**📍 util 코드분석**

1. 단일 성능 지표만 보면 절대 안 된다

* Accuracy(정확도) 하나만 보면 모델을 오해할 수 있어.
* **이 코드는 Precision, Recall, F1, MCC 같이 다양한 지표를 함께 본다.**
* 특히 MCC는 데이터가 "불균형"할 때(예: 환자가 90%, 비환자가 10%) 좋은 지표야.

💡 인사이트:  
여러 지표를 종합적으로 보지 않으면 "진짜 좋은 모델"을 찾을 수 없다.

2. 모델을 한 번 학습해서 평가하는 것은 "과적합 위험"이 있다

* 데이터를 나누는 방법(K-Fold)을 써서, 모델이 데이터에 "우연히 잘 맞는" 것을 방지한다.
* **Fold를 나눠서 여러 번 테스트해보고**, 평균을 내서 평가하는 것이 훨씬 신뢰도가 높다.

💡 인사이트:  
단 한 번의 train/test split로 모델 성능을 평가하면 운에 의존하는 결과가 나온다.  
여러 번 나누고 학습해야 함

3. 학습과 예측 과정은 항상 일관성 있게 관리해야 한다

* 이 코드는 매번 fold를 나눌 때마다, 같은 방식으로 모델을 새로 만들고(model\_func) 학습한다.
* deepcopy로 모델 복제해서 저장하는 것도, 모델이 훈련하면서 내부 상태를 바꿀 수 있으니까 안정성을 확보하는 방법이야.

💡 인사이트:  
실험할 때는 "모델을 매번 새로 생성해서" 깨끗한 상태에서 학습시켜야 한다.  
한 번 학습한 모델을 다시 써버리면 실험 결과가 더럽혀질 수 있다.

4. 테스트 데이터는 모델 개발 단계에서 절대 건드리면 안 된다

* 이 코드도 보면, 테스트 데이터(x\_test, y\_test)는 맨 마지막에만 평가에 사용하지, 학습 중에는 절대 보지 않아.
* 학습하는 동안은 오직 "train"과 "validation"만 사용.

💡 인사이트:  
테스트 데이터는 "미래를 대변"하는 데이터야.  
모델을 만들고 고치는 동안 절대 테스트 데이터 성능을 보고 조정하면 안 된다.

5. 실험 결과를 항상 "Fold 별로" 저장하고 관리해야 한다

* 이 코드는 fold마다 valid/test 성능을 리스트로 모아둬(defaultdict(list)).
* Fold 별로 점수 분포를 보면, 성능이 들쑥날쑥한지, 일관성 있는지 확인할 수 있어.

💡 인사이트:  
평균 점수만 보는 게 아니라 Fold 별 분산을 보면 모델이 얼마나 "안정적인지" 판단할 수 있다.

★ 전체 요약

"여러 성능 지표 사용 + K-Fold 검증 + 일관성 있는 실험 설계 가 필수적이라는 걸

해당 코드 분석을 통해 알게됨 “