📌 `sklearn.ensemble` 주요 클래스들

✅ 투표 / 스태킹

- `VotingClassifier` / `VotingRegressor`
  - 여러 다른 모델을 학습시켜서 예측을 "투표(분류)" 또는 "평균(회귀)"으로 합침
- `StackingClassifier` / `StackingRegressor`
  - 여러 모델의 예측 결과를 다시 "메타 모델"이 학습 → 투표보다 더 유연

✅ 배깅 계열 (Bootstrap Aggregating)

- `BaggingClassifier` / `BaggingRegressor`
  - 기본 추정기(estimator)를 데이터 샘플을 부트스트랩으로 여러 번 학습 → 평균/투표
- `RandomForestClassifier` / `RandomForestRegressor`
  - Bagging + 특성 무작위 선택까지 적용
  - 결정트리 기반, 가장 많이 쓰임

✅ 부스팅 계열 (Boosting)

- `AdaBoostClassifier` / `AdaBoostRegressor`
  - 이전 모델이 틀린 샘플에 가중치 ↑, 틀리기 어려운 방향으로 다음 학습기 훈련
- `GradientBoostingClassifier` / `GradientBoostingRegressor`
  - 손실 함수의 **잔차(gradient)**를 다음 모델이 계속 보정
- `HistGradientBoostingClassifier` / `HistGradientBoostingRegressor`
  - Gradient Boosting을 히스토그램 기반으로 최적화 → 대규모 데이터에서도 빠름 (XGBoost와 유사)

✅ 이외

- `IsolationForest`
  - 이상치 탐지(outlier detection)용 앙상블
- `ExtraTreesClassifier` / `ExtraTreesRegressor`
  - 랜덤포레스트와 비슷하지만, 분기 시 **더 무작위(random split)** → 더 빠르고 variance ↓

📌 1. VotingClassifier

(1) Hard voting

- 각 분류기 `h₁(x), h₂(x), …, hₖ(x)`의 `predict` 결과(0/1)를 모아서 다수결

$$\hat{y} = \text{mode}\{h_j(x)\}_{j=1}^{k}$$

(2) Soft voting (확률 기반, 일반적으로 성능이 더 좋음)

- 각 분류기 확률 예측 $p_j(y=1|x)$을 평균해서 최종 결정

$$\hat{p}(y=1|x) = \frac{1}{k}\sum_{j=1}^{k} p_j(y=1|x)$$

- 마지막 의사결정은 그냥 평균 확률 → argmax로 클래스 선택
- **학습 과정은 없고, 단순 집계만 함**

📌 2. StackingClassifier

- Base 모델들 각각이 예측한 확률값들을 **새로운 입력 피처 X_{meta}**로 사용
- $X_{meta} \in \mathbb{R}^{k \times C}$ (k=base 모델 개수, C=클래스 수)
- 이 X_{meta}를 가지고 **메타 모델을 학습**

즉,

$$X_{meta}(i) = \left[p_1(y=0|x_i), \ldots, p_1(y=C-1|x_i), \ldots, p_k(y=C-1|x_i)\right]$$

메타 모델(예: Logistic Regression)은 결국

$$\hat{y} = \arg\max_{y} \; g_\theta(X_{meta}(i))$$

여기서 g_θ는 메타 모델이고, MLE(Maximum Likelihood Estimation) 방식으로 파라미터 θ를 학습합니다.

In [1]:
```python
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42)

stack = StackingClassifier(
    estimators=[
        ("svm", SVC(probability=True, random_state=42)),
        ("dt", DecisionTreeClassifier(max_depth=3, random_state=42)),
    ],
    final_estimator=LogisticRegression(max_iter=1000)
)

stack.fit(X_train, y_train)

# 학습된 개별 모델
print(stack.estimators_)
print(stack.estimators_[0].predict(X_test[:5]))  # SVM 예측
print(stack.estimators_[1].predict(X_test[:5]))  # DT 예측
```

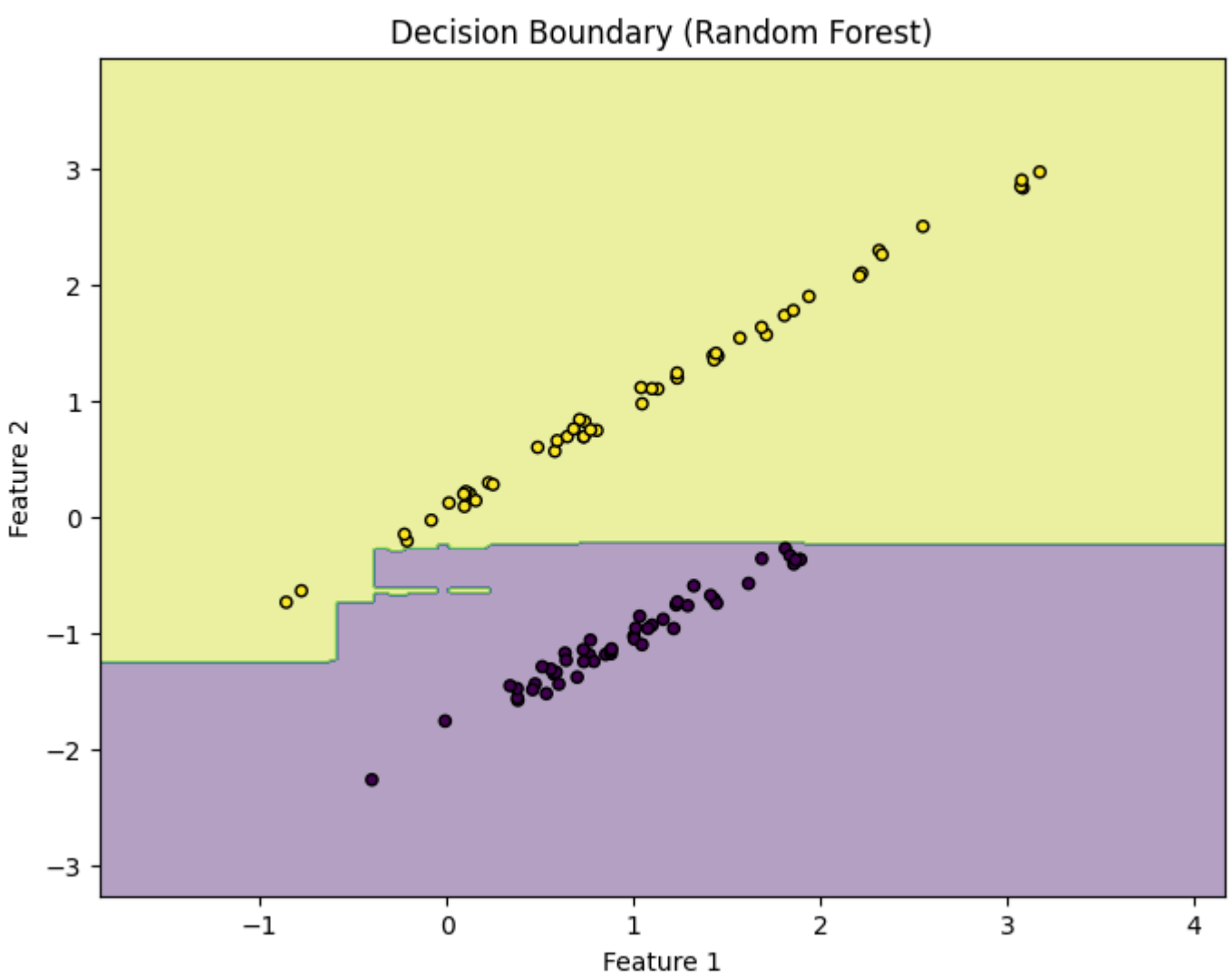
```
[SVC(probability=True, random_state=42), DecisionTreeClassifier(max_depth=3, random_state=42)]
[0 1 1 1 0]
[0 1 1 1 0]
```

In [2]:
```python
#📌 2. transform()으로 중간 예측값(메타 입력값) 확인
# StackingClassifier는 내부적으로 base estimator들의 예측 확률(또는 결정함수)을 최종 모델 입력으로 씁니다.
# 이건 .transform()으로 확인할 수 있습니다.
#  base 모델들의 예측값 → 메타모델 입력
Z = stack.transform(X_test[:5])
print("Meta input shape:", Z.shape)
print("Meta input values:\n", Z)
print("Final estimator:", stack.final_estimator_)
```

```
Meta input shape: (5, 6)
Meta input values:
 [[0.96957051 0.01966696 0.01076253 1.         0.         0.        ]
 [0.01170224 0.78977073 0.19852703 0.         1.         0.        ]
 [0.01204974 0.97691122 0.01103904 0.         1.         0.        ]
 [0.00959255 0.97951662 0.01089083 0.         1.         0.        ]
 [0.96381253 0.02447783 0.01170964 1.         0.         0.        ]]
Final estimator: LogisticRegression(max_iter=1000)
```

In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=42

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X, y)

# 예측 경계선 시각화
plt.figure(figsize=(8, 6))

# Plotting decision regions
h = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = rf_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4)

# Plotting data points
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Boundary (Random Forest)')
plt.show()
```


Decision Boundary (Random Forest)
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−1
0
1
2
3
4