60
c_22	37	man	34.7354	135.374	30
c_23	65	man	43.04559	141.2342	60
c_24	30	man	35.1382	135.5219	30
c_25	85	woman	35.32228	139.4041	80

60세이상 age_rank를 집약

customer_id	age	sex	home_latitude	home_longitude	age_rank
c_20	43	woman	34.44108	135.3925	40
c_21	62	woman	34.47108	135.3221	60 이상
c_22	37	man	34.7354	135.374	30
c_23	65	man	43.04559	141.2342	60 이상
c_24	30	man	35.1382	135.5219	30
c_25	85	woman	35.32228	139.4041	60 이상

범주 그룹 단순화

- 데이터 수가 극단적으로 적은 범주값을 다른 범주값과 묶는 방법을 범주값의 집약이라고 한다.

```
customer_tb['age_rank'] = \
  pd.Categorical(np.floor(customer_tb['age']/10)*10)

customer_tb['age_rank'] = customer_tb['age_rank'].cat.add_categories(['60 이상'])

customer_tb.loc[customer_tb['age_rank'] \
                .isin([60.0, 70.0, 80.0]), 'age_rank'] = '60 이상'

customer_tb['age_rank'] = customer_tb['age_rank'].cat.remove_unused_categories()
```

age_rank를 범주형으로 전환

'60 이상' 범주 추가

60이상 데이터를 '60 이상' 범주로 집약

미사용 범주 제거
본 실험에서 사용X

KNN 기반 결측 범주 보완

- 범주형 데이터에 결손이 발생하면 결손의 형태에 따라 적절한 대처 방법을 선택해야 한다.
- 결손값의 보완 방법
  - 고정값으로 보완
  - 집계값으로 보완
  - 결손이 발생하지 않은 데이터에 기반한 예측값으로 보완
  - 시간관계로 보완
  - 다중대입법으로 보완
  - 최대가능도로 보완

KNN 기반 결측 범주 보완

➢ KNN(K-nearest neighbor algorithm) 예측을 이용하여 결손을 보완해본다.

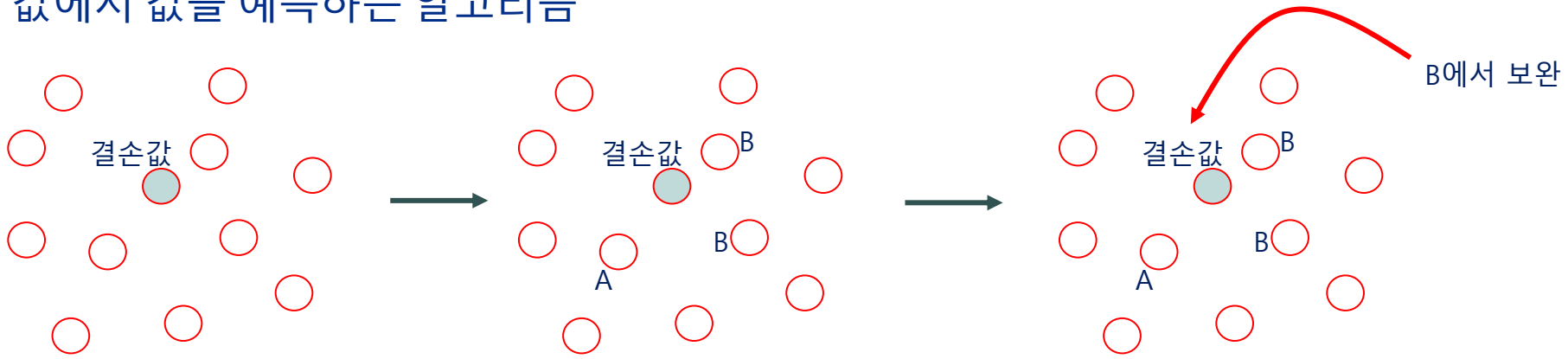
type	length	thickness	fault_flg
	203.379	30.286454	FALSE
B	153.1424	1.104218	FALSE
B	150.3598	10.996555	FALSE
C	249.4818	20.940242	FALSE
E	257.4337	37.156503	FALSE
	157.4632	11.166165	FALSE

type	length	thickness	fault_flg
C	203.379	30.286454	FALSE
B	153.1424	1.104218	FALSE
B	150.3598	10.996555	FALSE
C	249.4818	20.940242	FALSE
E	257.4337	37.156503	FALSE
E	157.4632	11.166165	FALSE

length와 thickness를 기준으로
type을 보완함

KNN 기반 결측 범주 보완

- KNN(K-nearest neighbor algorithm) : 이용중인 변수에 기반한 데이터 사이의 거리로부터 대상 데이터에 가까운 k개의 값에서 값을 예측하는 알고리즘



- KNN(K-nearest neighbor algorithm) 예측을 이용하여 결손을 보완해본다.

type	length	thickness	fault_flg
	203.379	30.286454	FALSE
B	153.1424	1.104218	FALSE
B	150.3598	10.996555	FALSE
C	249.4818	20.940242	FALSE
E	257.4337	37.156503	FALSE
	157.4632	11.166165	FALSE

type	length	thickness	fault_flg
C	203.379	30.286454	FALSE
B	153.1424	1.104218	FALSE
B	150.3598	10.996555	FALSE
C	249.4818	20.940242	FALSE
E	257.4337	37.156503	FALSE
E	157.4632	11.166165	FALSE

lengt와 thickness를 기준으로
type을 보완함

범주값의 보완 – KNN을 이용한 보완

➢ KNN(K-nearest neighbor algorithm) 예측을 이용하여 결손을 보완해본다.

```
production_missc_tb = load_production_missing_category()

from sklearn.neighbors import KNeighborsClassifier

production_missc_tb.replace('None', np.nan, inplace=True)

train = production_missc_tb.dropna(subset=['type'], inplace=False)

test = production_missc_tb \
  .loc[production_missc_tb.index.difference(train.index), :]

kn = KNeighborsClassifier(n_neighbors=3)

kn.fit(train[['length', 'thickness']], train['type'])

test['type'] = kn.predict(test[['length', 'thickness']])
```

KNN 객체 준비

None 데이터→nan으로 변환

비결손데이터 추출

결손데이터 추출

KNN 모델 학습

예측값 계산후 type 보완

1. 수치 데이터 전처리 기법

2. 범주 데이터 전처리 기법

3. 일시 데이터 전처리 기법

4. 위치정보 데이터 전처리 기법

문자열 시간 데이터를 날짜형으로 변환하기

- 일시형에는 Timestamp와 DateTime 같은 자료형이 존재한다.
- 데이터 읽을때 연월일시간의 문자열이나 UNIXTIME 형태는 일시형으로 변환하여야 한다.
- 호텔 예약 레코드를 사용하여 일시형과 날짜형으로 변환해본다.

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

datetime64 형으로 변환

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

날짜 정보 추출

연, 월, 일, 시 등 시간 구성 요소 분리

➢ 예약 테이블의 reserve_datetime에서 연, 월, 일, 시, 분, 초를 추려내보자

reserve_datetime	reserve_date
2016.3.6 13:09	2016.3.6
2016.7.16 23:39	2016.7.16
2016.9.24 10:03	2016.9.24
2017.3.8 2:01	2017.3.8
2017.9.5 19:50	2017.9.5
2017.11.27 18:47	2017.11.27

날짜, 시간 관련
정보 추출

reserve_datetime	reserve_date	month	day_in_r	wday	weekdays	hour	minute	second	format_str
2016.3.6 13:09	2016.3.6	3	31	7	일요일	13	9	42	2016.3.6 13:09
2016.7.16 23:39	2016.7.16	7	31	7	토요일	23	39	55	2016.7.16 23:39
2016.9.24 10:03	2016.9.24	9	24	7	토요일	10	3	17	2016.9.24 10:03
2017.3.8 2:01	2017.3.8	3	8	3	수요일	2	1	10	2017.3.8 2:01
2017.9.5 19:50	2017.9.5	9	5	3	화요일	19	50	37	2017.9.5 19:50
2017.11.27 18:47	2017.11.27	11	27	2	월요일	18	47	5	2017.11.27 18:47

연, 월, 일, 시 등 시간 구성 요소 분리

➢ 예약 테이블의 reserve_datetime에서 연, 월, 일, 시, 분, 초를 추려내보자

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

print(reserve_tb['reserve_datetime'].dt.year)

print(reserve_tb['reserve_datetime'].dt.month)

print(reserve_tb['reserve_datetime'].dt.day)

print(reserve_tb['reserve_datetime'].dt.dayofweek)

print(reserve_tb['reserve_datetime'].dt.hour)

print(reserve_tb['reserve_datetime'].dt.minute)

print(reserve_tb['reserve_datetime'].dt.second)

print(reserve_tb['reserve_datetime'].dt.strftime('%Y-%m-%d %H:%M:%S'))

print(reserve_tb)
```

datetime64 형으로 변경

년도 정보 추출

월 정보 추출

날짜 정보 추출

요일 정보 추출

시 정보 추출

분 정보 추출

초 정보 추출

지정 포맷 문자열로 변환

시간 간격 계산

- 일시형 데이터가 여러개 있을때 데이터 사이의 차이(연수, 월수, 주수, 일수, 시간차)를 구해보자

reserve_datetime	checkin_date	checkin_time
2016.3.6 13:09	2016.3.26	10:00:00
2016.7.16 23:39	2016.7.20	11:30:00
2016.9.24 10:03	2016.10.19	12:00:00
2017.3.8 2:01	2017.3.29	11:00:00
2017.9.5 19:50	2017.9.22	12:00:00
2017.11.27 18:47	2017.12.4	12:00:00

일시 차이 계산

reserve_datetime	checkin_datetime_ct	diff_year	diff_month	diff_day	diff_hour	diff_min	diff_sec
2016.3.6 13:09	2016.3.26 10:00	0	0	19.87625 days	476.835833 hr	28610.15 min	1716915 sec
2016.7.16 23:39	2016.7.20 11:30	0	0	3.493472 days	83.843322 hr	5030.6 min	301836 secs
2016.9.24 10:03	2016.10.19 12:00	0	1	24.91655 days	598.0 hr	35887.0 min	2153203 secs
2017.3.8 2:01	2017.3.29 11:00	0	0	21.37257 days	512.941833 hr	30776.51 min	1846591 secs
2017.9.5 19:50	2017.9.22 12:00	0	0	16.681805 days	400.363 hr	23981.83 min	1438903 secs
2017.11.27 18:47	2017.12.4 12:00	0	0	6.717703 days	161.262528 hr	9675.617 min	580375 secs

시간 간격 계산

- 일시형 데이터가 여러개 있을때 데이터 사이의 차이(연수, 월수, 주수, 일수, 시간차)를 구해보자

```
reserve_tb['reserve_datetime'] = \
  pd.to_datetime(reserve_tb['reserve_datetime'], format='%Y-%m-%d %H:%M:%S')
reserve_tb['checkin_datetime'] = \
  pd.to_datetime(reserve_tb['checkin_date'] + reserve_tb['checkin_time'],
                 format='%Y-%m-%d%H:%M:%S')
reserve_tb['reserve_datetime'].dt.year - \
reserve_tb['checkin_datetime'].dt.year

(reserve_tb['reserve_datetime'].dt.year * 12 +
 reserve_tb['reserve_datetime'].dt.month) \
 - (reserve_tb['checkin_datetime'].dt.year * 12 +
    reserve_tb['checkin_datetime'].dt.month)

(reserve_tb['reserve_datetime'] - reserve_tb['checkin_datetime']) \
    .dt.days
#  .astype('timedelta64[D]')
```

datetime64 형으로 변경

datetime64 형으로 변경

연도 차이 계산

월 차이 계산

일 차이 계산

시간 간격 계산

- 일시형 데이터가 여러개 있을때 데이터 사이의 차이(연수, 월수, 주수, 일수, 시간차)를 구해보자

```
(reserve_tb['reserve_datetime'] - reserve_tb['checkin_datetime']) \
    .dt.total_seconds() / 3600
#  .astype('timedelta64[h]')

(reserve_tb['reserve_datetime'] - reserve_tb['checkin_datetime']) \
    .dt.total_seconds() / 60
#  .astype('timedelta64[m]')

(reserve_tb['reserve_datetime'] - reserve_tb['checkin_datetime']) \
    .dt.total_seconds()
#  .astype('timedelta64[s]')
```

시 차이 계산

분 차이 계산

초 차이 계산

시간 정보의 증분 처리

- 예약 테이블의 예약 일시에 1일, 1시간, 1분, 1초를 추가해본다.

reserve_datetime	reserve_date
2016.3.6 13:09	2016.3.6
2016.7.16 23:39	2016.7.16
2016.9.24 10:03	2016.9.24
2017.3.8 2:01	2017.3.8
2017.9.5 19:50	2017.9.5
2017.11.27 18:47	2017.11.27

시간 증가

reserve_datetime	reserve_date	datetime_add_day	datetime_add_hou	datetime_add_min	datetime_add_sec	datetime_add_day (again)
2016.3.6 13:09	2016.3.6	2016.3.7 13:09	2016.3.6 14:09	2016.3.6 13:10	2016.3.6 13:09	2016.3.7
2016.7.16 23:39	2016.7.16	2016.7.17 23:39	2016.7.17 0:39	2016.7.16 23:40	2016.7.16 23:39	2016.7.17
2016.9.24 10:03	2016.9.24	2016.9.25 10:03	2016.9.24 11:03	2016.9.24 10:04	2016.9.24 10:03	2016.9.25
2017.3.8 2:01	2017.3.8	2017.3.9 2:01	2017.3.8 3:01	2017.3.8 2:02	2017.3.8 2:01	2017.3.9
2017.9.5 19:50	2017.9.5	2017.9.6 19:50	2017.9.5 20:50	2017.9.5 19:51	2017.9.5 19:50	2017.9.6
2017.11.27 18:47	2017.11.27	2017.11.28 18:47	2017.11.27 19:47	2017.11.27 18:48	2017.11.27 18:47	2017.11.28

시간 정보의 증분 처리

➢ 예약 테이블의 예약 일시에 1일, 1시간, 1분, 1초를 추가해본다.

```
import datetime

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

reserve_tb['reserve_datetime'] + datetime.timedelta(days=1)

reserve_tb['reserve_date'] + datetime.timedelta(days=1)

reserve_tb['reserve_datetime'] + datetime.timedelta(hours=1)

reserve_tb['reserve_datetime'] + datetime.timedelta(minutes=1)

reserve_tb['reserve_datetime'] + datetime.timedelta(seconds=1)
```

datetime64 형으로 변경

date 추출

1일 증가

1일 증가

1시간 증가

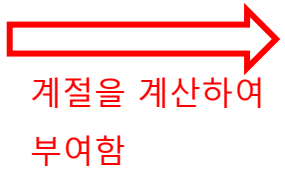
1분 증가

1초 증가

계절 항목으로 변환

➢ 예약 테이블의 reserve_datetime의 월부터 예약일의 계절 데이터를 생성해본다.

reserve_datetime
2016.3.6 13:09
2016.7.16 23:39
2016.9.24 10:03
2017.3.8 3:20
2017.9.5 19:50
2017.11.27 18:47



reserve_datetime	reserve_datetime_season
2016.3.6 13:09	spring
2016.7.16 23:39	summer
2016.9.24 10:03	autumn
2017.3.8 3:20	spring
2017.9.5 19:50	autumn
2017.11.27 18:47	autumn

계절로 변환

- 예약 테이블의 reserve_datetime의 월부터 예약일의 계절 데이터를 생성해본다.

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

def to_season(month_num):
  season = 'winter'
  if 3 <= month_num <= 5:
    season = 'spring'
  elif 6 <= month_num <= 8:
    season = 'summer'
  elif 9 <= month_num <= 11:
    season = 'autumn'

  return season

reserve_tb['reserve_season'] = pd.Categorical(
  reserve_tb['reserve_datetime'].dt.month.apply(to_season),
  categories=['spring', 'summer', 'autumn', 'winter']
)

print(reserve_tb['reserve_season'])
```

월의 숫자를 계절로 변환하는 함수

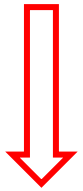
계절 변환 함수 호출

계절 범주 지정

평일/주말 및 공휴일 구분

- 예약 테이블의 checkin_date에 휴일 마스터(휴일 플래그, 휴일 전날 플래그)를 부여해보자

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	36000
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	13:00:00	2016.10.22	3	68100
r4	h_214	c_1	2017.3.8 2:01	2017.3.29	11:00:00	2017.3.30	4	194400
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	13:00:00	2017.9.23	2	36000
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	68100



휴일과 휴일 전날
플래그를 추가함

reserve_id	hotel_id	customer_id	reserve_datetime	checkin_date	checkin_time	checkout_date	people_num	total_price	holidayday	nextday_is_holiday
r1	h_75	c_1	2016.3.6 13:09	2016.3.26	10:00:00	2016.3.29	4	97200	TRUE	TRUE
r2	h_219	c_1	2016.7.16 23:39	2016.7.20	11:30:00	2016.7.21	2	36000	FALSE	FALSE
r3	h_179	c_1	2016.9.24 10:03	2016.10.19	13:00:00	2016.10.22	3	68100	FALSE	FALSE
r4	h_214	c_1	2017.3.8 2:01	2017.3.29	11:00:00	2017.3.30	4	194400	FALSE	FALSE
r5	h_16	c_1	2017.9.5 19:50	2017.9.22	13:00:00	2017.9.23	2	36000	FALSE	FALSE
r6	h_241	c_1	2017.11.27 18:47	2017.12.4	12:00:00	2017.12.6	3	68100	FALSE	FALSE

평일/주말 및 공휴일 구분

- 예약 테이블의 checkin_date에 휴일 마스터(휴일 플래그, 휴일 전날 플래그)를 부여해보자

```
holiday_mst = load_holiday_mst()

# 휴일 마스터와 결합
pd.merge(reserve_tb, holiday_mst,
        left_on='checkin_date', right_on='target_day')
```

NOTE> holiday_mst에 holiday_flag과 nextday_is_holiday_flg 필드가 존재함.

1. 수치 데이터 전처리 기법

2. 범주 데이터 전처리 기법

3. 일시 데이터 전처리 기법

4. 위치정보 데이터 전처리 기법

지역 기준 좌표계 변환

- 2010년 1월부터 공공측량 성과물에 세계 측지계 사용이 의무화되었다.
- 따라서 한국 측지계 데이터는 세계 측지계로 변환하여 사용하여야 한다.

customer_ic	age	sex	home_latitude	home_longitude
c_1	41	man	35.09219	136.5123
c_2	38	man	35.32508	139.4106
c_3	49	woman	35.12054	136.5112
c_4	43	man	43.03487	141.2403
c_5	31	man	35.10266	136.5238
c_6	52	man	34.44077	135.3905

한국 축지계를 세계 축지계로 변환함.

customer_ic	age	sex	home_latitude	home_longitude
c_1	41	man	35.15592	136.8536
c_2	38	man	35.56009	139.8616
c_3	49	woman	35.20473	136.8503
c_4	43	man	43.06595	141.9371
c_5	31	man	35.17728	136.8743
c_6	52	man	34.73871	135.6483

지역 기준 좌표계 변환

➢ 고객 테이블에서 집의 위도와 경도를 도 단위로 변경한 후 세계 측지계로 변환한다.

```
import pyproj

def convert_to_continuous(x):
  x_min = (x * 100 - int(x * 100)) * 100
  x_sec = (x - int(x) - x_min / 10000) * 100
  return int(x) + x_sec / 60 + x_min / 60 / 60

customer_tb['home_latitude'] = customer_tb['home_latitude'] \
  .apply(lambda x: convert_to_continuous(x))
customer_tb['home_longitude'] = customer_tb['home_longitude'] \
  .apply(lambda x: convert_to_continuous(x))

epsg_world = pyproj.Proj('+init=EPSG:4326')

epsg_korea = pyproj.Proj('+init=EPSG:4301')#4162

home_position = customer_tb[['home_longitude', 'home_latitude']] \
  .apply(lambda x:
        pyproj.transform(epsg_korea, epsg_world, x[0], x[1]), axis=1)

customer_tb['home_longitude'] = [x[0] for x in home_position]
customer_tb['home_latitude'] = [x[1] for x in home_position]
```

분 초를 도로 변환하는 함수

주의> GIS 관련 pyproj 를 설치하여야 한다.
pip install pyproj

분 초를 도로 변환

세계 축지계 구함

한국 축지계 구함

한국 축지계를 세계 축지계로 변환

customer_tb를 세계 축지계로 갱신

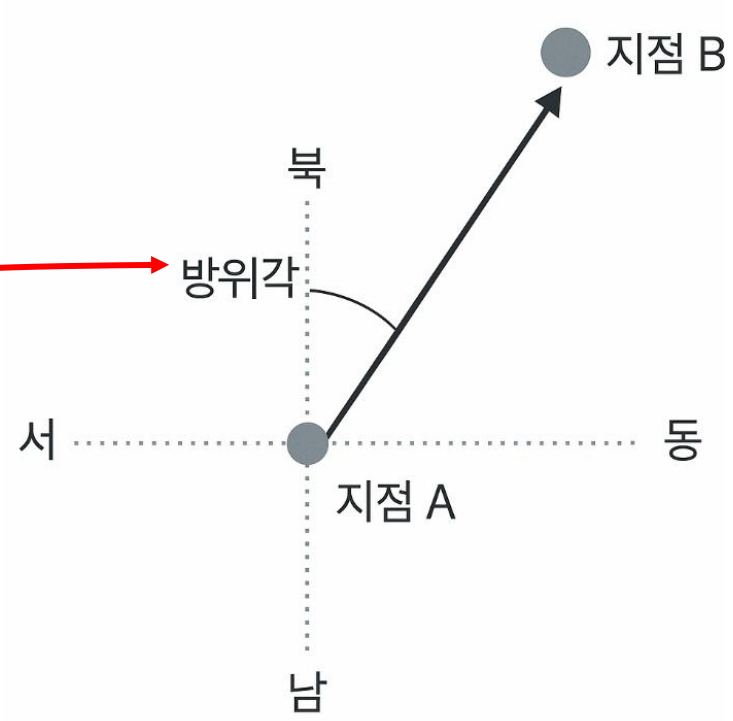
두 지점 간 거리 및 방향 계산

- 위도와 경도를 이용하여 두 지점 간의 거리와 방향을 계산한다.
- 2000km 이내일때 휴베니 방식, 그 이상에서 정확도를 요구할때 빈센티 또는 하버사인 방식을 사용한다.

reserve_id	home_latit	home_longitu	hotel_latitude	hotel_longitude
r1	35.15592	136.8536	35.54568	139.7012
r2	35.15592	136.8536	35.64473	139.6701
r3	35.15592	136.8536	35.59996	139.3204
r4	35.15592	136.8536	35.84793	138.2431
r5	35.15592	136.8536	35.91119	139.0913
r6	35.15592	136.8536	35.81541	139.839

위도와 경도에서
거리와 방위각을 계산함

reserve_id	dist_havers	dist_vincenty	dist_hubeny	azimuth
r1	262093.5	262093.5	262839.9	79.77027
r2	263272.9	263272.9	263872.5	77.90472
r3	597245.5	597245.5	597984.8	-105.0404
r4	437981.9	437981.9	438752.3	43.87947
r5	291191.6	291191.6	291508.3	72.47102
r6	280277.3	280277.3	280586.6	74.09673



두 지점 간 거리 및 방향 계산

주1> GIS 관련 geopy 를 설치하여야 한다.
pip install geopy

➢ 위도와 경도를 이용하여 두 지점 간의 거리와 방향을 계산한다.

```
def convert_to_continuous(x):
    x_min = (x * 100 - int(x * 100)) * 100
    x_sec = (x - int(x) - x_min / 10000) * 100
    return int(x) + x_sec / 60 + x_min / 60 / 60

customer_tb['home_latitude'] = customer_tb['home_latitude'] \
  .apply(lambda x: convert_to_continuous(x))
customer_tb['home_longitude'] = customer_tb['home_longitude'] \
  .apply(lambda x: convert_to_continuous(x))

epsg_world = pyproj.Proj('+init=EPSG:4326')

epsg_korea = pyproj.Proj('+init=EPSG:4301')

home_position = customer_tb[['home_longitude', 'home_latitude']] \
  .apply(lambda x:
        pyproj.transform(epsg_korea, epsg_world, x[0], x[1]), axis=1)

customer_tb['home_longitude'] = [x[0] for x in home_position]
customer_tb['home_latitude'] = [x[1] for x in home_position]
```

분 초를 도로 변환하는 함수

분 초를 도로 변환

세계 축지계 구함

한국 축지계 구함

한국 축지계를 세계 축지계로 변환

customer_tb를 세계 축지계로 갱신

두 지점 간 거리 및 방향 계산

➢ 위도와 경도를 이용하여 두 지점 간의 거리와 방향을 계산한다.

```
import math
import pyproj

#from geopy.distance import great_circle, vincenty
from geopy.distance import great_circle

reserve_tb = \
  pd.merge(reserve_tb, customer_tb, on='customer_id', how='inner')
reserve_tb = pd.merge(reserve_tb, hotel_tb, on='hotel_id', how='inner')

home_and_hotel_points = reserve_tb \
  .loc[:, ['home_longitude', 'home_latitude',
           'hotel_longitude', 'hotel_latitude']]

g = pyproj.Geod(ellps='WGS84')

home_to_hotel = home_and_hotel_points \
  .apply(lambda x: g.inv(x[0], x[1], x[2], x[3]), axis=1)
```

거리 계산을 위한 라이브러리 로드

예약 테이블에 고객/호텔 테이블 결합

집과 호텔의 위도/경도 정보 획득

적도 반경을 WGS84 기준 설정

Vincenty거리 계산

두 지점 간 거리 및 방향 계산

- 위도와 경도를 이용하여 두 지점 간의 거리와 방향을 계산한다.

```
[x[0] for x in home_to_hotel]

[x[2] for x in home_to_hotel]

home_and_hotel_points.apply(
  lambda x: great_circle((x[1], x[0]), (x[3], x[2])).meters, axis=1)

#home_and_hotel_points.apply(
#  lambda x: vincenty((x[1], x[0]), (x[3], x[2])).meters, axis=1)


def hubeny(lon1, lat1, lon2, lat2, a=6378137, b=6356752.314245):
    e2 = (a ** 2 - b ** 2) / a ** 2
    (lon1, lat1, lon2, lat2) = \
      [x * (2 * math.pi) / 360 for x in (lon1, lat1, lon2, lat2)]
    w = 1 - e2 * math.sin((lat1 + lat2) / 2) ** 2
    c2 = math.cos((lat1 + lat2) / 2) ** 2
    return math.sqrt((b ** 2 / w ** 3) * (lat1 - lat2) ** 2 +
                     (a ** 2 / w) * c2 * (lon1 - lon2) ** 2)

home_and_hotel_points \
  .apply(lambda x: hubeny(x[0], x[1], x[2], x[3]), axis=1)
```

Haversine거리 계산

hubeny 거리 계산 함수

hubeny거리 계산