DATA KUBWA

평가 및 적용

평가 지표

DATA KUBWA

회귀 성능 평가 지표

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Where,

$\hat{y}$ – *predicted value of* y

$\bar{y}$ – *mean value of* y

분류 성능 평가 지표

❶ 정확도 (Accuracy)

❷ 오차행렬 (Confusion Matrix)

❸ 정밀도(Precision) & 재현율(Recall)

❹ F1 스코어

❺ ROC AUC

❶ 정확도(ACCURACY)

DATA KUBWA

$$\frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

정확도 = ML model performance?

❶ 정확도(ACCURACY)

- ✓ No 학습
- ✓ Sex 남성이면 0, 여성이면 1로 예측
- ✓ 정확도는?
- ✓ 78.77%

❷ 오차행렬 (CONFUSION MATRIX)

- from sklearn.metrics import confusion_matrix
- confusion_matrix(y_test, mypredictions)

```
array([[92, 18],
       [20, 49]], dtype=int64)
```

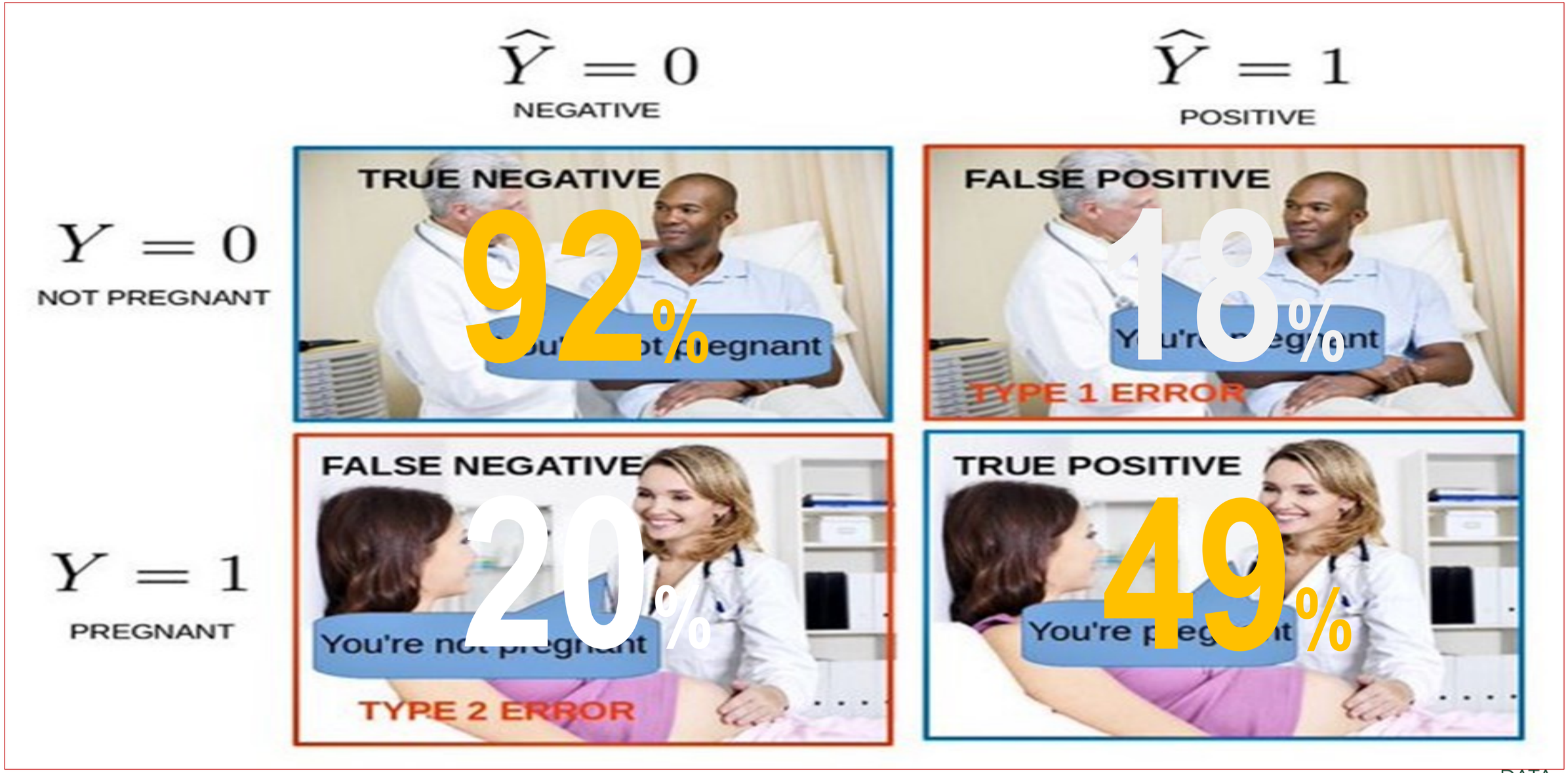
- pd.DataFrame(confusion_matrix(y_test, mypredictions), columns=['0', '1'])

	0	1
0	92	18
1	20	49

- TN (True Negative): 예측 값 0, 실제 값 0
- FP (False Positive): 예측 값 1, 실제 값 0
- FN (False Negative): 예측 값 0, 실제 값 1
- TP (True Positive): 예측 값 1, 실제 값 1

		Prediction	
		0	1
Actual	0	TN	FP
	1	FN	TP

❷ 오차행렬 (CONFUSION MATRIX)



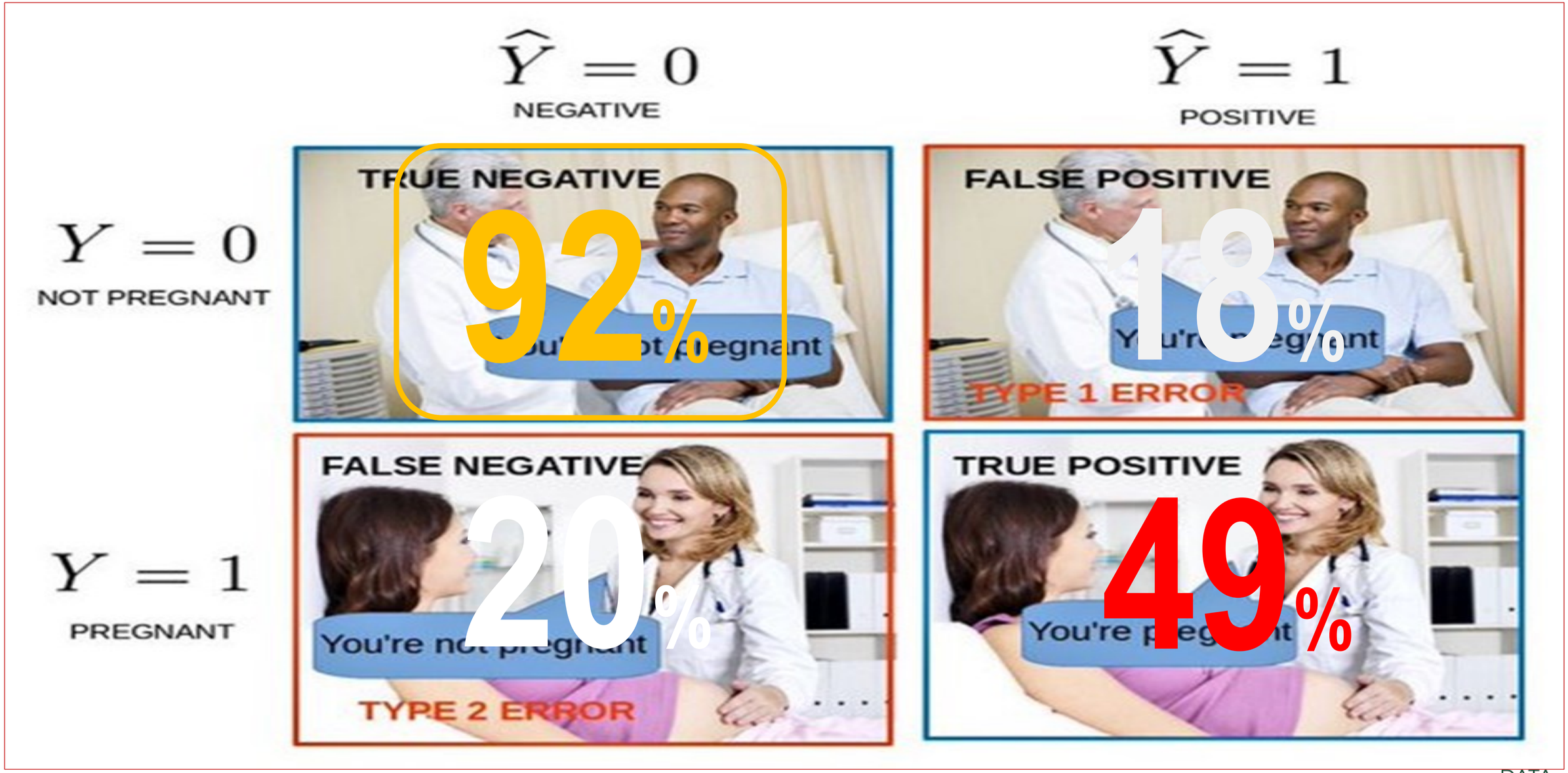
❷ 오차행렬 (CONFUSION MATRIX)

DATA KUBWA

- 정확도 = 예측값과 실제값이 얼마나 동일한가?

$$\frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}} = \frac{(TN+TP)}{(TN+FP+FN+TP)}$$

❷ 오차행렬 (CONFUSION MATRIX)



❸ 정밀도(PRECISION) & 재현율(RECALL)

DATA KUBWA

- 정밀도
- 실제 음성인 데이터 예측을 양성으로 잘못 판단 시 업무상 큰 영향이 발생하는 경우

$$\frac{TP}{(FP+TP)}$$

False Positive + True Positive

- 재현율
- 실제 양성 데이터를 음성으로 잘못 판단 시 업무 상 큰 영향이 발생하는 경우

$$\frac{TP}{(FN+TP)}$$

False Negative + True Positive

❸ 정밀도(PRECISION) & 재현율(RECALL)

	정밀도	재현율
설명	양성이라고 예측한 모든 경우에서 실제 양성인 경우	실제 양성인 모든 경우에서 양성이라고 예측한 경우
암 판단 모델		√
금융 사기 모델		√
재판	√	
스팸 메일 판단	√	
목표	FP를 낮추는 것	FN을 낮추는 것

정확성 문제

다음 중 어떤 시나리오에서 높은 정확성 값을 통해 ML 모델이 제대로 실행되고 있음을 알 수 있나요?

비싼 로봇 닭이 혼잡한 도로를 하루에 수천 번 건넙니다. ML 모델은 교통 패턴을 평가하여 이 닭이 안전하게 길을 건널 수 있는 때를 99.99%의 정확성으로 예측합니다.

전 세계 인구의 0.01%가 치명적이지만 치유 가능한 질병으로 고통받고 있습니다. ML 모델은 증상을 특징으로 사용하여 99.99%의 정확성으로 이러한 질병을 예측합니다.

룰렛 게임에서 회전하는 룰렛 휠에 공이 떨어지면 결국 슬롯 38개 중 하나로 들어갑니다. ML 모델은 시각적 특징(공의 회전, 공이 떨어졌을 때 룰렛 휠의 위치, 룰렛 휠 위 공의 높이)을 사용하여 공이 들어갈 슬롯을 4%의 정확성으로 예측할 수 있습니다.

정밀도 문제

이메일을 '스팸'과 '스팸 아님', 두 카테고리로 구분하는 분류 모델을 가정해 보겠습니다. 분류 임계값을 올리면 정밀도는 어떻게 될까요?

- 아마도 감소할 것입니다.
- 확실히 증가합니다.
- 확실히 감소합니다.
- 아마도 증가할 것입니다.

재현율 문제

이메일을 '스팸'과 '스팸 아님', 두 카테고리로 구분하는 분류 모델을 가정해 보겠습니다. 분류 임계값을 올리면 재현율은 어떻게 될까요?

항상 증가합니다.

항상 감소하거나 그대로 유지됩니다.

항상 일정하게 유지됩니다.

❸ 정밀도(PRECISION) & 재현율(RECALL)

DATA KUBWA

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

def get_clf_eval(y_test, pred):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)
    print('오차 행렬')
    print(confusion)
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}'.format(accuracy, precision, recall))
```

```
오차 행렬
[[108  10]
 [ 14  47]]
정확도: 0.8659, 정밀도: 0.8246, 재현율: 0.7705
```

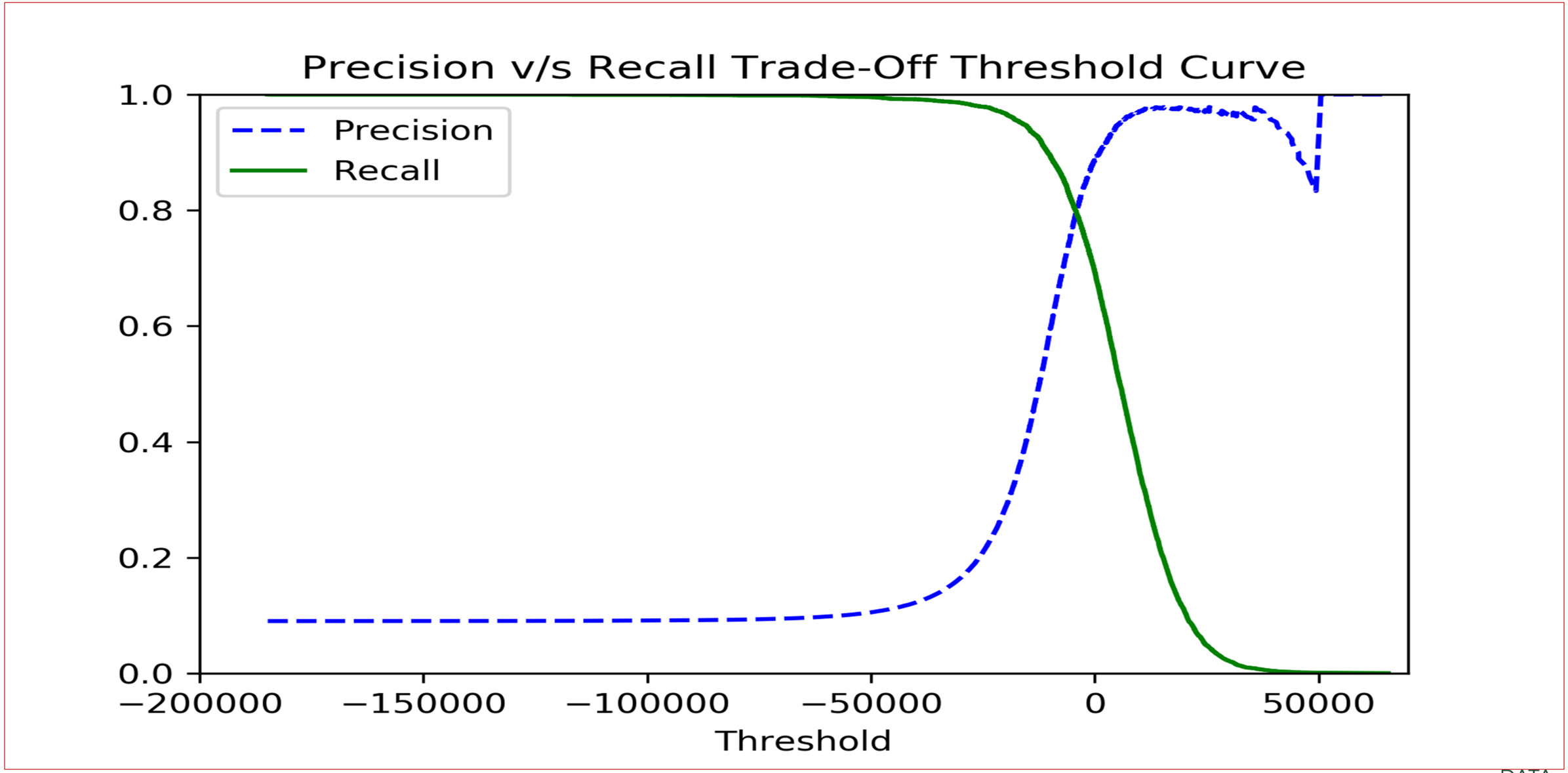
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# 원본 데이터를 재로딩, 데이터 가공, 학습 데이터/테스트 데이터 분할
titanic_df = pd.read_csv('./train.csv')
y_titanic_df = titanic_df['Survived']
X_titanic_df = titanic_df.drop('Survived', axis=1)
X_titanic_df = transform_features(X_titanic_df)

X_train, X_test, y_train, y_test = train_test_split(X_titanic_df, y_titanic_df, test_size=0.20, random_state=11)
lr_clf = LogisticRegression()

lr_clf.fit(X_train, y_train)
pred = lr_clf.predict(X_test)
get_clf_eval(y_test, pred)
```

TRADE-OFF



TRADE-OFF

- pred_proba = lr_clf.predict_proba(X_test)
- pred = lr_clf.predict(X_test)
- print('pred_proba()결과 Shape : {0}'.format(pred_proba.shape))
- print('pred_proba array에서 앞 3개만 샘플로 추출 \n:', pred_proba[:3])

- # 예측 확률 array와 예측 결과값 array를 병합(concatenate)해 예측 확률과 결과값을 한눈에 확인
- pred_proba_result = np.concatenate([pred_proba, pred.reshape(-1,1)], axis=1)
- print('두 개의 class 중에서 더 큰 확률을 클래스 값으로 예측 \n', pred_proba_result[:3])

```
pred_proba()결과 Shape : (179, 2)
pred_proba array에서 앞 3개만 샘플로 추출
: [[0.44935228 0.55064772]
 [0.86335513 0.13664487]
 [0.86429645 0.13570355]]
두 개의 class 중에서 더 큰 확률을 클래스 값으로 예측
 [[0.44935228 0.55064772 1.        ]
 [0.86335513 0.13664487 0.        ]
 [0.86429645 0.13570355 0.        ]]
```

	0 예측 확률	1 예측 확률	예측
0	0.449352	0.550648	1.0
1	0.863355	0.136645	0.0
2	0.864296	0.135704	0.0

TRADE-OFF

- from sklearn.preprocessing import Binarizer

- X = [[1,-1,2],
- [2, 0, 0],
- [0, 1.1, 1.2]]

- # X의 개별 원소들이 threshod값보다 같거나 작으면 0을, 크면 1을 반환
- binarizer = Binarizer(threshold=1.1)
- print(binarizer.fit_transform(X))

```
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

TRADE-OFF

DATA KUBWA

- from sklearn.preprocessing import Binarizer

- # Binarizer의 threshold 설정값. 분류 결정 임곗값임
- custom_threshold = 0.5

- # predict_proba() 반환값의 두 번째 칼럼, 즉 Positive 클래스 칼럼 하나만 추출해 Binarizer를 적용
- pred_proba_1 = pred_proba[:, 1].reshape(-1,1)

- binarizer = Binarizer(threshold=custom_threshold).fit(pred_proba_1)
- custom_predict = binarizer.transform(pred_proba_1)

- get_clf_eval(y_test, custom_predict)

```
오차 행렬
[[108  10]
 [ 14  47]]
정확도: 0.8659, 정밀도: 0.8246, 재현율: 0.7705
```

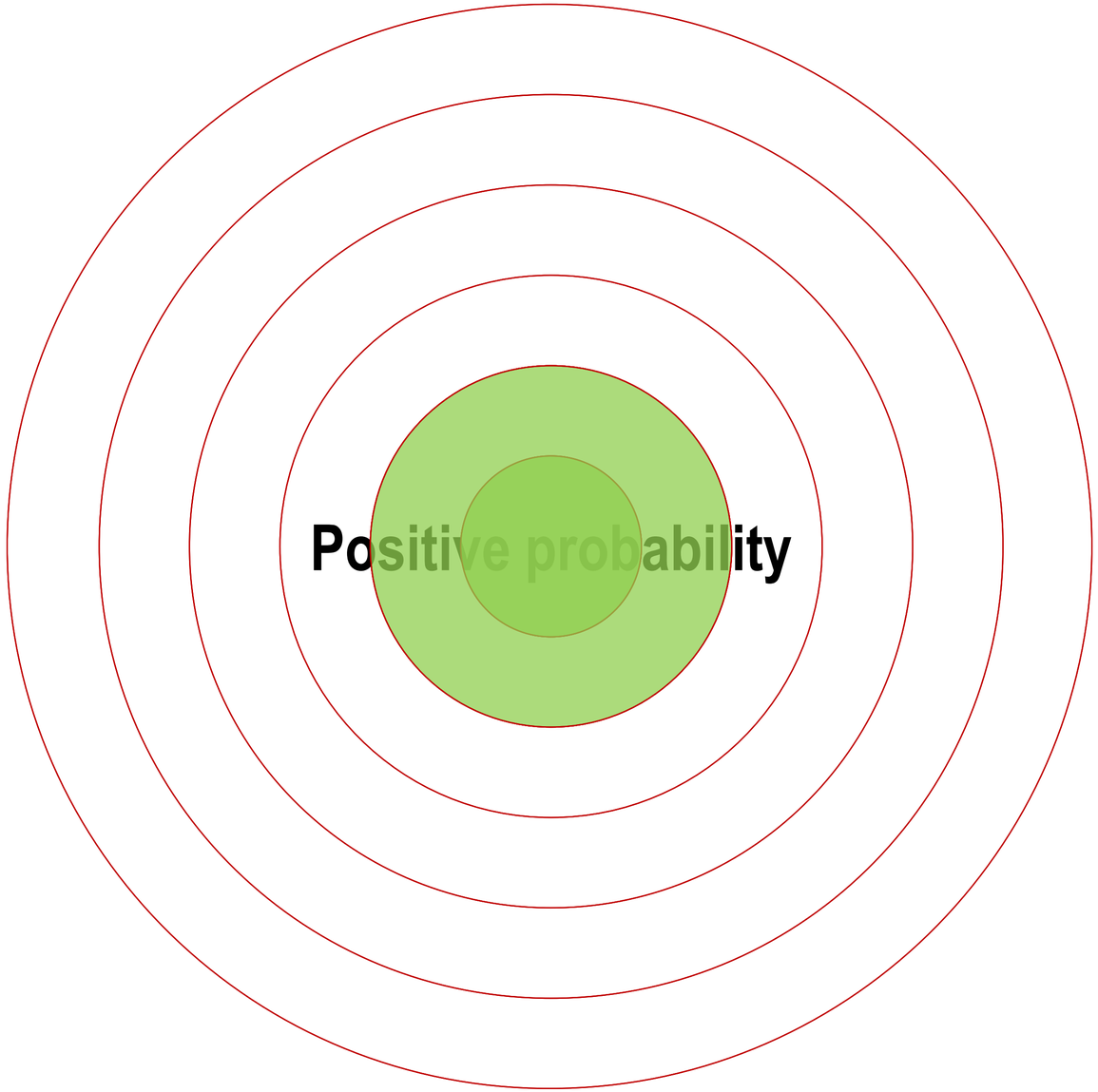
- # Binarizer의 threshold 설정값 0.5 → 0.4
- custom_threshold = 0.4
- pred_proba_1 = pred_proba[:, 1].reshape(-1,1)
- binarizer = Binarizer(threshold=custom_threshold).fit(pred_proba_1)
- custom_predict = binarizer.transform(pred_proba_1)

- get_clf_eval(y_test, custom_predict)

```
오차 행렬
[[97 21]
 [11 50]]
정확도: 0.8212, 정밀도: 0.7042, 재현율: 0.8197
```

TRADE-OFF

DATA KUBWA



- threshold = 0.5

TN 108	FP 10
FN 14	TP 47

- threshold = 0.4

TN 97	FP 21
FN 11	TP 50

TRADE-OFF

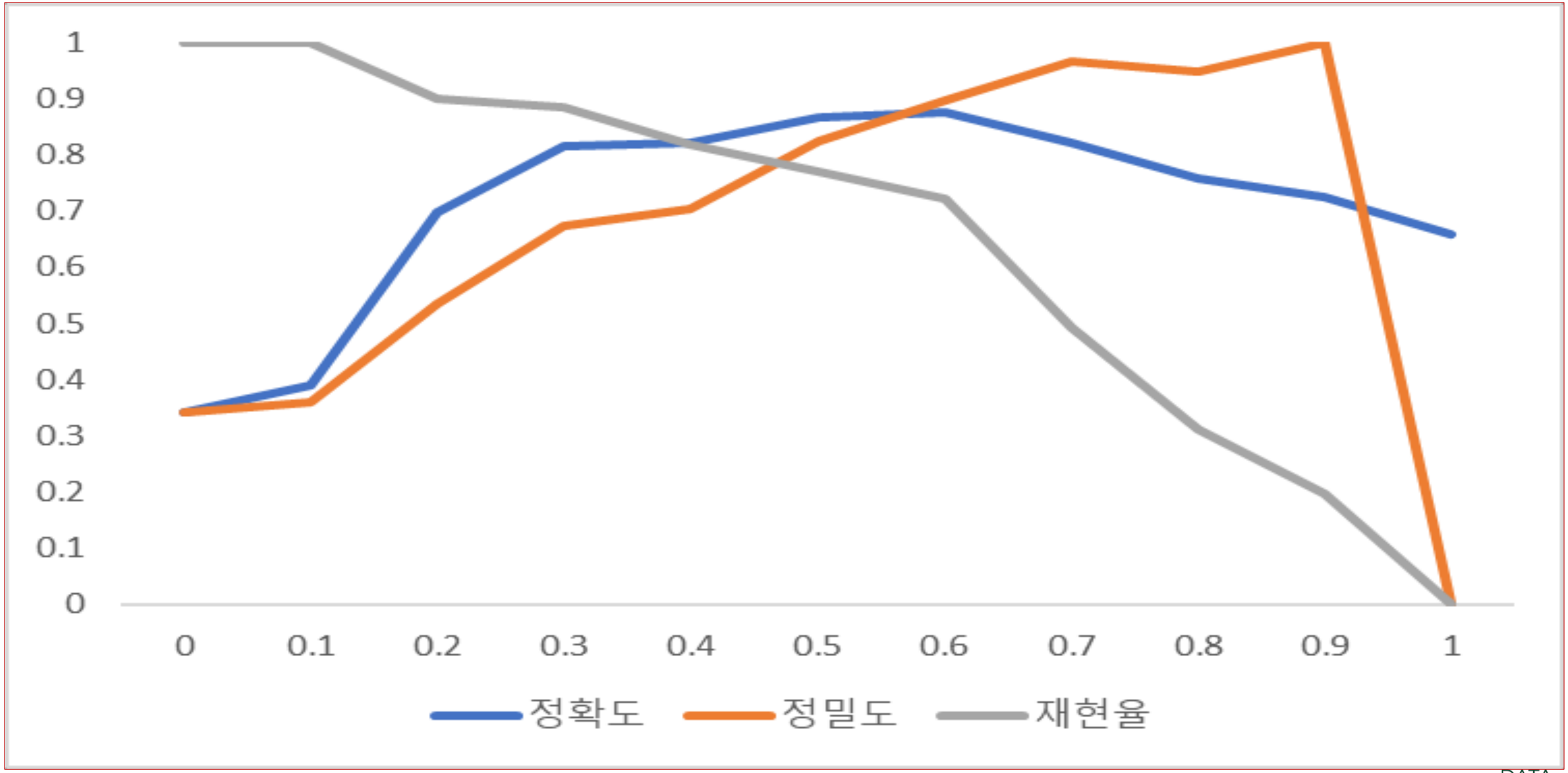
```
# 테스트 수행을 위한 모든 임계값을 리스트로 저장
thresholds = [0.4, 0.45, 0.5, 0.55, 0.6]

def get_eval_by_threshold(y_test, pred_proba_c1, thresholds):
    # thresholds list 객체 내의 값을 차례로 iteration하면서 Evaluation 수행.
    for custom_threshold in thresholds:
        binarizer = Binarizer(threshold=custom_threshold).fit(pred_proba_c1)
        custom_predict = binarizer.transform(pred_proba_c1)
        print('임계값:', custom_threshold)
        get_clf_eval(y_test, custom_predict)

get_eval_by_threshold(y_test, pred_proba[:,1].reshape(-1,1), thresholds)
```

임곗값	정확도	정밀도	재현율
0	0.3408	0.3408	1
0.1	0.3911	0.3588	1
0.2	0.6983	0.5340	0.9016
0.3	0.8156	0.6750	0.8852
0.4	0.8212	0.7042	0.8197
0.5	0.8659	0.8246	0.7705
0.6	0.8771	0.8980	0.7213
0.7	0.8212	0.9677	0.4918
0.8	0.7598	0.9500	0.3115
0.9	0.7263	1	0.1967
1.0	0.6592	0	0

TRADE-OFF



TRADE-OFF

```
from sklearn.metrics import precision_recall_curve

# 레이블 값이 1일 때의 예측 확률을 추출
pred_proba_class1 = lr_clf.predict_proba(X_test)[:, 1]

# 실제값 데이터 세트와 레이블 값이 1일 때 예측 확률을 precision_recall_curve 인자로 입력
precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_class1)
print('반환된 분류 결정 임곗값 배열의 shape:', thresholds.shape)

# 반환된 임곗값 배열 로우가 147건이므로 샘플로 10건만 추출하되, 임곗값을 15 step으로 추출
thr_index = np.arange(0, thresholds.shape[0], 15) # thresholds.shape = (147, )
print('샘플 추출을 위한 임계값 배열의 index 10개:', thr_index)
print('샘플용 10개의 임계값:', np.round(thresholds[thr_index], 2))

# 15 step 단위로 추출된 임계값에 따른 정밀도와 재현율 값
print('샘플 임계값별 정밀도:', np.round(precisions[thr_index], 3))
print('샘플 임계값별 재현율:', np.round(recalls[thr_index], 3))
```

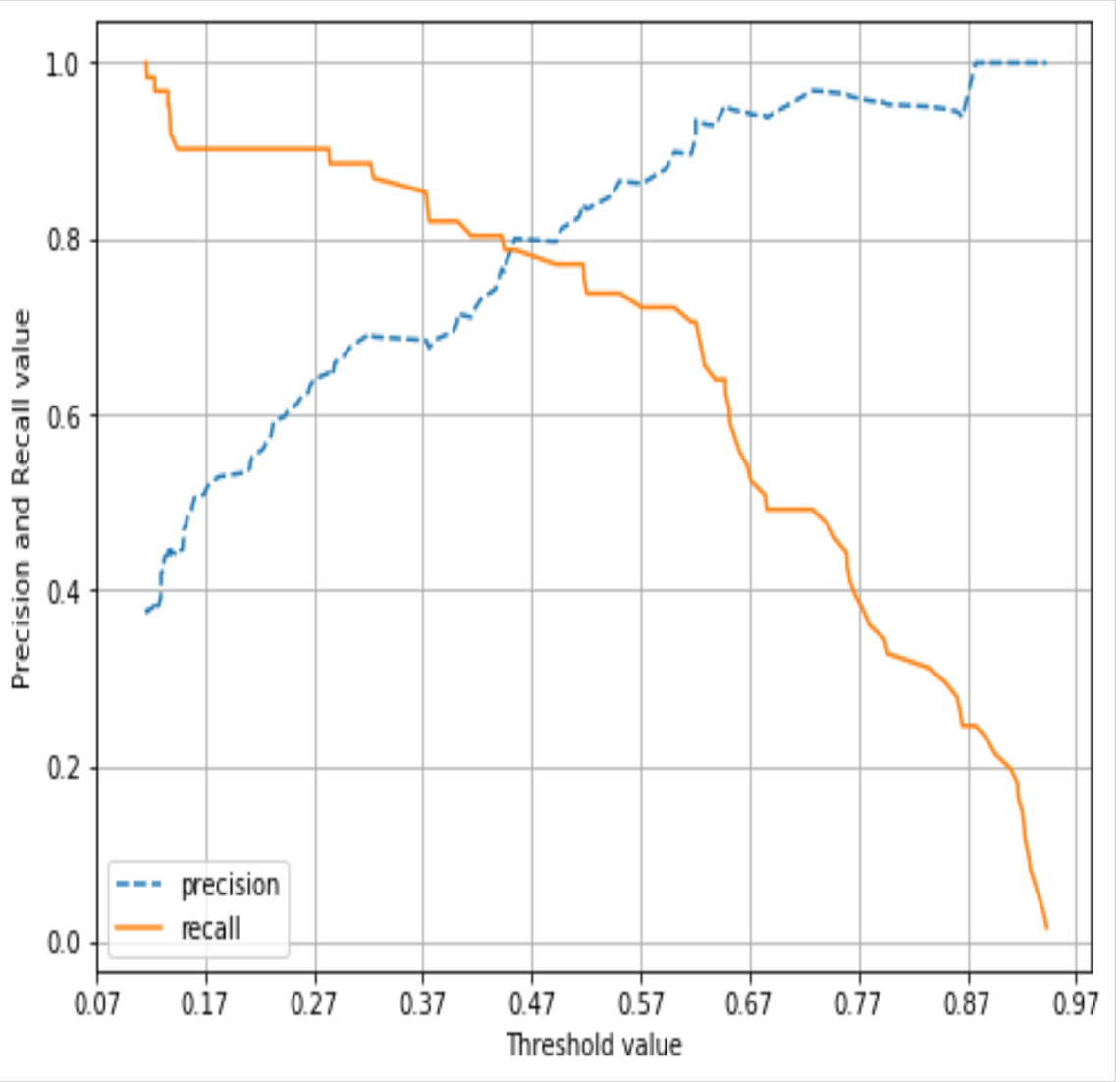
```
반환된 분류 결정 임곗값 배열의 shape: (147,)
샘플 추출을 위한 임계값 배열의 index 10개: [  0  15  30  45  60  75  90 105 120 135]
샘플용 10개의 임계값: [0.12 0.13 0.15 0.17 0.26 0.38 0.49 0.63 0.76 0.9 ]
샘플 임계값별 정밀도: [0.379 0.424 0.455 0.519 0.618 0.676 0.797 0.93  0.964 1.    ]
샘플 임계값별 재현율: [1.    0.967 0.902 0.902 0.902 0.82  0.77  0.656 0.443 0.213]
```

TRADE-OFF

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
%matplotlib inline

def precision_recall_curve_plot(y_test, pred_proba_c1):
    # threshold ndarray와 이 threshold에 따른 정밀도, 재현율 ndarray 추출.
    precision, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)
    # X 축을 threshold 값으로, Y 축은 정밀도, 재현율 값으로 각각 Plot 수행. 정밀도는 점선으로 표시
    plt.figure(figsize=(8,6))
    threshold_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    plt.plot(thresholds, recalls[0:threshold_boundary], label='recall')
    # threshold 값 X 축의 Scale을 0,1 단위로 변경
    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))
    # x축, y축 label과 legend, 그리고 grid 설정
    plt.xlabel('Threshold value'); plt.ylabel('Precision and Recall value')
    plt.legend(); plt.grid()
    plt.show()

precision_recall_curve_plot(y_test, lr_clf.predict_proba(X_test)[:,1])
```



❹ F1 스코어

Equation 3-3. F_1 score

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

To compute the F_1 score, simply call the `f1_score()` function:

	precision	recall	F1
A	0.9	0.1	0.18
B	0.5	0.5	0.5

❹ F1 스코어

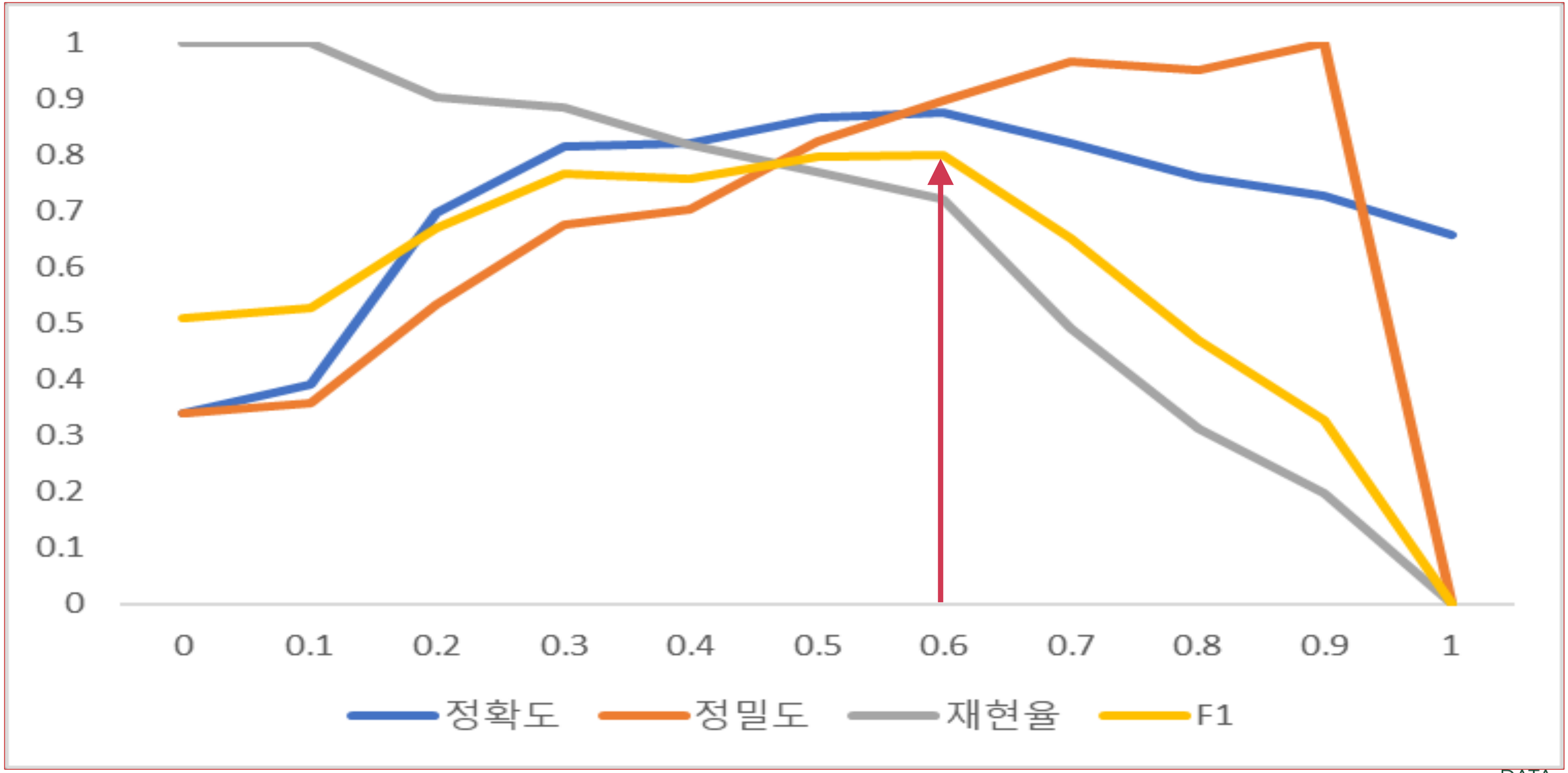
```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, pred)
print('F1 스코어: {0:.4f}'.format(f1))
F1 스코어: 0.7966

def get_clf_eval(y_test, pred):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)
    # F1 스코어 추가
    f1 = f1_score(y_test, pred)
    print('오차 행렬')
    print(confusion)
    # f1 score print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},
F1:{3:.4f}'.format(accuracy, precision, recall, f1))

thresholds = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
pred_prob = lr_clf.predict_proba(X_test)
get_eval_by_threshold(y_test, pred_proba[:, 1].reshape(-1,1),
thresholds)
```

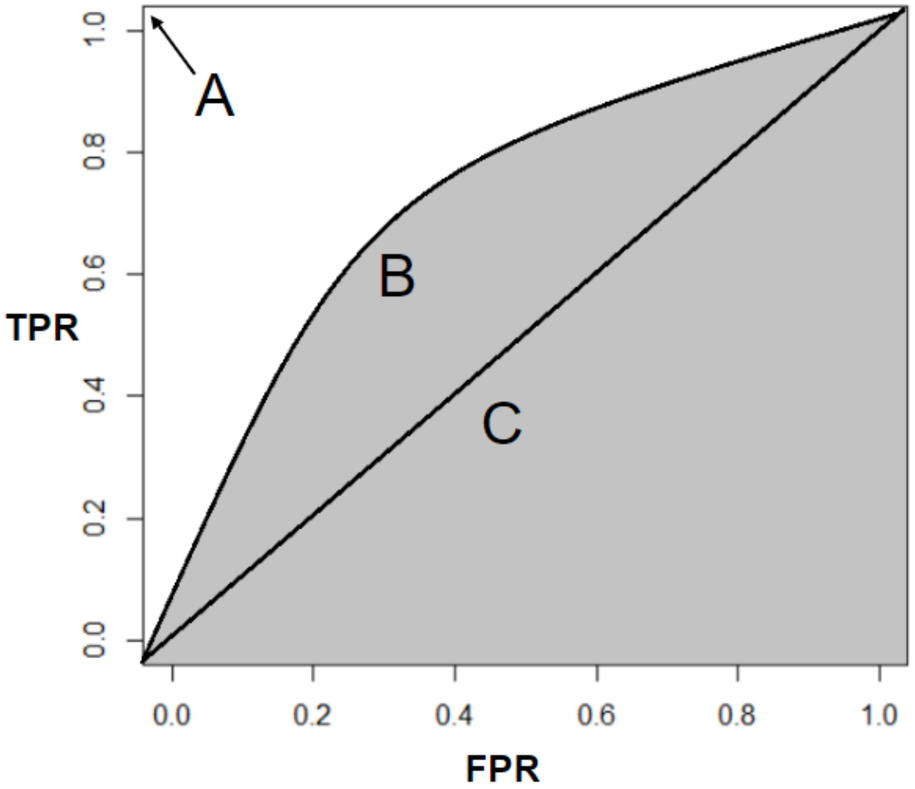
임곗값	정확도	정밀도	재현율	F1
0	0.3408	0.3408	1	0.5083
0.1	0.3911	0.3588	1	0.5281
0.2	0.6983	0.5340	0.9016	0.6707
0.3	0.8156	0.6750	0.8852	0.7660
0.4	0.8212	0.7042	0.8197	0.7576
0.5	0.8659	0.8246	0.7705	0.7966
0.6	0.8771	0.8980	0.7213	**0.8000**
0.7	0.8212	0.9677	0.4918	0.6522
0.8	0.7598	0.9500	0.3115	0.4691
0.9	0.7263	1	0.1967	0.3288
1.0	0.6592	0	0	0

F1 SCORE



❺ ROC AUC

- ROC: Receiver Operation Characteristic Curve, 수신자 판단 곡선
- 2차 대전 때 통신 장비 성능 평가를 위해 고안된 수치
- 의학 분야에서 많이 사용
- 머신러닝 이진 분류 모델의 예측 성능 판단에 중요한 평가 지표



A - 완벽한 모델

B - 좋은 모델

C - 평균/기본 모델

- 커브 아래 영역

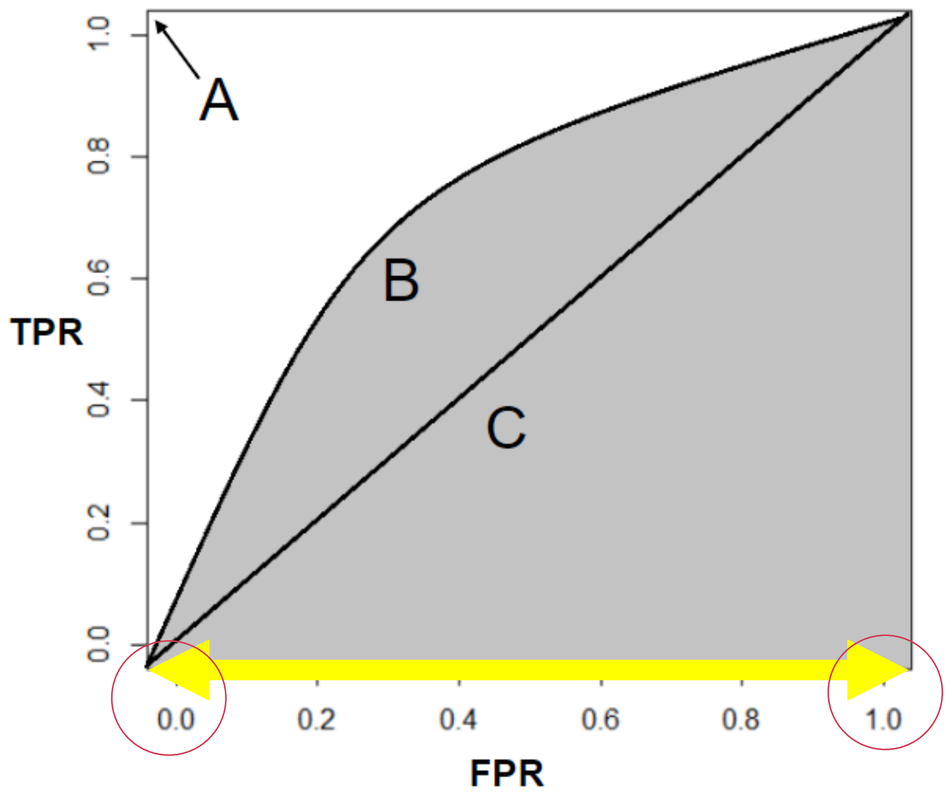
TPR = TP / (FN + TP), 재현율

TNR = TN / (FP + TN)

FPR = FP / (FP + TN)

FPR = 1 - TNR

❺ ROC AUC



- FPR → 0
  - 분류 결정 임곗값 → 1
  - FP / (FP + TN)
  - Positive 예측 기준이 Max
  - Positive가 틀릴 확률 0
  - Positive 선택 0
  - FP는 항상 0

- FPR → 1
  - FP / (FP + TN) → TN = 0 → FP/FP = 1
  - 분류 결정 임곗값 → 0
  - 모두 Positive로 예측
  - Negative 예측 확률 0이므로 TN도 0

❺ ROC AUC

```
from sklearn.metrics import roc_curve

# 레이블 값이 1일 때의 예측 확률을 추출
pred_prob_class1 = lr_clf.predict_proba(X_test)[:,1]

fprs, tprs, thresholds = roc_curve(y_test, pred_prob_class1)
# 반환된 임곗값 배열 로우가 47건이므로 샘플로 10건만 추출하되, 임곗값을 5 Step으로 추출
thr_index = np.arange(0, thresholds.shape[0], 5)
print('샘플 추출을 위한 임곗값 배열의 index 10개:', thr_index)
print('샘플용 10개의 임곗값:', np.round(thresholds[thr_index], 2))

# 5 step 단위로 추출된 임곗값에 따른 FPR, TPR 값
print('샘플 임곗값별 FPR:', np.round(fprs[thr_index], 3))
print('샘플 임곗값별 TPR:', np.round(tprs[thr_index], 3))
```

```
샘플 추출을 위한 임곗값 배열의 index 10개: [ 0  5 10 15 20 25 30 35 40 45]
샘플용 10개의 임곗값: [1.94 0.87 0.63 0.55 0.44 0.32 0.15 0.14 0.13 0.12]
샘플 임곗값별 FPR: [0.    0.008 0.025 0.059 0.127 0.203 0.559 0.602 0.695 0.847]
샘플 임곗값별 TPR: [0.    0.246 0.672 0.738 0.787 0.885 0.902 0.951 0.967 0.984]
```

❺ ROC AUC

```
from sklearn.metrics import roc_curve

# 레이블 값이 1일 때의 예측 확률을 추출
pred_prob_class1 = lr_clf.predict_proba(X_test)[:,1]

fprs, tprs, thresholds = roc_curve(y_test, pred_prob_class1)
# 반환된 임곗값 배열 로우가 47건 추출
thr_index = np.arange(0, thresholds.shape[0])
print('임곗값 배열의 index:', thr_index)
print('전체 임곗값:', np.round(thresholds[thr_index], 2))

# 추출된 임곗값에 따른 FPR, TPR 값
print('전체 임곗값별 FPR:', np.round(fprs[thr_index], 3))
print('전체 임곗값별 TPR:', np.round(tprs[thr_index], 3))
```

❺ ROC AUC

#	임계값	FPR	TPR
47	0.02	1	1
46	0.12	0.85	1
45	0.12	0.85	0.98
44	0.12	0.82	0.98
43	0.12	0.82	0.97
42	0.13	0.78	0.97
41	0.13	0.75	0.97
40	0.13	0.7	0.97
39	0.13	0.68	0.97
38	0.13	0.66	0.97
37	0.14	0.62	0.97
36	0.14	0.61	0.95
35	0.14	0.6	0.95
34	0.14	0.6	0.92
33	0.14	0.59	0.92
32	0.14	0.59	0.9
31	0.15	0.58	0.9
30	0.15	0.56	0.9
29	0.23	0.35	0.9
28	0.23	0.32	0.9
27	0.28	0.25	0.9
26	0.28	0.25	0.89
25	0.32	0.2	0.89
24	0.38	0.2	0.82
23	0.4	0.17	0.82
22	0.41	0.17	0.8
21	0.44	0.13	0.8
20	0.44	0.13	0.79
19	0.45	0.1	0.79
18	0.49	0.1	0.77
17	0.52	0.08	0.77
16	0.52	0.08	0.74
15	0.55	0.06	0.74
14	0.57	0.06	0.72
13	0.6	0.04	0.72
12	0.62	0.04	0.71
11	0.62	0.03	0.71
10	0.63	0.03	0.67
9	0.64	0.03	0.64
8	0.65	0.02	0.64
7	0.69	0.02	0.49
6	0.73	0.01	0.49
5	0.87	0.01	0.25
4	0.88	0	0.25
3	0.93	0	0.07
2	0.94	0	0.03
1	0.94	0	0.02
0	1.94	0	0

0

1

DATA KUBWA

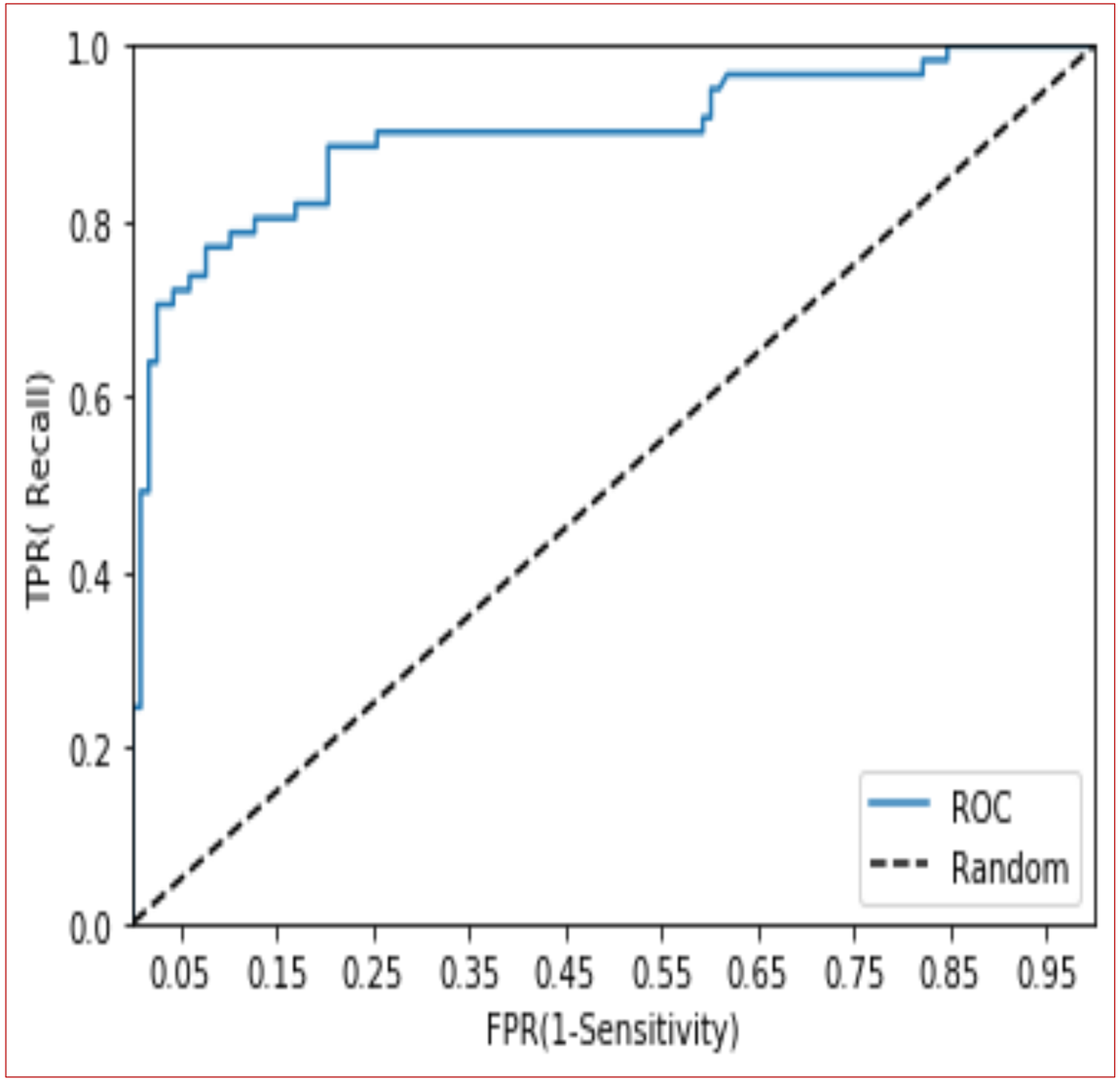
❺ ROC AUC

DATA KUBWA

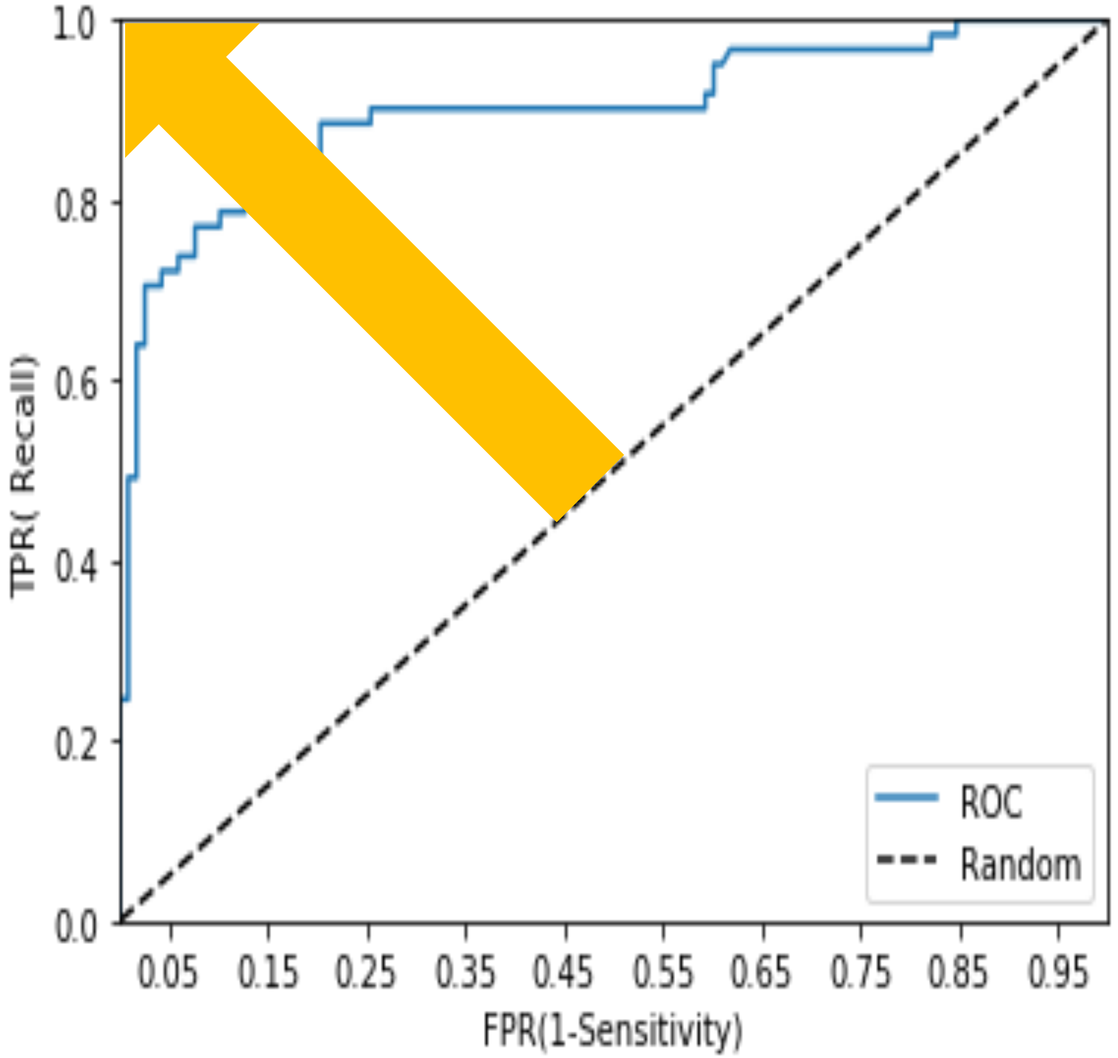
```
def roc_curve_plot(y_test, pred_prob_c1):
    # 임곗값에 따른 FPR, TPR 값을 반환받음
    fprs, tprs, thresholds = roc_curve(y_test, pred_proba_class1)
    # ROC 곡선을 그래프 곡선으로 그림
    plt.plot(fprs, tprs, label='ROC')
    # 가운데 대각선 직선을 그림
    plt.plot([0,1], [0,1], 'k--', label='Random')

    # FPR X 축의 Scale을 0.1 단위로 변경, X, Y축 명 설정 등
    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))
    plt.xlim(0,1); plt.ylim(0,1)
    plt.xlabel('FPR(1-Sensitivity)'); plt.ylabel('TPR( Recall)')
    plt.legend()

roc_curve_plot(y_test, pred_proba[:,1])
```



❺ ROC AUC



- AUC
  - Area Under Curve
  - ROC 곡선 밑의 면적
  - 1에 가까울 수록 좋은 수치
  - FPR이 작은 상태에서 TPR이 커지는 것
  - 가운데 대각선은 동전 던지기 수준의 AUC 값
  - 보통 분류는 0.5 이상의 AUC 값

❺ ROC AUC

- from sklearn.metrics import roc_auc_score

- pred = lr_clf.predict(X_test)
- roc_score = roc_auc_score(y_test, pred)
- print('ROC AUC 값: {0:.4f}'.format(roc_score))

- ROC AUC 값: 0.8429

- 타이타닉 생존자 예측 Logistic Regression model의 ROC AUC 값: 0.842

https://www.kaggle.com/uciml/pima-Indians-diabetes-database

- Pregnancies: 임신 횟수
- Glucose: 포도당 부하 검사 수치
- BloodPressure: 혈압
- SkinThickness: 팔 삼두근 뒤쪽의 피하지방 측정값
- Insulin: 혈청 인슐린
- BMI: 체질량지수 (체중/키)
- DiabetesPedigreeFunction: 당뇨 내력 가중치 값
- Age: 나이
- Outcome: 클래스 결정 값 (레이블 칼럼)