In [3]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import mglearn
%matplotlib inline
import seaborn as sns
import platform
from matplotlib import font_manager , rc

if platform.system() == 'Darwin':
  rc('font' , family = 'AppleGothic')
elif platform.system() == 'Windows':
  path = 'C:/Windows/Fonts/malgun.ttf'
  font_name = font_manager.FontProperties(fname = path).get_name()
  rc('font' , family = font_name)
else:
  print('모름')
plt.rcParams['axes.unicode_minus'] = False
import warnings
warnings.filterwarnings('ignore')
```
executed in 3.31s, finished 15:32:17 2023-11-10

In [4]:
```
#필요한 모델들
from sklearn.ensemble import RandomForestRegressor , RandomForestClassifier
from sklearn.linear_model import LinearRegression , Ridge , LogisticRegression
from sklearn.model_selection import GridSearchCV , train_test_split , cross_val
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures as PF
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import precision_score , classification_report , confusion
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
```
executed in 1.12s, finished 15:32:18 2023-11-10

1 문제 정의

- 수질 지표의 지수에 따라 이 물이 먹어도 되는지 안되는지에 대해 판단하기

1.1 데이터셋 로딩

In [5]:
```
data = pd.read_csv('water_potability.csv')
```
executed in 13ms, finished 15:32:18 2023-11-10

1.2 데이터 탐색

- pH: 물의 pH 수준.
- Hardness: 물의 경도, 미네랄 함량의 척도
- Solids: 물에 용해된 총 고형물

- Chloramines: 물 속의 클로라민 농도.
- Sulfate: 물 속의 황산염 농도.
- Conductivity: 물의 전기 전도도.
- Organic_carbon: 물 속 유기탄소 함량.
- Trihalomethanes: 물 속의 트리할로메탄 농도.
- Turbidity: 탁도 수준, 물의 투명도를 나타내는 척도.
- Potability: target(0 : 불가 , 1: 가능)

In [6]:
```
data.info()
```
executed in 14ms, finished 15:32:18 2023-11-10

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

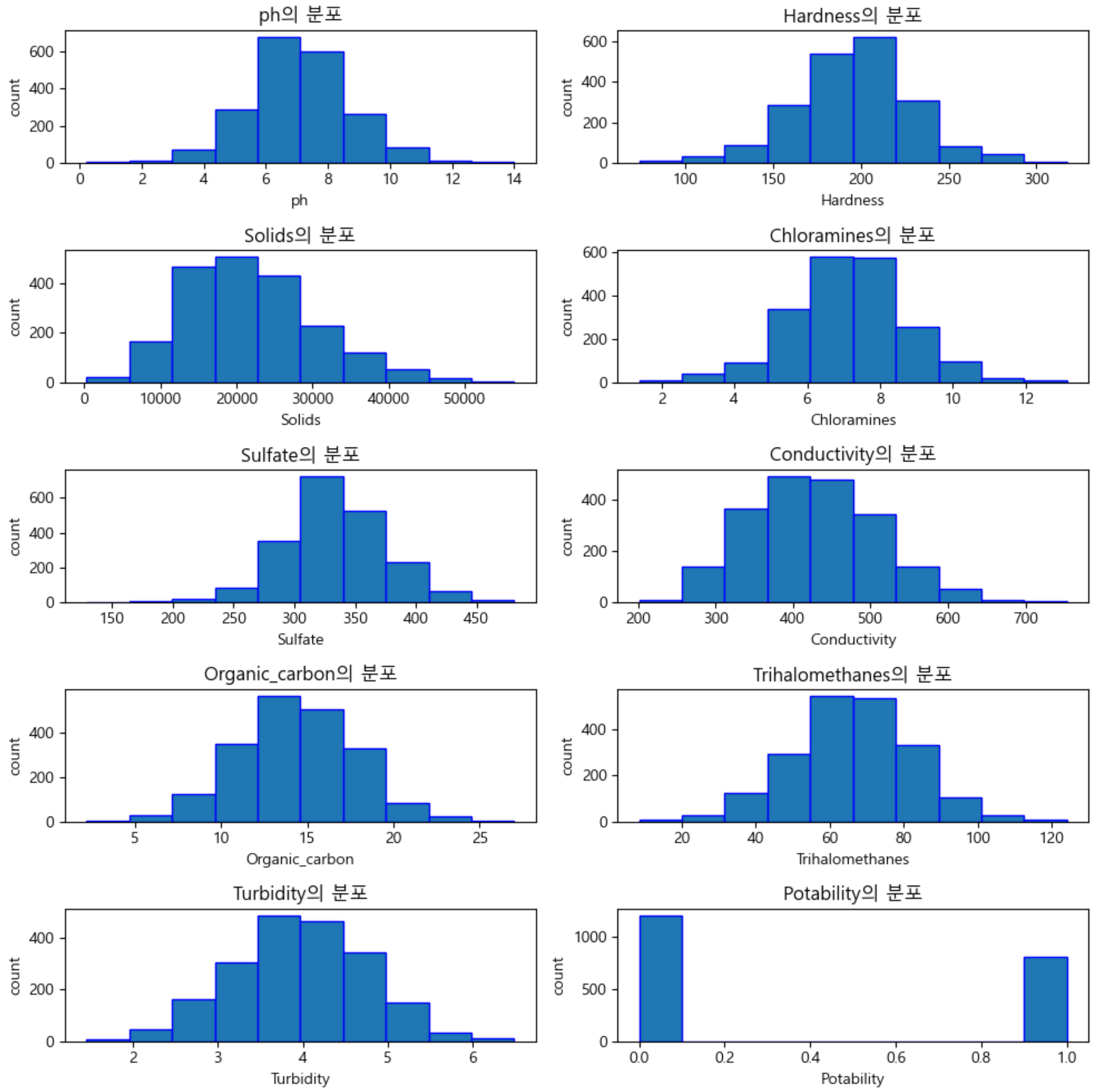
- 결측치가 많음.

- 복사본 생성

In [7]:
```
data1 = data.dropna(axis = 0)
```
executed in 14ms, finished 15:32:18 2023-11-10

In [8]:
```
plt.figure(figsize = (10,10))
for i , col in enumerate(data1.columns):
    plt.subplot(5,2,i+1)
    data1[col].plot(kind = 'hist' , edgecolor = 'b')
    plt.title(f'{col}의 분포')
    plt.xlabel(col)
    plt.ylabel('count')
plt.tight_layout()
plt.show()
```

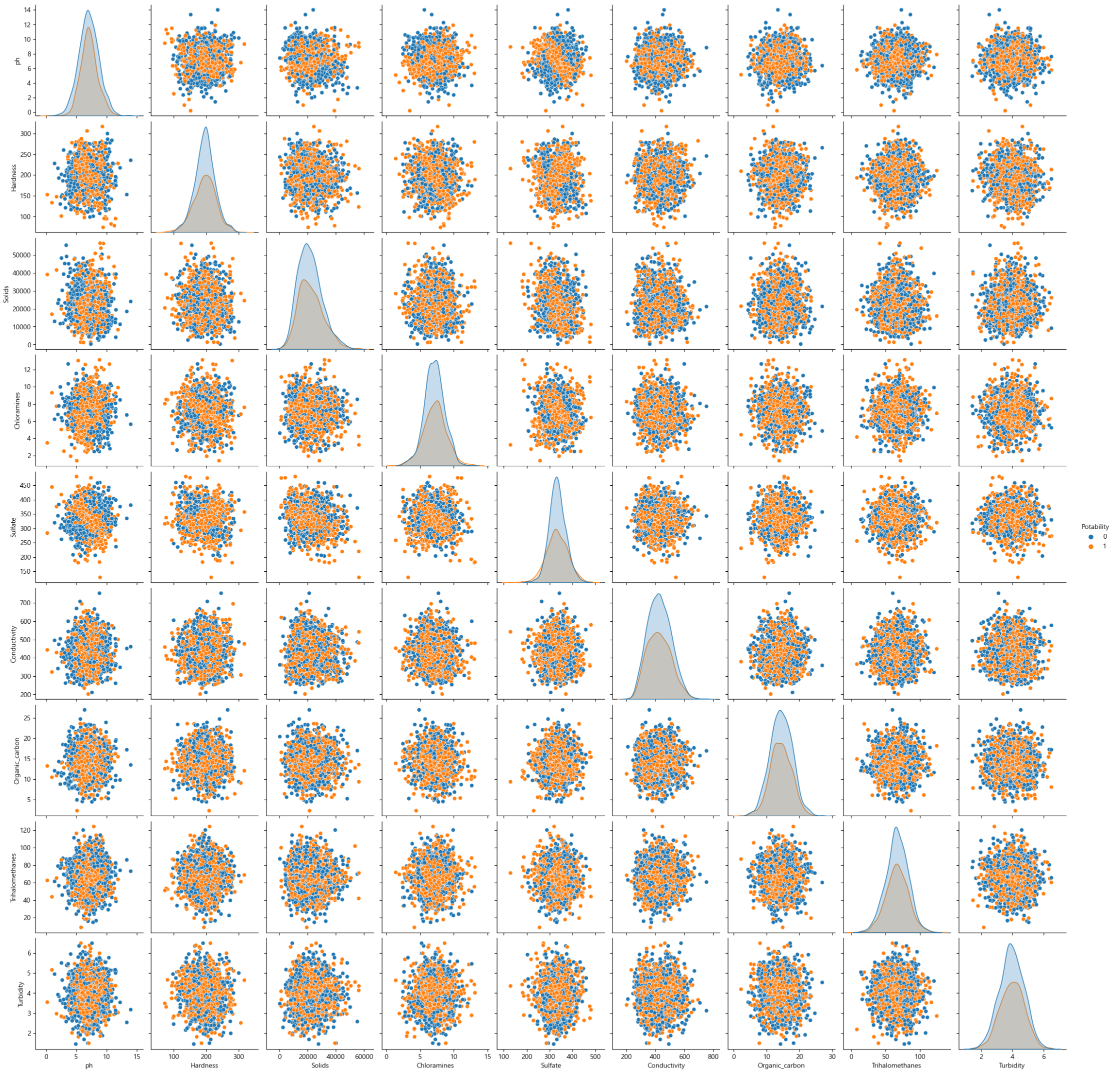
executed in 1.34s, finished 15:32:20 2023-11-10



In [9]:
```
plt.figure(figsize = (20,20))
sns.pairplot(data1 , hue = 'Potability')
```
executed in 25.1s, finished 15:32:45 2023-11-10

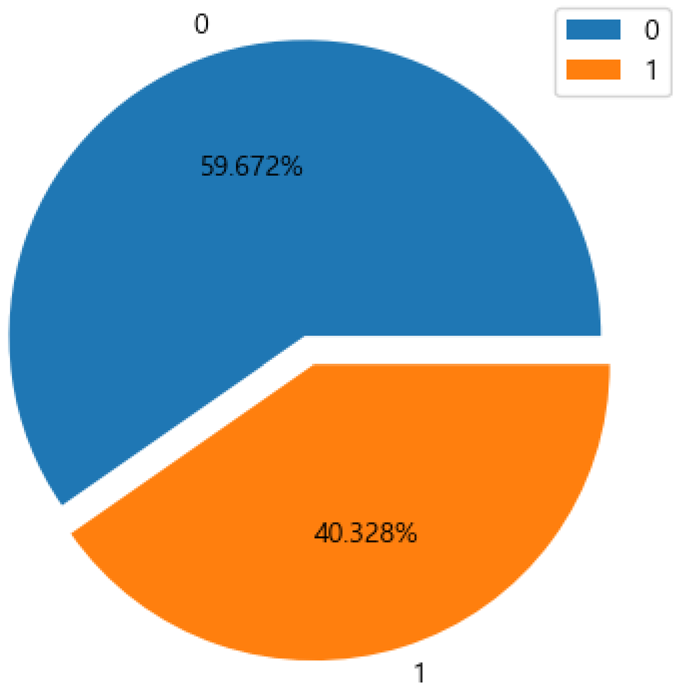
Out[9]: <seaborn.axisgrid.PairGrid at 0x1fe4c5fbe50>

<Figure size 2000x2000 with 0 Axes>



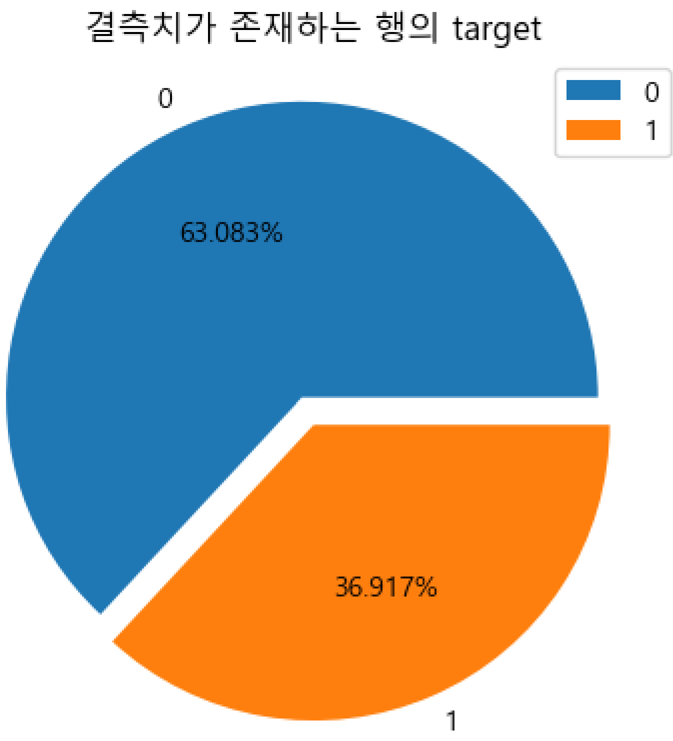
- 성능 내기가 상당히 어려워보인다.

In [10]:
```
plt.pie(data1.Potability.value_counts() ,labels=data.Potability.unique() ,  autop
plt.legend()
plt.show()
```
executed in 107ms, finished 15:32:45 2023-11-10



In [11]:
```
data_null = data[data.ph.isna()|data.Sulfate.isna()|data.Trihalomethanes.isna()]
```
executed in 13ms, finished 15:32:45 2023-11-10

In [13]:
```
plt.pie(data_null.Potability.value_counts() ,labels=data.Potability.unique() , a
plt.legend()
plt.title('결측치가 존재하는 행의 target')
plt.show()
```
executed in 106ms, finished 15:35:06 2023-11-10



In [ ]:

- ph의 결측값은 ph의 평균값으로 대체

In [14]:
```
df = data.copy()
df.dropna(subset = ['ph'] , inplace = True)
```
executed in 17ms, finished 15:35:07 2023-11-10

In [15]:
```
df
```
executed in 31ms, finished 15:35:07 2023-11-10

Out[15]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11
5	5.584087	188.313324	28748.687739	7.544869	326.678363	280.467916	8
...	...	...	...	...	...	...	
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13
3272	7.808856	193.553212	17329.802160	8.061362	NaN	392.449580	19
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16

2785 rows × 10 columns

In [16]:
```
data.ph.fillna(df.ph.mean() , inplace = True)
```
executed in 14ms, finished 15:35:07 2023-11-10

In [17]:
```
data.info()
```
executed in 16ms, finished 15:35:07 2023-11-10

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               3276 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

- Trihalomethanes : 물 속의 트리할로메탄 농도
- Conductivity: 물의 전기 전도도.

- 할로메탄이란?
  - 염소 소독시 발생하는 발암물질이다.

- 위 두 개의 변수의 결측치를 채우기 위해 모델학습 이용

In [18]:
```
data.columns
```
executed in 19ms, finished 15:35:07 2023-11-10

Out[18]:
```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

1.3 결측치가 있는 행을 제거한 후 , 거기에서 채울 열을 제거해서 학습시킬 데이터 생성

In [19]:
```
fill_Tri = data.dropna(subset = ['Trihalomethanes']).drop(['Trihalomethanes' , 'P
fill_Tri_target = data.dropna(subset = ['Trihalomethanes'])['Trihalomethanes']
```
executed in 18ms, finished 15:35:07 2023-11-10

In [20]:
```
fill_Tri.shape , fill_Tri_target.shape
```
executed in 20ms, finished 15:35:08 2023-11-10

Out[20]:
```
((3114, 7), (3114,))
```

In [21]:
```
fill_target = data[data.Trihalomethanes.isna()].dropna(axis = 1).drop('Potability
```
executed in 21ms, finished 15:35:08 2023-11-10

In [22]:
```
fill_target.shape
```
executed in 8ms, finished 15:35:08 2023-11-10

Out[22]:
```
(162, 7)
```

1.4 결측치 채우는 함수 정의

In [23]:
```
def standard(fit_model , transform_model):
    ss = StandardScaler()
    ss.fit(fit_model)
    return ss.transform(transform_model)
```
executed in 27ms, finished 15:35:08 2023-11-10

In [24]:
```
fill_Tri_scaled = standard(fill_Tri , fill_Tri)
fill_target_scaled = standard(fill_Tri , fill_target)
```
executed in 21ms, finished 15:35:08 2023-11-10

In [25]:
```
def grid(model , train_input , train_target):
    params = {'RandomForestRegressor' : { 'n_estimators' : [10, 100],
                                          'max_depth' : [6, 8, 10],
                                          'min_samples_leaf' : [8, 12],
                                          'min_samples_split' : [8, 16]},
              'Ridge' : {'alpha' : [0.001,0.01,0.1,1,10]},
              'LinearRegression' : {'n_jobs' : [1,-1]},
              'SVC' : {'C' : [0.001 , 0.01 , 0.1 , 1],
                       'kernel' : ['poly','rbf'],
                       'gamma' : [0.0001 , 0.001 , 0.01 , 0.1]},
              'RandomForestClassifier' : { 'n_estimators' : [10, 100],
                                           'max_depth' : [6, 8, 10],
                                           'min_samples_leaf' : [8, 12],
                                           'min_samples_split' : [8, 16]}
             }
    model_best = []
    for i in model:
        gs = GridSearchCV(i , params[i.__class__.__name__] , cv = 10)
        gs.fit(train_input , train_target)
        model_best.append(gs.best_estimator_)
    return model_best
```

executed in 14ms, finished 15:35:08 2023-11-10

In [26]:
```
model = [Ridge(random_state = 42) , RandomForestRegressor(random_state = 42) , L
best_model = grid(model , fill_Tri_scaled , fill_Tri_target)
```

executed in 5m 0s, finished 15:40:08 2023-11-10

In [27]:
```
best_model
```

executed in 14ms, finished 15:40:08 2023-11-10

Out[27]:
```
[Ridge(alpha=10, random_state=42),
 RandomForestRegressor(max_depth=6, min_samples_leaf=8, min_samples_split=8,
                       random_state=42),
 LinearRegression(n_jobs=1)]
```

In [28]:
```
def fit(model , data , target , fill_target):
    model.fit(data , target)
    return model.predict(fill_target)
```

executed in 13ms, finished 15:40:08 2023-11-10

1.5 3개의 모델로 학습시켜서 예측한 값을 result에 저장하고 , 3개의 값의 평균으로 결측치 대체

In [29]:
```
result = []
for i in best_model:
    result.append(fit(i , fill_Tri_scaled , fill_Tri_target , fill_target_scaled)
```

executed in 2.08s, finished 15:40:10 2023-11-10

In [30]:
```
Tri_fill = np.mean(result , axis = 0)
```

executed in 15ms, finished 15:40:10 2023-11-10

In [31]:
```
data.loc[data['Trihalomethanes'].isna(), 'Trihalomethanes'] = Tri_fill
```
executed in 29ms, finished 15:40:10 2023-11-10

In [32]:
```
data.info()
```
executed in 14ms, finished 15:40:10 2023-11-10

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ph               3276 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3276 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

In [33]:
```
fill_Sul = data.dropna(subset = ['Sulfate']).drop(['Potability' , 'Sulfate'] , a
fill_Sul_target = data.dropna(subset = ['Sulfate'])['Sulfate']
```
executed in 14ms, finished 15:40:10 2023-11-10

In [34]:
```
fill_Sul.shape , fill_Sul_target.shape
```
executed in 14ms, finished 15:40:10 2023-11-10

Out[34]:
```
((2495, 8), (2495,))
```

In [35]:
```
fill_target_Sul = data[data.Sulfate.isna()].dropna(axis = 1).drop('Potability' ,
```
executed in 11ms, finished 15:40:10 2023-11-10

In [36]:
```
fill_Sul_scaled = standard(fill_Sul , fill_Sul)
fill_target_Sul_scaled = standard(fill_Sul , fill_target_Sul)
```
executed in 14ms, finished 15:40:10 2023-11-10

In [37]:
```
model = [Ridge(random_state = 42) , RandomForestRegressor(random_state = 42) , L
best_model = grid(model , fill_Sul_scaled , fill_Sul_target)
```
executed in 4m 18s, finished 15:44:28 2023-11-10

In [38]:
```
best_model
```
executed in 15ms, finished 15:44:28 2023-11-10

Out[38]:
```
[Ridge(alpha=10, random_state=42),
 RandomForestRegressor(max_depth=6, min_samples_leaf=12, min_samples_split=8,
                       random_state=42),
 LinearRegression(n_jobs=1)]
```

In [39]:
```
Sulfate = []

for i in best_model:
    Sulfate.append(fit(i , fill_Sul_scaled , fill_Sul_target , fill_target_Sul_sc
```
executed in 1.84s, finished 15:44:30 2023-11-10

In [40]:
```
Sul_fill = np.mean(Sulfate , axis = 0)
```
executed in 14ms, finished 15:44:30 2023-11-10

In [41]:
```
data.loc[data['Sulfate'].isna(), 'Sulfate'] = Sul_fill
```
executed in 13ms, finished 15:44:30 2023-11-10

In [42]:
```
data.info()
```
executed in 14ms, finished 15:44:30 2023-11-10

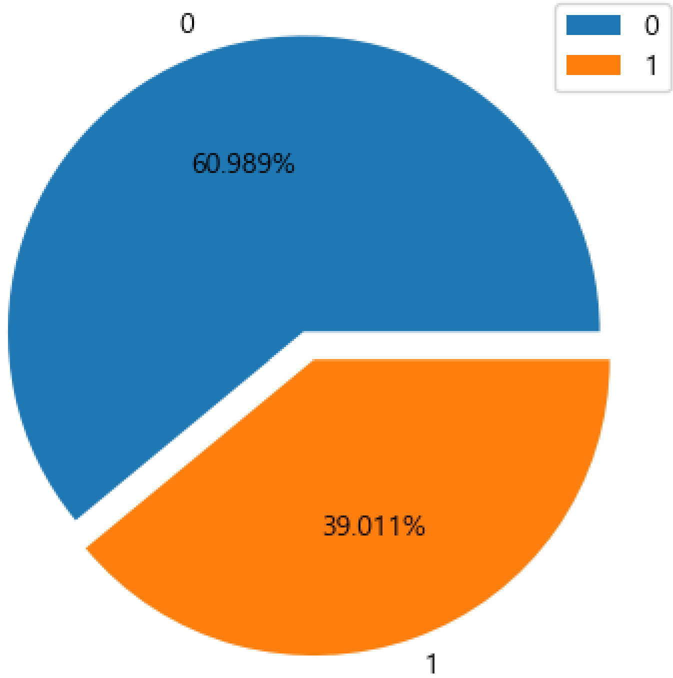
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               3276 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          3276 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3276 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

- 결측치 대체 완료

In [43]:
```
data.Potability.value_counts()
```
executed in 14ms, finished 15:44:30 2023-11-10

Out[43]:
```
0    1998
1    1278
Name: Potability, dtype: int64
```

In [44]:
```
plt.pie(data.Potability.value_counts() ,labels=data.Potability.unique() ,  autopc
plt.legend()
plt.show()
```
executed in 91ms, finished 15:44:30 2023-11-10



1.6 학습 데이터와 테스트 데이터로 나누고 , StandScaler를 이용한 정규화

In [45]:
```
def divide(x , y):
    train_input , test_input , train_target , test_target = train_test_split(x ,
    ss = StandardScaler()
    train_scaled = ss.fit_transform(train_input)
    test_scaled = ss.transform(test_input)

    return train_scaled , test_scaled , train_target , test_target
```
executed in 13ms, finished 15:44:31 2023-11-10

In [46]:
```
train_input , test_input , train_target , test_target =divide(data.iloc[:,:-1] ,
```
executed in 14ms, finished 15:44:31 2023-11-10

In [47]:
```
rf = RandomForestClassifier(random_state = 42)
rf.fit(train_input , train_target)
print(classification_report(test_target , rf.predict(test_input)))
```
executed in 2.00s, finished 15:44:33 2023-11-10

```
              precision    recall  f1-score   support

           0       0.67      0.89      0.76       400
           1       0.64      0.31      0.42       256

    accuracy                           0.66       656
   macro avg       0.65      0.60      0.59       656
weighted avg       0.66      0.66      0.63       656
```

1.7 교차검증

In [48]:
```
scores = cross_validate(rf , train_input , train_target , cv = 10 , scoring = 'p
np.mean(scores['test_score'])
```
executed in 17.1s, finished 15:44:50 2023-11-10

Out[48]: 0.6278899706951955

In [49]:
```
scores = cross_validate(rf , train_input , train_target , cv = 10)
np.mean(scores['test_score'])
```
executed in 17.0s, finished 15:45:07 2023-11-10

Out[49]: 0.6622137404580153

1.8 특성공학을 이용하여 feature의 개수 늘리기

In [50]:
```

def poly(train_input , test_input):
    poly = PF(include_bias = False)
    train_poly = poly.fit_transform(train_input)
    test_poly = poly.transform(test_input)
    return train_poly , test_poly
```
executed in 14ms, finished 15:45:07 2023-11-10

In [51]:
```
train_poly , test_poly = poly(train_input, test_input)
```
executed in 13ms, finished 15:45:07 2023-11-10

In [52]:
```
train_poly.shape , train_input.shape
```
executed in 15ms, finished 15:45:07 2023-11-10

Out[52]: ((2620, 54), (2620, 9))

- 특성이 늘어남.

In [53]:
```
rf = RandomForestClassifier(random_state = 42)
rf.fit(train_poly , train_target)
print(classification_report(test_target , rf.predict(test_poly)))
```
executed in 3.99s, finished 15:45:11 2023-11-10

```
              precision    recall  f1-score   support

           0       0.68      0.90      0.77       400
           1       0.68      0.32      0.44       256

    accuracy                           0.68       656
   macro avg       0.68      0.61      0.61       656
weighted avg       0.68      0.68      0.64       656
```

- 효과가 미미하다.

1.9 모든 분류모델을 써서 '정밀도'를 측정해보기

In [54]:
```
data.describe().T
```
executed in 45ms, finished 15:45:11 2023-11-10

Out[54]:

	count	mean	std	min	25%	50'
ph	3276.0	7.080795	1.469956	0.000000	6.277673	7.0807ę
Hardness	3276.0	196.369496	32.879761	47.432000	176.850538	196.96762
Solids	3276.0	22014.092526	8768.570828	320.942611	15666.690297	20927.8336(
Chloramines	3276.0	7.122277	1.583085	0.352000	6.127421	7.1302ę
Sulfate	3276.0	333.789000	36.402113	129.000000	316.134811	333.49987
Conductivity	3276.0	426.205111	80.824064	181.483754	365.734414	421.8849(
Organic_carbon	3276.0	14.284970	3.308162	2.200000	12.065801	14.2183:
Trihalomethanes	3276.0	66.397363	15.770505	0.738000	56.647656	66.4934ę
Turbidity	3276.0	3.966786	0.780382	1.450000	3.439711	3.95502
Potability	3276.0	0.390110	0.487849	0.000000	0.000000	0.0000(

In [55]:
```
kn = KNeighborsClassifier()
lr = LogisticRegression(random_state = 42)
rf = RandomForestClassifier(random_state = 42)
svc = SVC(random_state = 42)
xgb = XGBClassifier(random_state = 42)
lgb = LGBMClassifier(random_state = 42)


model = [kn,lr,rf,svc,xgb,lgb]
```
executed in 13ms, finished 15:45:11 2023-11-10

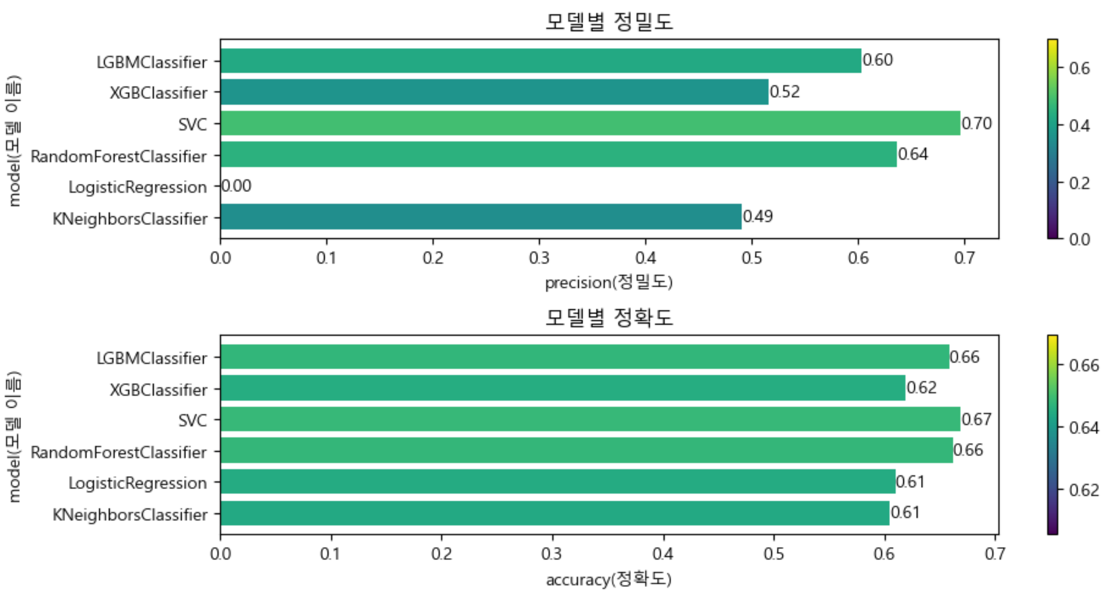
In [56]:
```
#정확도 , 정밀도를 측정
def add_precision(model):
    names.append(model.__class__.__name__)
    model.fit(train_input , train_target)
    precision.append(precision_score(test_target , model.predict(test_input)))
    accuracy.append(accuracy_score(test_target , model.predict(test_input)))
```
executed in 13ms, finished 15:45:11 2023-11-10

In [57]:
```
x = data.iloc[:,:-1]
y = data.iloc[:,-1]

train_input , test_input , train_target , test_target = divide(x , y)
```
executed in 14ms, finished 15:45:11 2023-11-10

In [58]:
```
names = []
precision = []
accuracy = []
for i in model:
    add_precision(i)
```
executed in 2.59s, finished 15:45:13 2023-11-10

In [59]:
```
plt.figure(figsize = (10,5))
plt.subplot(2,1,1)
plt.barh(names , precision , color=plt.cm.viridis(precision))
sm = plt.cm.ScalarMappable(cmap='viridis', norm=plt.Normalize(vmin=min(precision
cbar = plt.colorbar(sm)

for i, val in enumerate(precision):
    plt.text(val, i, f'{val:.2f}', va='center')

plt.xlabel('precision(정밀도)')
plt.ylabel('model(모델 이름)')
plt.title('모델별 정밀도')

plt.subplot(2,1,2)
plt.barh(names , accuracy , color=plt.cm.viridis(accuracy))
sm = plt.cm.ScalarMappable(cmap='viridis', norm=plt.Normalize(vmin=min(accuracy)
cbar = plt.colorbar(sm)
for i, val in enumerate(accuracy):
    plt.text(val, i, f'{val:.2f}', va='center')

plt.xlabel('accuracy(정확도)')
plt.ylabel('model(모델 이름)')
plt.title('모델별 정확도')


plt.tight_layout()

plt.show()
```

executed in 390ms, finished 15:45:14 2023-11-10



1.10 정밀도가 중요하다고 판단 , 정밀도를 올리는 방향

1.10.1 SVM

In [60]:
```
train_poly , test_poly = poly(train_input, test_input)
```
executed in 14ms, finished 15:45:14 2023-11-10

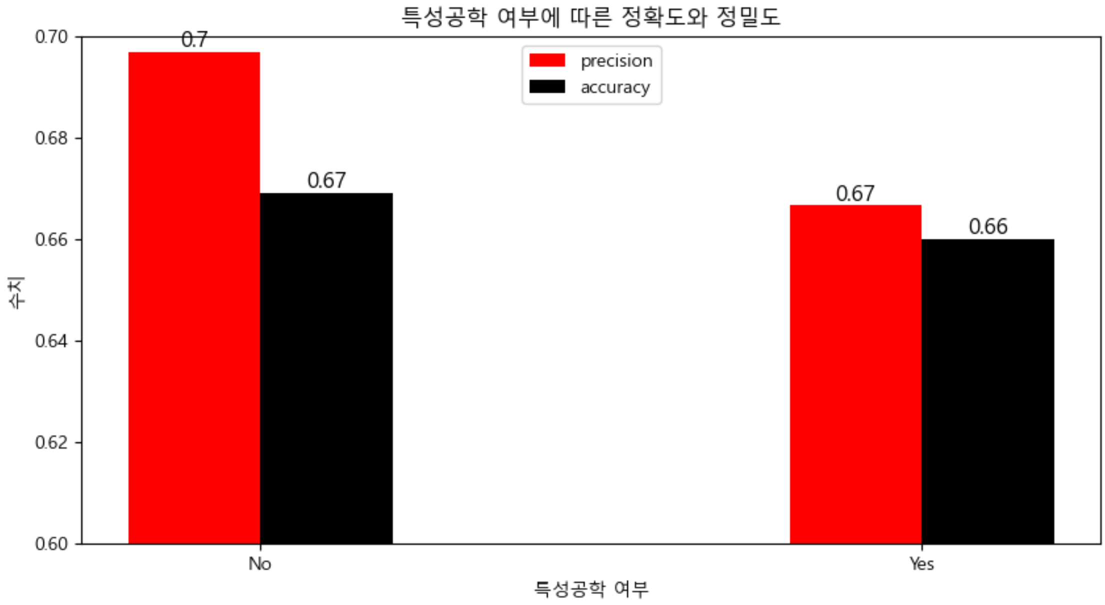
In [61]:
```
svc.fit(train_poly , train_target)
```
executed in 265ms, finished 15:45:14 2023-11-10

Out[61]:
```
▾ SVC
SVC(random_state=42)
```

In [62]:
```
def text(plot):
    for i in plot:
        height = i.get_height()
        plt.text(i.get_x()+i.get_width()/2.0,height,  round(height, 2) ,  ha = '
```
executed in 14ms, finished 15:45:14 2023-11-10

In [63]:
```
def poly_comparison(model , train_input , test_input):
    train_poly , test_poly = poly(train_input , test_input)
    p_list = ['No' , 'Yes']
    prec = []
    accu = []
    model.fit(train_input , train_target)
    prec.append(precision_score(test_target , model.predict(test_input)))
    accu.append(accuracy_score(test_target , model.predict(test_input)))

    model.fit(train_poly , train_target)
    prec.append(precision_score(test_target , model.predict(test_poly)))
    accu.append(accuracy_score(test_target , model.predict(test_poly)))

    plt.figure(figsize = (10,5))
    width = 0.2
    pre = plt.bar(np.arange(len(p_list))-width/2 , prec ,width = width , color =
    acc = plt.bar(np.arange(len(p_list))+width/2 , accu ,width = width , color =
    plt.xticks(np.arange(len(p_list)) , p_list)
    plt.legend(loc = 'upper center')
    plt.xlabel('특성공학 여부')
    plt.ylim(0.6 , 0.7)
    plt.ylabel('수치')
    plt.title('특성공학 여부에 따른 정확도와 정밀도')
    for i in [pre , acc]:
        text(i)
    plt.show()
```
executed in 14ms, finished 15:45:14 2023-11-10

In [64]:
```
poly_comparison(svc , train_input , test_input)
```
executed in 979ms, finished 15:45:15 2023-11-10



In [ ]:

In [65]:
```
svc_best = grid([SVC(random_state = 42 , probability=True)] , train_input , trai
svc_best
```
executed in 3m 58s, finished 15:49:13 2023-11-10

Out[65]:
```
[SVC(C=1, gamma=0.1, probability=True, random_state=42)]
```

In [66]:
```
svc_best = svc_best[0]
```
executed in 14ms, finished 15:49:13 2023-11-10

In [67]:
```
svc_best.fit(train_input , train_target)
```
executed in 1.15s, finished 15:49:14 2023-11-10

Out[67]:
```
SVC
SVC(C=1, gamma=0.1, probability=True, random_state=42)
```

In [68]:
```
pred = svc_best.decision_function(test_input)
```
executed in 107ms, finished 15:49:14 2023-11-10

In [69]:
```
confusion_matrix(test_target , svc_best.predict(test_input))
```
executed in 123ms, finished 15:49:15 2023-11-10

Out[69]:
```
array([[372,  28],
       [190,  66]], dtype=int64)
```

In [70]:
```
print(classification_report(test_target , svc_best.predict(test_input)))
```
executed in 108ms, finished 15:49:15 2023-11-10

```
              precision    recall  f1-score   support

           0       0.66      0.93      0.77       400
           1       0.70      0.26      0.38       256

    accuracy                           0.67       656
   macro avg       0.68      0.59      0.58       656
weighted avg       0.68      0.67      0.62       656
```

In [71]:
```
svc_best.predict_proba(test_input)
```
executed in 106ms, finished 15:49:15 2023-11-10

Out[71]:
```
array([[0.77541742, 0.22458258],
       [0.80223513, 0.19776487],
       [0.80184952, 0.19815048],
       ...,
       [0.6236285 , 0.3763715 ],
       [0.73972162, 0.26027838],
       [0.60709308, 0.39290692]])
```

In [72]:
```
def proba(model):
    fper , tper , thresholds = roc_curve(test_target , model.predict_proba(test_
    return fper , tper , thresholds
```
executed in 14ms, finished 15:49:15 2023-11-10

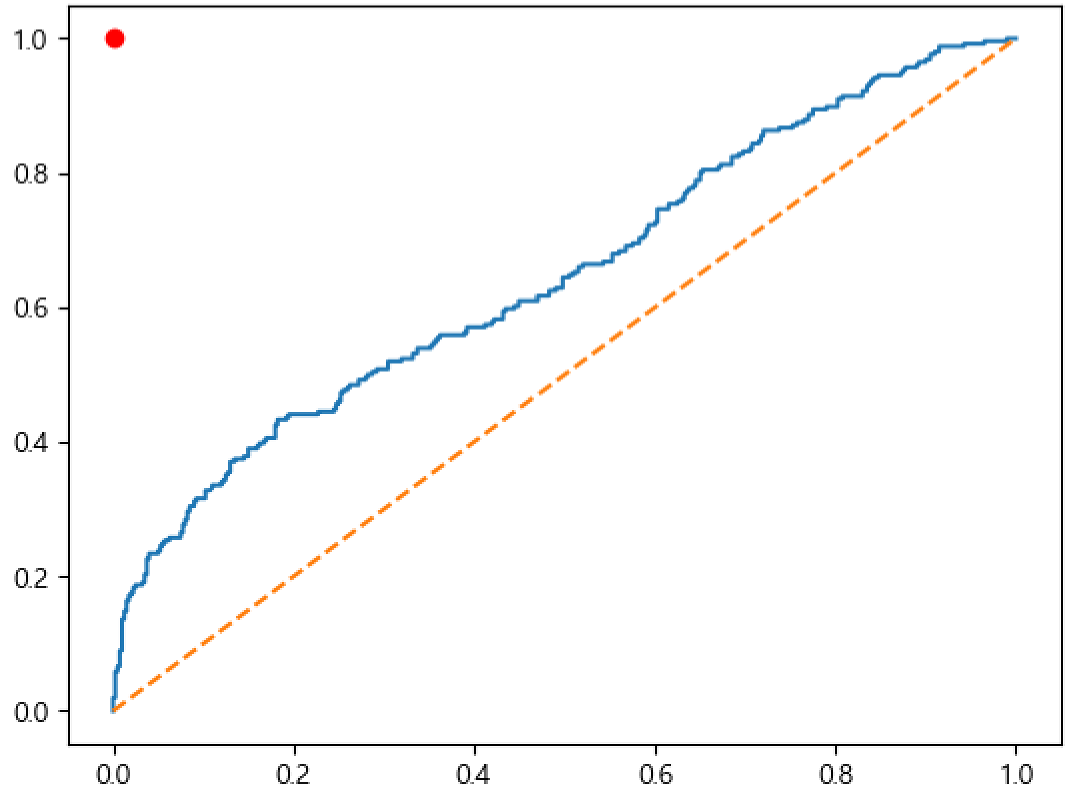
In [ ]:

In [73]:
```
fper , tper , thresholds = proba(svc_best)
```
executed in 108ms, finished 15:49:15 2023-11-10

In [74]:
```
roc_auc_score(test_target , svc_best.predict_proba(test_input)[:,1])
```
executed in 106ms, finished 15:49:15 2023-11-10

Out[74]:
```
0.6476123046875
```

In [75]:
```
plt.plot(fper , tper)
plt.scatter(0,1 , color = 'red')
plt.plot([0,1] , [0,1] , ls = '--')
plt.show()
```
executed in 107ms, finished 15:49:15 2023-11-10



- x축 : 정답이 Negative(0)인 것 중에서 모델이 Positive라고 잘못 예측한 것의 비율이 된다.
- y축 : 정답이 Positive(1)인 것들 중에서 정말로 정답을 맞춘 수의 비율이 된다. (재현율)

1.10.2 RandomForestClassification

In [76]:
```
rf_best = grid([RandomForestClassifier(random_state = 42)] , train_input , train_
```
executed in 1m 59.0s, finished 15:51:14 2023-11-10

In [77]:
```
rf_best = rf_best[0]
```
executed in 12ms, finished 15:51:14 2023-11-10

In [78]:
```
rf_best.fit(train_input , train_target)
```
executed in 1.28s, finished 15:51:16 2023-11-10

Out[78]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=10, min_samples_leaf=8, min_samples_split=8,
                       random_state=42)
```

In [79]:
```
rf_best.predict_proba(test_input)
```
executed in 29ms, finished 15:51:16 2023-11-10

Out[79]:
```
array([[0.67966937, 0.32033063],
       [0.71645111, 0.28354889],
       [0.63582395, 0.36417605],
       ...,
       [0.67555439, 0.32444561],
       [0.68443376, 0.31556624],
       [0.56179748, 0.43820252]])
```

In [80]:
```
print(classification_report(test_target , rf_best.predict(test_input)))
```
executed in 30ms, finished 15:51:16 2023-11-10

```
              precision    recall  f1-score   support

           0       0.65      0.94      0.77       400
           1       0.71      0.22      0.33       256

    accuracy                           0.66       656
   macro avg       0.68      0.58      0.55       656
weighted avg       0.68      0.66      0.60       656
```

In [81]:
```
imp = rf_best.feature_importances_
```
executed in 15ms, finished 15:51:16 2023-11-10

In [82]:
```
imp2 = []
for i in imp:
    imp2.append(round(i , 3))
```
executed in 15ms, finished 15:51:16 2023-11-10

In [83]:
```
col = data.columns[:-1]
```
executed in 14ms, finished 15:51:16 2023-11-10

In [84]:
```
feature = dict(zip(col , imp2))
```
executed in 14ms, finished 15:51:16 2023-11-10

In [85]:
```
feature
```
executed in 14ms, finished 15:51:16 2023-11-10

Out[85]:
```
{'ph': 0.164,
 'Hardness': 0.124,
 'Solids': 0.113,
 'Chloramines': 0.109,
 'Sulfate': 0.206,
 'Conductivity': 0.082,
 'Organic_carbon': 0.069,
 'Trihalomethanes': 0.07,
 'Turbidity': 0.063}
```

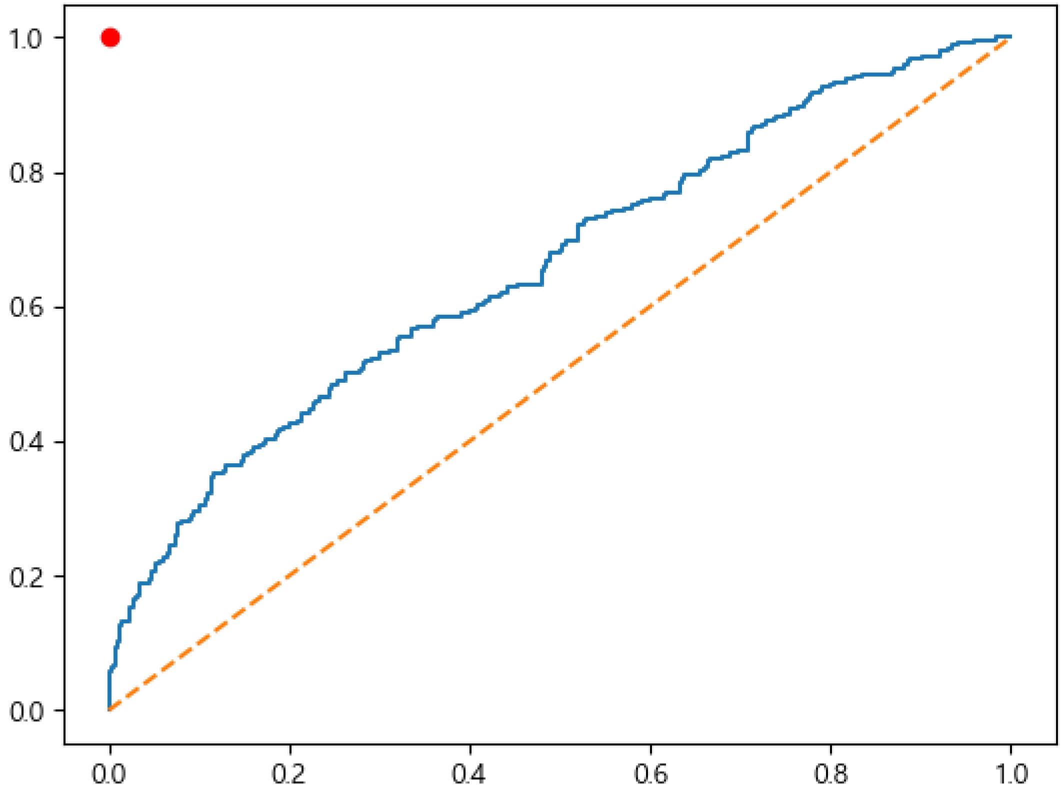
In [86]:
```
fper , tper , thresholds = proba(rf_best)
```
executed in 31ms, finished 15:51:16 2023-11-10

In [87]:
```
roc_auc_score(test_target , rf_best.predict_proba(test_input)[:,1])
```
executed in 30ms, finished 15:51:16 2023-11-10

Out[87]: 0.659560546875

In [88]:
```
plt.plot(fper , tper)
plt.scatter(0,1 , color = 'red')
plt.plot([0,1] , [0,1] , ls = '--')
```
executed in 152ms, finished 15:51:16 2023-11-10

Out[88]: [<matplotlib.lines.Line2D at 0x1fe53bc27c0>]



In [89]:
```
def acc_pred(model):
    try:
        thresholds_list = [0.1,0.2,0.3,0.4 , 0.5]
        pred = model.decision_function(test_input)
    except:
        thresholds_list = [0.4,0.45,0.5,0.55,0.6]
        pred = model.predict_proba(test_input)[:,1]



    accuracy_list = []
    precision_list = []
    for i in thresholds_list:
        y_pred = np.where(pred>i , 1 , 0)
        acc = accuracy_score(test_target , y_pred)
        precision = precision_score(test_target , y_pred)
        accuracy_list.append(acc)
        precision_list.append(precision)
#         print(f'임계값 : {i}')
#         print(f'정확도 : {acc:.4f}₩n정밀도 : {precision:.4f}')
#         print()
    return thresholds_list , accuracy_list , precision_list
```

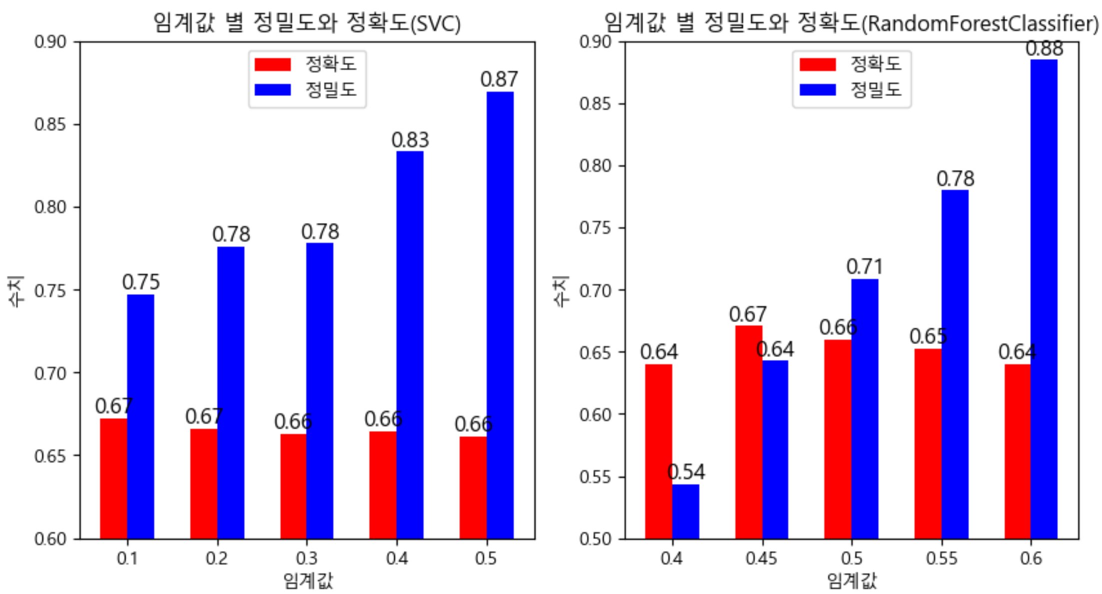
executed in 13ms, finished 15:51:16 2023-11-10

In [93]:
```
def plotting(model1 , model2):
    thresholds_list , accuracy_list , precision_list = acc_pred(model1)
    plt.figure(figsize = (10,5))
    plt.subplot(1,2,1)
    width = 0.3
    acc = plt.bar(np.arange(len(thresholds_list)) - width/2 , accuracy_list ,wid
    pre = plt.bar(np.arange(len(thresholds_list)) + width/2 , precision_list ,wi

    for i in [acc , pre]:
        text(i)


    plt.xticks(np.arange(len(thresholds_list)) , thresholds_list)
    plt.xlabel('임계값')
    plt.ylabel('수치')
    plt.title(f'임계값 별 정밀도와 정확도({model1.__class__.__name__})')
    plt.legend(loc = 'upper center')
    plt.ylim(0.6 , 0.9)

    plt.subplot(1,2,2)
    thresholds_list , accuracy_list , precision_list = acc_pred(model2)
    width = 0.3
    acc = plt.bar(np.arange(len(thresholds_list)) - width/2 , accuracy_list ,wid
    pre = plt.bar(np.arange(len(thresholds_list)) + width/2 , precision_list ,wi

    for i in [acc , pre]:
        text(i)


    plt.xticks(np.arange(len(thresholds_list)) , thresholds_list)
    plt.xlabel('임계값')
    plt.ylabel('수치')
    plt.title(f'임계값 별 정밀도와 정확도({model2.__class__.__name__})')
    plt.legend(loc = 'upper center')
    plt.ylim(0.5 , 0.9)
```

executed in 13ms, finished 15:51:16 2023-11-10

In [94]:
```
plotting(svc_best , rf_best)
```

executed in 470ms, finished 15:51:17 2023-11-10



2 결론

In [95]:
```
print('SVC')
for i in [0.4, 0.5]:
    predic = np.where(svc_best.decision_function(test_input)>i , 1 , 0)
    print(f'임계값 : {i}')
    print(confusion_matrix(test_target , predic))
    print()

print('RandomForest')
for i in [0.55 , 0.6]:
    predic = np.where(rf_best.predict_proba(test_input)[:,1] > i , 1 , 0)
    print(f'임계값 : {i}')
    print(confusion_matrix(test_target , predic))
    print()
```

executed in 233ms, finished 15:51:17 2023-11-10

```
SVC
임계값 : 0.4
[[391   9]
 [211  45]]

임계값 : 0.5
[[394   6]
 [216  40]]

RandomForest
임계값 : 0.55
[[389  11]
 [217  39]]

임계값 : 0.6
[[397   3]
 [233  23]]
```

- TP가 가장 많은 것 채택

In [96]:
```
prediction = np.where(svc_best.decision_function(test_input) > 0.4 , 1 , 0)
```

executed in 108ms, finished 15:51:17 2023-11-10

In [97]:
```
print(classification_report(test_target , prediction))
```

executed in 14ms, finished 15:51:17 2023-11-10

```
              precision    recall  f1-score   support

           0       0.65      0.98      0.78       400
           1       0.83      0.18      0.29       256

    accuracy                           0.66       656
   macro avg       0.74      0.58      0.54       656
weighted avg       0.72      0.66      0.59       656
```

- 정확도를 적당히 잃으면서 정밀도를 끌어올렸다.

In [ ]:

```

```