In [39]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

> (1) 데이터를 탐색하고 탐색결과를 제시하시오.

In [41]:
```
df = pd.read_csv('hotel_bookings.csv')
```

In [42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 11 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   is_canceled                     20000 non-null  int64  
 1   deposit_type                    20000 non-null  object 
 2   lead_time                       19995 non-null  float64
 3   stays_in_weekend_nights         20000 non-null  int64  
 4   stays_in_week_nights            20000 non-null  int64  
 5   is_repeated_guest               19642 non-null  float64
 6   previous_cancellations          20000 non-null  int64  
 7   previous_bookings_not_canceled  20000 non-null  int64  
 8   booking_changes                 20000 non-null  int64  
 9   days_in_waiting_list            20000 non-null  int64  
 10  adr                             18937 non-null  float64
dtypes: float64(3), int64(7), object(1)
memory usage: 1.7+ MB
```

In [43]:
```
df
```

Out[43]:

	is_canceled	deposit_type	lead_time	stays_in_weekend_nights	stays_in_week_nights	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	days_in_waiting_list	adr
0	0	No Deposit	105.0	2	5	NaN	0	0	1	0	131.50
1	0	No Deposit	303.0	2	2	NaN	0	0	0	0	73.95
2	0	No Deposit	33.0	2	3	0.0	0	0	0	0	NaN
3	0	No Deposit	48.0	0	1	0.0	0	0	1	0	80.30
4	0	No Deposit	216.0	4	7	0.0	0	0	2	0	60.90
...	...	...	...	...	...	...	...	...	...	...	...
19995	1	Non Refund	89.0	2	2	0.0	0	0	0	0	62.00
19996	1	Non Refund	101.0	0	3	0.0	0	0	0	0	130.00
19997	1	Non Refund	277.0	1	2	0.0	0	0	0	0	100.00
19998	1	No Deposit	0.0	0	1	0.0	0	0	0	0	209.00
19999	1	Non Refund	40.0	0	2	0.0	0	0	0	0	130.00

20000 rows × 11 columns

In [44]:
```
df.describe(include='all')
```

Out[44]:

	is_canceled	deposit_type	lead_time	stays_in_weekend_nights	stays_in_week_nights	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	days_in_waiting_list	adr
count	20000.00000	20000	19995.000000	20000.000000	20000.000000	19642.000000	20000.000000	20000.000000	20000.000000	20000.000000	18937.000000
unique	NaN	3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	No Deposit	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	19138	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	0.12000	NaN	85.978345	0.892550	2.380400	0.038133	0.032900	0.169050	0.269400	1.983950	101.410239
std	0.32497	NaN	96.427240	0.952077	1.777345	0.191521	0.455552	1.502426	0.687566	15.927212	49.245097
min	0.00000	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-6.380000
25%	0.00000	NaN	11.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	68.800000
50%	0.00000	NaN	51.000000	1.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	94.500000
75%	0.00000	NaN	132.000000	2.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	126.000000
max	1.00000	NaN	629.000000	13.000000	30.000000	1.000000	26.000000	66.000000	17.000000	379.000000	451.500000

In [45]:
```
df.columns
```

Out[45]:
```
Index(['is_canceled', 'deposit_type', 'lead_time', 'stays_in_weekend_nights',
       'stays_in_week_nights', 'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'booking_changes',
       'days_in_waiting_list', 'adr'],
      dtype='object')
```

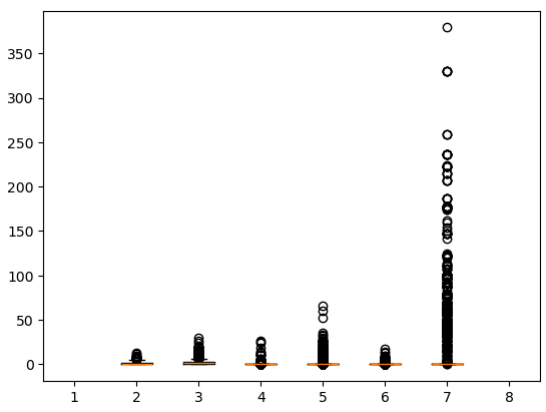
In [46]:
```
df['is_canceled'].value_counts()
```

Out[46]:
```
is_canceled
0    17600
1     2400
Name: count, dtype: int64
```

In [47]:
```
df['is_repeated_guest'].value_counts()
```

Out[47]:
```
is_repeated_guest
0.0    18893
1.0      749
Name: count, dtype: int64
```

In [48]:
```
plt.boxplot(df[['lead_time', 'stays_in_weekend_nights', 'stays_in_week_nights', 'previous_cancellations', 'previous_bookings_not_canceled', 'booking_changes', 'days_in_waiting_list', 'adr' ]])
plt.show()
```



> 총 20000개의 데이터개수를 가지고 있으며, 'is_canceled', 'lead_time', 'stays_in_weekend_nights', ''stays_in_week_nights', 'is_repeated_guest', 'previous_cancellations', 'previous_bookings_not_canceled', 'booking_changes ', 'days_in_waiting_list', 'adr' 등 10개의 수치형 변수와 'deposit_type' 등 1개의 범주형 변수를 갖는다. 수치형 변수 중 'is_canceled', 'is_repeated_guest'의 경우 0 또는 1 의 값을 가짐을 알 수 있다.

> 데이터는 객실 사용 여부에 대한 것이므로 'is_canceled'가 target 변수로 판단되고, 이 중 취소된 숫자는 20000 개중 2400 개로 그 비율이 적다. Boxplot 시각화에서 따르면 수치형 변수 모두는 이상치를 갖고 있다.

> (2) 결측치를 탐색하고 대체방법 및 근거를 제시하시오

In [51]:
```
df.isna().sum()
```

Out[51]:
```
is_canceled                         0
deposit_type                        0
lead_time                           5
stays_in_weekend_nights             0
stays_in_week_nights                0
is_repeated_guest                 358
previous_cancellations              0
previous_bookings_not_canceled      0
booking_changes                     0
days_in_waiting_list                0
adr                              1063
dtype: int64
```

In [52]:
```
df[df['lead_time'].isna()]
```

Out[52]:

	is_canceled	deposit_type	lead_time	stays_in_weekend_nights	stays_in_week_nights	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	days_in_waiting_list	adr
985	0	No Deposit	NaN	0	3	0.0	0	0	2	0	24.00
1087	0	No Deposit	NaN	2	6	0.0	0	0	1	0	99.68
4125	0	No Deposit	NaN	0	2	0.0	0	0	0	0	46.00
4923	0	No Deposit	NaN	1	4	0.0	0	0	0	0	129.60
16221	0	No Deposit	NaN	0	1	0.0	0	0	1	0	199.00

In [53]:
```
df['lead_time'].describe()
```

Out[53]:
```
count    19995.000000
mean        85.978345
std         96.427240
min          0.000000
25%         11.000000
50%         51.000000
75%        132.000000
max        629.000000
Name: lead_time, dtype: float64
```

In [54]:
```
df[df['is_repeated_guest'].isna()].head()
```

Out[54]:

	is_canceled	deposit_type	lead_time	stays_in_weekend_nights	stays_in_week_nights	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	days_in_waiting_list	adr
0	0	No Deposit	105.0	2	5	NaN	0	0	1	0	131.50
1	0	No Deposit	303.0	2	2	NaN	0	0	0	0	73.95
7	0	No Deposit	219.0	1	2	NaN	0	0	0	0	76.67
37	0	No Deposit	57.0	1	3	NaN	0	0	0	0	85.85
38	0	No Deposit	116.0	0	4	NaN	0	0	0	0	56.10

In [55]:
```
df['is_repeated_guest'].describe()
```

Out[55]:
```
count    19642.000000
mean         0.038133
std          0.191521
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: is_repeated_guest, dtype: float64
```

In [56]:
```
df[df['adr'].isna()].head()
```

Out[56]:

	is_canceled	deposit_type	lead_time	stays_in_weekend_nights	stays_in_week_nights	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	days_in_waiting_list	adr
2	0	No Deposit	33.0	2	3	0.0	0	0	0	0	NaN
11	0	No Deposit	77.0	2	2	0.0	0	0	0	0	NaN
14	0	No Deposit	377.0	0	2	0.0	0	0	1	0	NaN
26	0	No Deposit	0.0	0	1	0.0	0	0	0	0	NaN
31	0	No Deposit	67.0	2	4	0.0	0	0	1	0	NaN

In [57]:
```
df['adr'].describe()
```

Out[57]:
```
count    18937.000000
mean       101.410239
std         49.245097
min         -6.380000
25%         68.800000
50%         94.500000
75%        126.000000
max        451.500000
Name: adr, dtype: float64
```

> 변수 'lead_time', 'is_repeated_guest', 'adr' 가 각각 결측치를 가지며, 'lead_time'와 'adr' 의 경우 describe 분석 상에서 범위가 크기 때문에 평균값 대체를 하고,
>
> 'is_repeated_guset' 변수의 경우 0또는 1의 값을 가지나 평균값이 0.038로 0에 가깝기 때문에 최빈값인 0으로 대체한다.

In [59]:
```
df['lead_time'] = df['lead_time'].fillna(df['lead_time'].mean())
df['lead_time'].isna().sum()
```

Out[59]:
```
0
```

In [60]:
```
df['is_repeated_guest'] = df['is_repeated_guest'].fillna(0)
df['is_repeated_guest'].isna().sum()
```

Out[60]:
```
0
```

In [61]:
```
df['adr'] = df['adr'].fillna(df['adr'].mean())
df['adr'].isna().sum()
```

Out[61]:
```
0
```

> (3) 데이터 질을 향상시킬 수 있는 방법을 제시하시오.
>
> 데이터의 질을 향상시키기 위해서는 데이터 클리닝으로서 결측치를 처리하거나 이상치에 대한 확인을 통해 대체하는 것과, 데이터 변환으로서 추후 분석방법을 고려하여 스케일링 또는 요약하는 것 그리고 변수를 축소하거나 라벨링을 통해 데이터 축소를 하는 등의 방법이 있고, 또한 클래스 불균형이 있을 경우 오버샘플링 또는 언더샘플링을 통하여 이를 조정할 수 있다.
>
> 해당 데이터에서는 이상치가 있으나 클래스 불균형이 있어 이상치에 의미가 있을 수 있으므로 이상치 대체를 보류한다. 우선 아래처럼 범주형 변수 'deposit_type'을 라벨링 한다. 변수 내 종류가 3가지로 많지 않으므로 원핫 인코딩을 선택한다.

In [63]:
```
df['deposit_type'].value_counts()
```

Out[63]:
```
deposit_type
No Deposit    19138
Non Refund      834
Refundable       28
Name: count, dtype: int64
```

In [64]:
```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
result = pd.DataFrame(ohe.fit_transform(df[['deposit_type']]).toarray())
print(result.head())
```
```
     0    1    2
0  1.0  0.0  0.0
1  1.0  0.0  0.0
2  1.0  0.0  0.0
3  1.0  0.0  0.0
4  1.0  0.0  0.0
```

In [65]:
```
result.value_counts()
```

Out[65]:
```
0    1    2
1.0  0.0  0.0    19138
0.0  1.0  0.0      834
     0.0  1.0       28
Name: count, dtype: int64
```

In [66]:
```
df2 = df.copy()
df['deposit_type_NoD'] = result[0]
df['deposit_type_NonR'] = result[1]
df['deposit_type_R'] = result[2]

df = df.drop(columns='deposit_type')
```

In [90]:
```
df.head()
```

Out[90]:

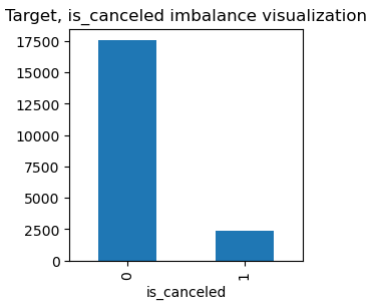
	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_nights	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	days_in_waiting_list	adr	deposit_type_NoD	deposit_type_NonR	deposit_type_R
0	0	105.0	2	5	0.0	0	0	1	0	131.500000	1.0	0.0	0.0
1	0	303.0	2	2	0.0	0	0	0	0	73.950000	1.0	0.0	0.0
2	0	33.0	2	3	0.0	0	0	0	0	101.410239	1.0	0.0	0.0
3	0	48.0	0	1	0.0	0	0	1	0	80.300000	1.0	0.0	0.0
4	0	216.0	4	7	0.0	0	0	2	0	60.900000	1.0	0.0	0.0

> (4) 데이터 불균형을 시각화하여 식별하고 불균형 판단근거를 작성하시오.

In [92]:
```
df['is_canceled'].value_counts()
```

Out[92]:
```
is_canceled
0    17600
1     2400
Name: count, dtype: int64
```

In [98]:
```
fig, ax = plt.subplots(figsize=(3,3))

ax.set_title('Target, is_canceled imbalance visualization')
df['is_canceled'].value_counts().plot.bar(ax=ax)
plt.show()
```



In [96]:
```
np.divide(2400,20000)
```

Out[96]:
```
0.12
```

Target 변수 'is_canceled' 에서 0은 17600 으로 전체의 88% 를 차지하며, 1은 2400 으로 전체의 12% 에 불과하여 이는 분류 문제에서 클래스 불균형으로 판단된다.

> (5) 오버 샘플링 기법을 설명하고 비교한 뒤 2개 기법을 선정하고 근거를 제시하시오.

> 데이터 클래스가 불균형한 경우 데이터 수가 적은 클래스의 분포를 적절히 학습하지 못하여 다수 클래스에 과대적합되어 대부분의 데이터를 다수 클래슬 분류하는 문제가 발생한다. 이 때문에 오버 샘플링 또는 언더샘플링을 사용하는데, 오버 샘플링이란 소수의 데이터를 다수 클래스의 데이터 만큼 붕식시켜 학습에 사용하기 위한 충분한 양과 비율의 데이터를 확보하는 기법이다. 데이터의 손실이 없어 보통 언더 샘플링보다 성능이 우리하여 주로 사용된다.

> 오버샘플링 기법에는 Random Over sampling 과 SMOTE, SVMSOMTE, ADASYN 등이 있는데, 이 중 학습이 빠른 Random Over Sampling 과 SMOTE 을 고려하되, 랜덤오버샘플링은 소수의레이블을 지닌 데이터세트를 단순복제하여 다수의 레이블과 비율을 맞추는 기법으로 오버피팅의 위험성이 있다. SMOTE 는 소수의 레이블을 지닌 데이터세트의 관측 값에 대한 K개의 최근접 이웃(K-nearest Neighbor)을 찾고 관측값과 이웃으로 선택된 값 사이에 임의의 새로운 데이터를 생성하는 방법으로 샘플의 수를 늘리는 방법이다.

In [116…
```
print(df.columns)
x = df[['lead_time', 'stays_in_weekend_nights', 'stays_in_week_nights', 'is_repeated_guest', 'previous_cancellations',
        'previous_bookings_not_canceled', 'booking_changes',
       'days_in_waiting_list', 'adr', 'deposit_type_NoD', 'deposit_type_NonR',
       'deposit_type_R']]
y = df[['is_canceled']]

print(y.head())
```

```
Index(['is_canceled', 'lead_time', 'stays_in_weekend_nights',
       'stays_in_week_nights', 'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'booking_changes',
       'days_in_waiting_list', 'adr', 'deposit_type_NoD', 'deposit_type_NonR',
       'deposit_type_R'],
      dtype='object')
   is_canceled
0            0
1            0
2            0
3            0
4            0
```

In [131…
```
from imblearn.over_sampling import RandomOverSampler
oversample = RandomOverSampler(sampling_strategy = 'minority')

x_over, y_over = oversample.fit_resample(x, y)
print('imbalanced data의 class 분포: %s' % len(x))
print('oversampled data의 class 분포: %s' % len(x_over))
```

```
imbalanced data의 class 분포: 20000
oversampled data의 class 분포: 35200
```

In [133…
```
y_over.value_counts()
```

Out[133…
```
is_canceled
0              17600
1              17600
Name: count, dtype: int64
```

In [135…
```
from imblearn.over_sampling import SMOTE
oversample = SMOTE(sampling_strategy = 'minority')

x_sm, y_sm = oversample.fit_resample(x, y)
print('imbalanced data의 class 분포: %s' % len(x))
print('oversampled data의 class 분포: %s' % len(x_sm))

print(y_sm.value_counts())
```

```
imbalanced data의 class 분포: 20000
oversampled data의 class 분포: 35200
is_canceled
0              17600
1              17600
Name: count, dtype: int64
```

> (6) 기법을 선정한 이유를 작성하고, 원 데이터를 포함해 3개의 데이터 세트를 구성하시오.

> Random Over Sampling 보다 새로운 데이터의 분포가 다양하기 때문에 과적합의 위험이 적어 SMOTE 방법을 선택하기로 한다.

In [141…
```
# 원데이터 세트 x, y
# 랜덤오버샘플링 데이터 세트 x_over, y_over
# SMOTE 오버샘플링 데이터 세트 x_sm, y_sm
```

(7) 오버샘플링 데이터와 원데이터를 사용해 정확도 측면 모델 하나와 속도 측면의 모델 하나를 선정하고 그 이유를 설명하시오.

정확도 분석을 위해 로지스틱 회귀모델을 사용하며, 속도측면에서 의사결정나무를 사용한다. 의사결정나무 사용 시 스케일링은 필요없다.

In [146…
```
# 로지스틱 회귀분석에 앞서 Scaling 을 시행한다.

from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()

mm_x = mm.fit_transform(x)
mm_x = pd.DataFrame(mm_x, columns = x.columns)

mm_x.describe()
```

Out[146…

	lead_time	stays_in_weekend_nights	stays_in_week_nights	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	days_in_waiting_list	adr	deposit_type_NoD	deposit_type_NonR	deposit_type_R
count	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000
mean	0.136691	0.068658	0.079347	0.037450	0.001265	0.002561	0.015847	0.005235	0.235412	0.956900	0.041700	0.001400
std	0.153283	0.073237	0.059245	0.189867	0.017521	0.022764	0.040445	0.042024	0.104653	0.203087	0.199908	0.037391
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.017488	0.000000	0.033333	0.000000	0.000000	0.000000	0.000000	0.000000	0.168996	1.000000	0.000000	0.000000
50%	0.081081	0.076923	0.066667	0.000000	0.000000	0.000000	0.000000	0.000000	0.225234	1.000000	0.000000	0.000000
75%	0.209857	0.153846	0.100000	0.000000	0.000000	0.000000	0.000000	0.000000	0.283710	1.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [148…
```
mm_x_over = mm.fit_transform(x_over)
mm_x_over = pd.DataFrame(mm_x_over, columns = x.columns)

mm_x_sm = mm.fit_transform(x_sm)
mm_x_sm = pd.DataFrame(mm_x_sm, columns = x.columns)
```

In [204…
```
from sklearn.model_selection import train_test_split

train_x, test_x, train_y, test_y = train_test_split(mm_x, y, train_size = 0.7)
print(train_x.shape, test_x.shape, train_y.shape, test_y.shape)
```

```
(14000, 12) (6000, 12) (14000, 1) (6000, 1)
```

In [208…
```
train_y = train_y.values
train_y
```

Out[208…
```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [1],
       [0]], dtype=int64)
```

In [210…
```
from sklearn.linear_model import LogisticRegression
logR = LogisticRegression()


logR.fit(train_x, train_y)
```

```
C:\Users\minje\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[210…
```
LogisticRegression()
```

In [214…
```
proba = pd.DataFrame(logR.predict_proba(train_x))
cs = logR.decision_function(train_x)

df = pd.concat([proba, pd.DataFrame(cs)], axis=1)
df.columns = ['Not A', 'A', 'decision_function']
```

```
df.sort_values(['decision_function'], inplace=True)
df.reset_index(inplace=True, drop=True)

df
```

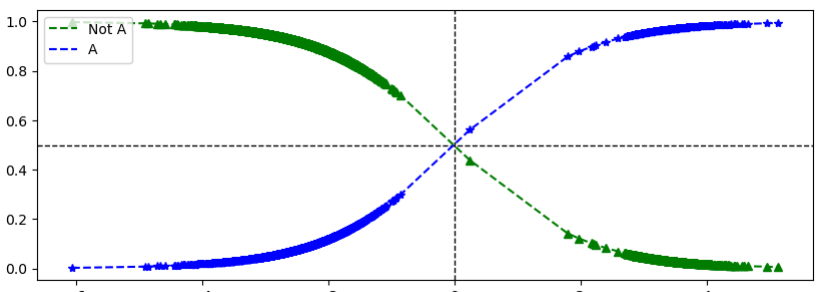
Out[214…

	Not A	A	decision_function
0	0.997674	0.002326	-6.061097
1	0.992583	0.007417	-4.896504
2	0.992266	0.007734	-4.854374
3	0.991117	0.008883	-4.714677
4	0.990654	0.009346	-4.663365
…	…	…	…
13995	0.010066	0.989934	4.588427
13996	0.009948	0.990052	4.600359
13997	0.009363	0.990637	4.661567
13998	0.006994	0.993006	4.955630
13999	0.005891	0.994109	5.128419

14000 rows × 3 columns

In [220…

```
plt.figure(figsize=(10,3.5))

plt.axhline(y=0.5, linestyle='--', color='black', linewidth=1)
plt.axvline(x=0, linestyle='--', color='black', linewidth=1)

plt.plot(df['decision_function'], df['Not A'], 'g--', label='Not A')
plt.plot(df['decision_function'], df['Not A'], 'g^')
plt.plot(df['decision_function'], df['A'], 'b--', label='A')
plt.plot(df['decision_function'], df['A'], 'b*')

plt.xlabel
plt.ylabel

plt.legend(loc='upper left')

plt.show()
```



In [222…

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

pred = logR.predict(test_x)

test_cm = confusion_matrix(test_y, pred)
test_acc = accuracy_score(test_y, pred)
test_prc = precision_score(test_y, pred)
test_rcll = recall_score(test_y, pred)
test_f1 = f1_score(test_y, pred)

print(test_cm)
print('\n')
print('정확도\t{}%'.format(round(test_acc*100, 2)))
print('정밀도\t{}%'.format(round(test_prc*100, 2)))
print('재현율\t{}%'.format(round(test_rcll*100, 2)))
print('F1\t{}%'.format(round(test_f1*100, 2)))
```

```
[[5266    3]
 [ 500  231]]


정확도	91.62%
정밀도	98.72%
재현율	31.6%
F1	47.88%
```

> 원 데이터를 로지스틱 회귀분석한 결과의 정확도 등 성능은 위와 같다.

> 아래는 오버샘플링한 데이터를 이용해 로지스틱 회귀분석한 결과이다.

In [231…

```
# mm_x_over, y_over
# mm_x_sm, y_sm

train_x, test_x, train_y, test_y = train_test_split(mm_x_over, y_over, train_size = 0.7)
print(train_x.shape, test_x.shape, train_y.shape, test_y.shape)

logR.fit(train_x, train_y)

pred = logR.predict(test_x)

test_cm = confusion_matrix(test_y, pred)
test_acc = accuracy_score(test_y, pred)
test_prc = precision_score(test_y, pred)
test_rcll = recall_score(test_y, pred)
test_f1 = f1_score(test_y, pred)

print(test_cm)
print('\n')
print('정확도\t{}%'.format(round(test_acc*100, 2)))
print('정밀도\t{}%'.format(round(test_prc*100, 2)))
print('재현율\t{}%'.format(round(test_rcll*100, 2)))
print('F1\t{}%'.format(round(test_f1*100, 2)))
```

```
(24640, 12) (10560, 12) (24640, 1) (10560, 1)
[[4602  619]
 [2686 2653]]


정확도	68.7%
정밀도	81.08%
재현율	49.69%
F1	61.62%
C:\Users\minje\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [235…

```
train_x, test_x, train_y, test_y = train_test_split(mm_x_sm, y_sm, train_size = 0.7)
print(train_x.shape, test_x.shape, train_y.shape, test_y.shape)

logR.fit(train_x, train_y)

pred = logR.predict(test_x)

test_cm = confusion_matrix(test_y, pred)
test_acc = accuracy_score(test_y, pred)
test_prc = precision_score(test_y, pred)
test_rcll = recall_score(test_y, pred)
test_f1 = f1_score(test_y, pred)

print(test_cm)
print('\n')
print('정확도\t{}%'.format(round(test_acc*100, 2)))
print('정밀도\t{}%'.format(round(test_prc*100, 2)))
print('재현율\t{}%'.format(round(test_rcll*100, 2)))
print('F1\t{}%'.format(round(test_f1*100, 2)))
```

```
(24640, 12) (10560, 12) (24640, 1) (10560, 1)
[[4758  489]
 [2398 2915]]


정확도	72.66%
정밀도	85.63%
재현율	54.87%
F1	66.88%
C:\Users\minje\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

> 아래는 속도 측면의 모델 랜덤포레스트이다.

In [243…

```
train_x, test_x, train_y, test_y = train_test_split(x, y, train_size = 0.7)

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=5)
clf.fit(train_x, train_y)

pred = clf.predict(test_x)
```

```
test_cm = confusion_matrix(test_y, pred)
test_acc = accuracy_score(test_y, pred)
test_prc = precision_score(test_y, pred)
test_rcll = recall_score(test_y, pred)
test_f1 = f1_score(test_y, pred)

print(test_cm)
print('\n')
print('정확도\t{}%'.format(round(test_acc*100, 2)))
print('정밀도\t{}%'.format(round(test_prc*100, 2)))
print('재현율\t{}%'.format(round(test_rcll*100, 2)))
print('F1\t{}%'.format(round(test_f1*100, 2)))

from sklearn.metrics import classification_report
report = classification_report(test_y, pred)
print(report)
```

```
[[5315    7]
 [ 436  242]]


정확도  92.62%
정밀도  97.19%
재현율  35.69%
F1      52.21%
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      5322
           1       0.97      0.36      0.52       678

    accuracy                           0.93      6000
   macro avg       0.95      0.68      0.74      6000
weighted avg       0.93      0.93      0.91      6000
```

> 랜덤오버샘플링 후 의사결정나무

In [253…

```
train_x, test_x, train_y, test_y = train_test_split(x_over, y_over, train_size = 0.7)

clf = DecisionTreeClassifier(max_depth=5)
clf.fit(train_x, train_y)

pred = clf.predict(test_x)

test_cm = confusion_matrix(test_y, pred)
test_acc = accuracy_score(test_y, pred)
test_prc = precision_score(test_y, pred)
test_rcll = recall_score(test_y, pred)
test_f1 = f1_score(test_y, pred)

print(test_cm)
print('\n')
print('정확도\t{}%'.format(round(test_acc*100, 2)))
print('정밀도\t{}%'.format(round(test_prc*100, 2)))
print('재현율\t{}%'.format(round(test_rcll*100, 2)))
print('F1\t{}%'.format(round(test_f1*100, 2)))

report = classification_report(test_y, pred)
print(report)
```

```
[[3632 1633]
 [1484 3811]]


정확도  70.48%
정밀도  70.0%
재현율  71.97%
F1      70.97%
              precision    recall  f1-score   support

           0       0.71      0.69      0.70      5265
           1       0.70      0.72      0.71      5295

    accuracy                           0.70     10560
   macro avg       0.70      0.70      0.70     10560
weighted avg       0.70      0.70      0.70     10560
```

> SMOTE 후 의사결정나무

In [257…

```
train_x, test_x, train_y, test_y = train_test_split(x_sm, y_sm, train_size = 0.7)

clf = DecisionTreeClassifier(max_depth=5)
clf.fit(train_x, train_y)

pred = clf.predict(test_x)

test_cm = confusion_matrix(test_y, pred)
test_acc = accuracy_score(test_y, pred)
test_prc = precision_score(test_y, pred)
test_rcll = recall_score(test_y, pred)
test_f1 = f1_score(test_y, pred)

print(test_cm)
print('\n')
print('정확도\t{}%'.format(round(test_acc*100, 2)))
print('정밀도\t{}%'.format(round(test_prc*100, 2)))
print('재현율\t{}%'.format(round(test_rcll*100, 2)))
print('F1\t{}%'.format(round(test_f1*100, 2)))

report = classification_report(test_y, pred)
print(report)
```

```
[[3406 1866]
 [ 963 4325]]


정확도  73.21%
정밀도  69.86%
재현율  81.79%
F1      75.35%
              precision    recall  f1-score   support

           0       0.78      0.65      0.71      5272
           1       0.70      0.82      0.75      5288

    accuracy                           0.73     10560
   macro avg       0.74      0.73      0.73     10560
weighted avg       0.74      0.73      0.73     10560
```

> 로지스틱 회귀, 원데이터 - 랜덤오버샘플링 - SMOTE 순서

정확도 91.62% 정밀도 98.72% 재현율 31.6% F1 47.88%

정확도 68.7% 정밀도 81.08% 재현율 49.69% F1 61.62%

정확도 72.66% 정밀도 85.63% 재현율 54.87% F1 66.88%

> 의사결정나무 회귀, 원데이터 - 랜덤오버샘플링 - SMOTE 순서

정확도 92.62% 정밀도 97.19% 재현율 35.69% F1 52.21%

정확도 70.48% 정밀도 70.0% 재현율 71.97% F1 70.97%

정확도 73.21% 정밀도 69.86% 재현율 81.79% F1 75.35%

로지스틱회귀와 의사결정나무를 비교할 때 정확도, 재현율, F1 score 는 데이터에 상관없이 의사결정 나무에서 더 높았고, 정밀도는 로지스틱 회귀에서 더 높다.

두 모델 모두에서 정확도와 정밀도는 원데이터가 가장 높고 랜덤오버샘플링에서보단 SMOTE 에서 약간 더 높다. 그러나 재현율과 F1 score는 원데이터보다 랜덤오버샘플링에서, 랜덤오버샘플링에서보다 SMOTE 에서 더 높다.