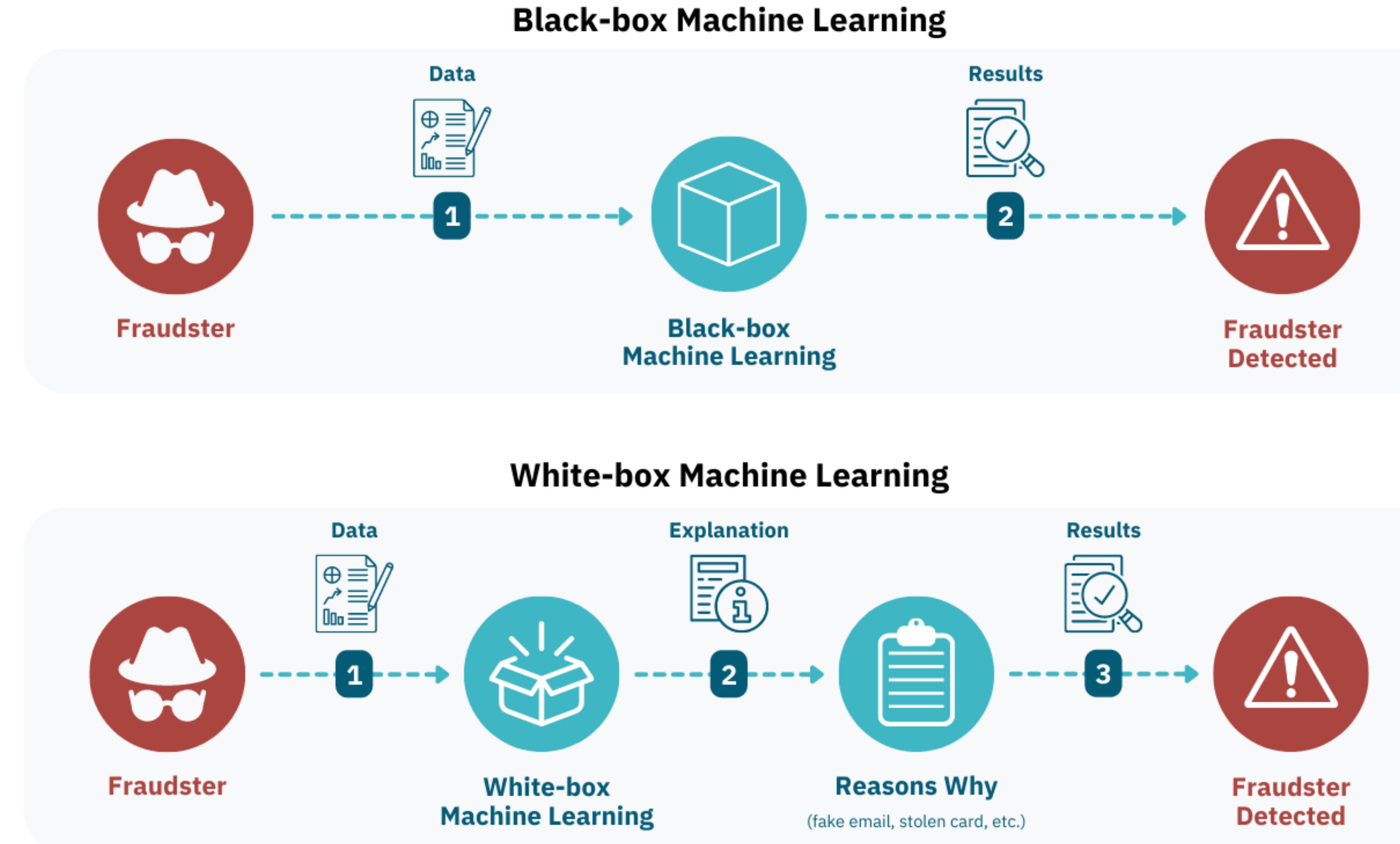


Chapter 11. Model interpretation

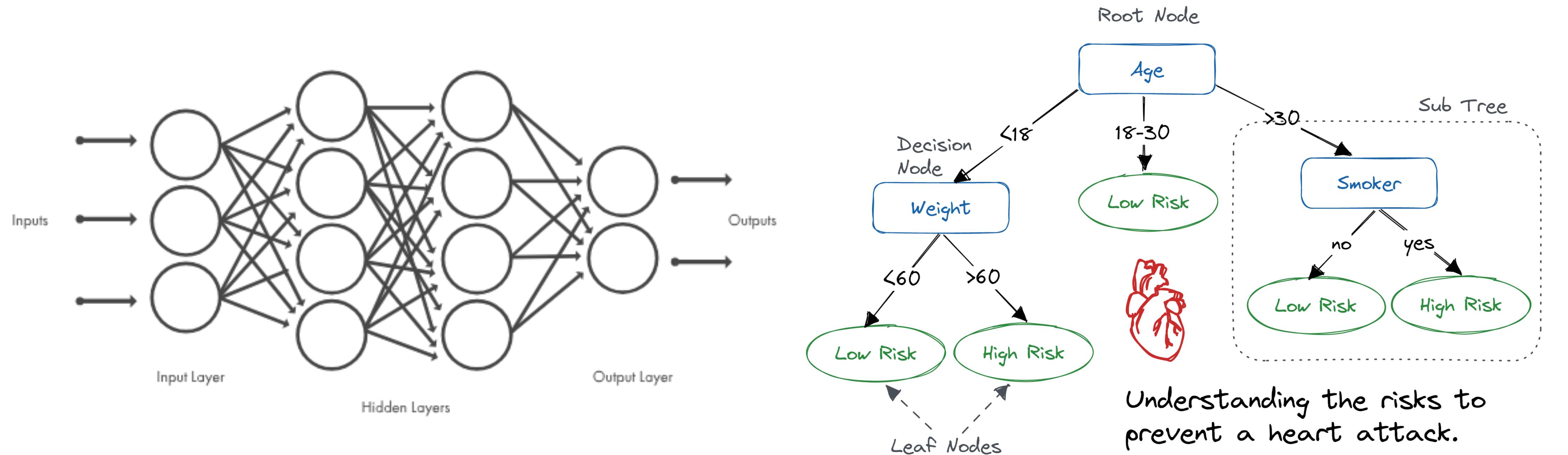
Blackbox & Whitebox model



Feature importance

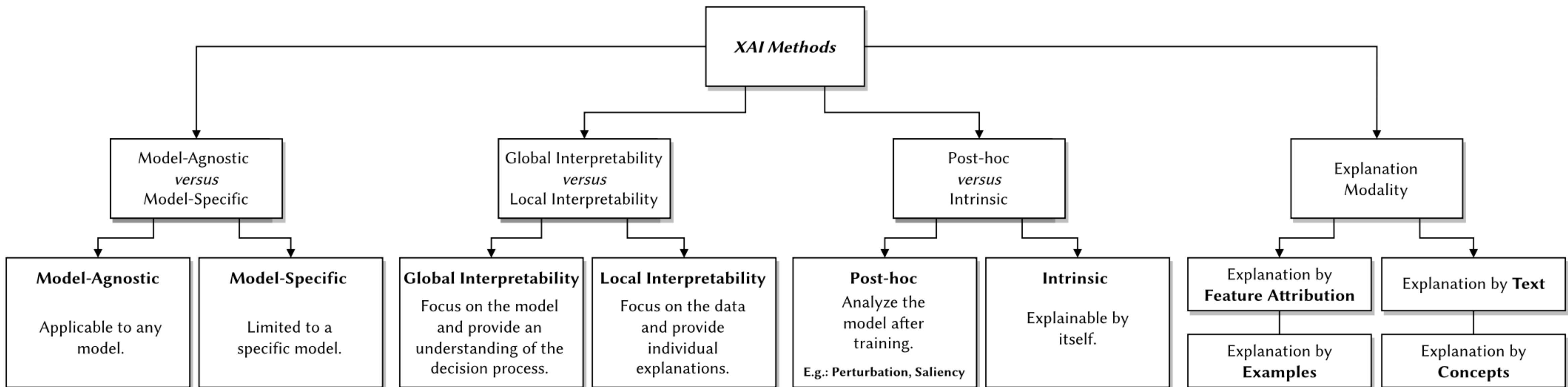
- Feature importance란?
 - 모델이 예측을 수행할 때 각 피처가 얼마나 중요한 역할을 했는지를 나타내는 지표
- 중요성
 - 모델 결과에 대한 해석력 향상
 - 피쳐 선택에 도움
 - 도메인 지식과의 연결 고리

다양한 ML 모델



- 모델에 따라서 feature importance를 측정하기 위한 방법이 달라질 수 있음.

XAI Method 구분



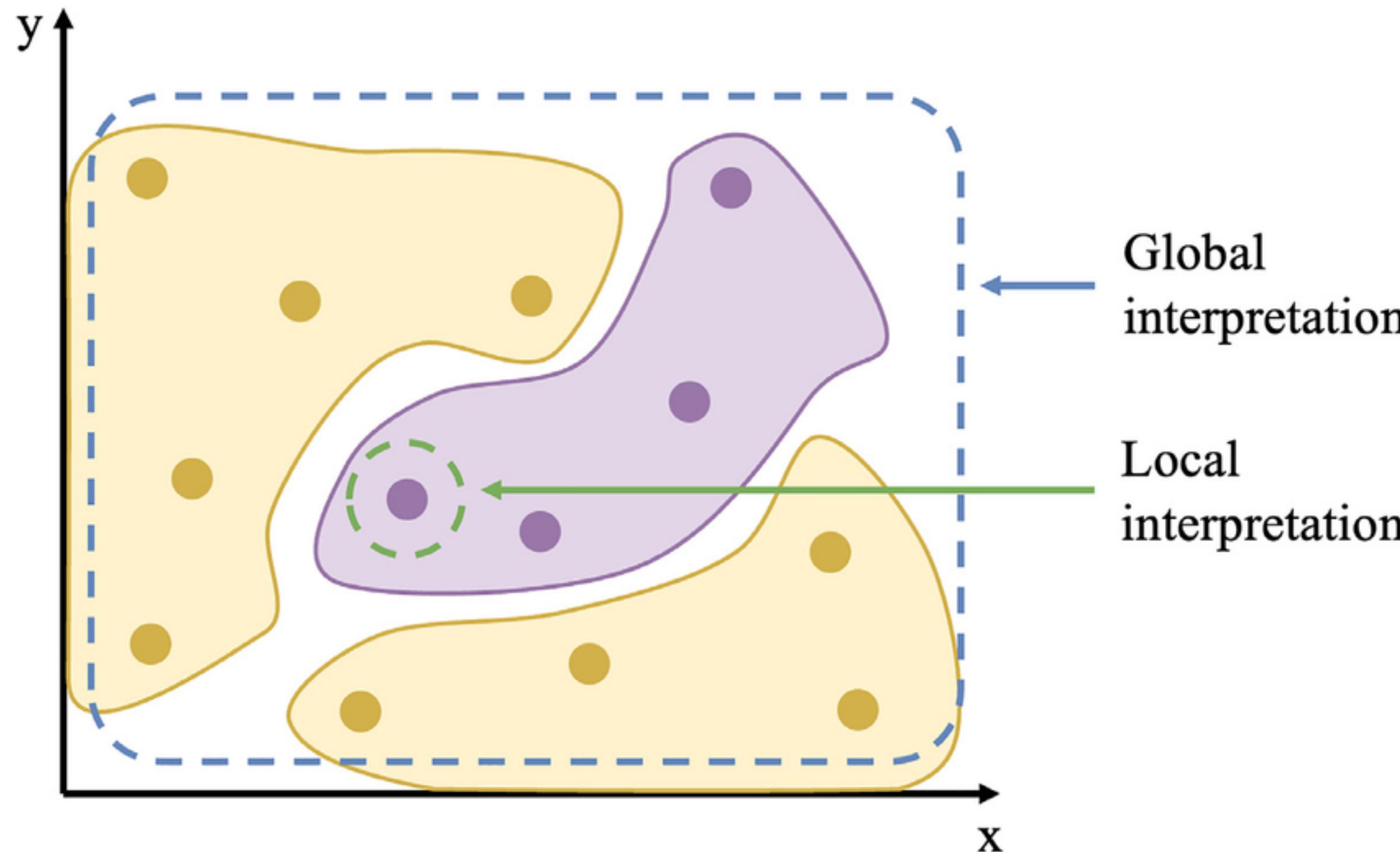
Model dependency

- **Model Specific method :**
 - 모델 내부 로직(구조, 파라미터)을 분석
 - 해석 대상 모델에 따라 설명 방식이 제한됨
- 예시
 - 선형회귀모델 : 회귀 계수
 - 트리기반 모델 : 트리 시각화, Information gain 등

Model dependency

- **Model Agnostic method :**
 - 모델 내부 구조와 무관하게 입력과 출력만을 이용해 설명하는 기법
 - 어떤 모델에도 적용 가능 (신경망, 앙상블, 트리, etc.)
 - 일반화된 해석 도구 제공
- 예시
 - SHAP, LIME, Permutation importance 등

Interpretation scope



Interpretation scope

- **Global interpretability**
 - 모델 전체의 작동 방식과 전체 데이터에 대한 일반적인 의사결정 패턴을 설명
- 목적
 - 전체적으로 어떤 피쳐가 중요한지
 - 모델이 어떤 방식으로 예측을 구성하는지
 - 피쳐와 타겟 간의 전반적인 관계

Interpretation scope

- Local interpretability
 - 개별 예측값(1개의 샘플)이 왜 그렇게 나왔는지를 설명
 - 즉, 특정 입력에 대해 모델이 왜 그 출력을 냈는지
- 목표
 - 개별 사례의 예측 근거 파악
 - 예측 결과를 사용자의 관점에서 납득하게 만듦

Computation scope

- **Intrinsic**
 - 모델이 학습 과정 중 내재된 계산 로직에 의해 결정 되는 피처 중요도
- 특징
 - 모델 내부 정보 활용 (예: 분기, 가중치, 계수)
 - 계산이 빠르고 직관적
 - 일반적으로 model-specific 기법에 해당
 - feature 스케일이나 상관관계에 따라 영향 받을 수 있음

Computation scope

- **Post-Hoc**
 - 모델 학습이 끝난 후, 예측 결과로부터 피처의 중요도를 추정하는 방식
- 특징
 - 일반적으로 post-hoc 기법은 모델에 무관(model-agnostic)하게 사용 가능
 - 예측값 변화량을 기반으로 중요도 측정
 - 블랙박스 모델에도 적용 가능
 - 계산 비용이 클 수 있음

선형 및 로지스틱 회귀

- 선형 모델의 기본구조
- $y = ax_1 + bx_2 + cx_3 + \cdots + kx_k$
- 회귀계수(coefficient) : a, b, c, d, \dots
- Feature importance : 회귀계수의 절대값 활용
- Why? coefficient \times variable 값만큼 결과값에 영향을 미치기 때문에

Preprocessing

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

mean_imputer = SimpleImputer(strategy='mean')
df['Age'] = mean_imputer.fit_transform(df[['Age']])

label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df['Pclass'] = label_encoder.fit_transform(df['Pclass'])

mode_imputer = SimpleImputer(strategy='most_frequent')
df[['Embarked']] = mode_imputer.fit_transform(df[['Embarked']])
df['Embarked'] = label_encoder.fit_transform(df['Embarked'])

df_ = df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)
df_X = df_.drop('Survived', axis=1)
df_y = df_.Survived
```

Logistic regression 모델 학습

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
import numpy as np

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_X)

# 모델 학습
model = LogisticRegression()
model.fit(X_scaled, df_y)
```

Coefficient를 통한 피쳐 중요도

```
# 회귀 계수 추출
feature_names = df_X.columns
coefficients = model.coef_[0]

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients,
    'AbsImportance': np.abs(coefficients)
}).sort_values(by='AbsImportance', ascending=False)

print(importance_df)
```

결과해석

	Feature	Coefficient	AbsImportance
1	Sex	-1.294455	1.294455
0	Pclass	-0.889791	0.889791
2	Age	-0.500663	0.500663
3	SibSp	-0.354175	0.354175
6	Embarked	-0.174141	0.174141
5	Fare	0.102073	0.102073
4	Parch	-0.079872	0.079872

Decision tree

- Decision tree의 분할 규칙: Gini 불순도, 정보이득 등에 기반
- 특정 변수의 특정 값을 기준으로 데이터를 분할했을 때 불확실성이 감소하는 방향
- Sklearn 기준 Decision tree classifier 내장 (Gini importance)
- feature_importances_

Gini impurity

- $GINI(S) = 1 - \sum p_k^2$
- p_k 는 집합 S 내의 k 클래스의 비율 (k 클래스 샘플 수 / 전체 샘플 수)
- e.g.,
 - $S = \{1,1,1,1,1,1\} \gg p_1 = 1, p_2 = 0$ 일 때 $GINI(S) = 0$
 - $S = \{1,1,1,2,2,2\} \gg p_1 = 0.5, p_2 = 0.5$ 일 때 $GINI(S) = 0.5$

Decision tree

```
from sklearn.tree import DecisionTreeClassifier

# 모델 학습
model = DecisionTreeClassifier()
model.fit(df_X, df_y)

# Feature importance 추출
importances = model.feature_importances_
feature_names = df_X.columns

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print(importance_df)
```

	Feature	Importance
1	Sex	0.309335
2	Age	0.255551
5	Fare	0.235299
0	Pclass	0.105872
3	SibSp	0.052344
4	Parch	0.026535
6	Embarked	0.015063

Random Forest

```
from sklearn.ensemble import RandomForestClassifier  
  
#모델 학습  
model = RandomForestClassifier(n_estimators=100, random_state=42)  
model.fit(df_X, df_y)  
  
# Feature importance 추출  
importances = model.feature_importances_  
feature_names = df_X.columns  
  
importance_df = pd.DataFrame({  
    'Feature': feature_names,  
    'Importance': importances  
}).sort_values(by='Importance', ascending=False)  
  
print(importance_df)
```

	Feature	Importance
5	Fare	0.269896
1	Sex	0.261793
2	Age	0.261520
0	Pclass	0.088197
3	SibSp	0.047071
4	Parch	0.038157
6	Embarked	0.033367

개별 트리의 feature importance 평균

Permutation importance

- 모델 추론 과정에서 특정 feature의 값을 바꿔버린다면?
- 특정한 feature의 내용을 바꿨을 때 모델의 성능이 많이 하락하게되는 경우
- 해당 feature를 중요한 feature로 간주할 수 있음.
- Post-Hoc >> validation / test set에 대해 계산

Permutation importance

- 성능평가 지표 선정
- 기준 성능 계산
- 각 feature 별로 해당 피처의 값만 무작위로 섞어 새로운 데이터셋 생성
- 특정 Feature 만 permutation 된 데이터셋에 대해 성능하락량 계산
- 이를 n회 반복하여 평균치를 계산하여 성능하락량이 큰 경우 중요 feature로 간주

Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...
156	142	...	8
153	130	...	24

Permutation importance

```
from sklearn.model_selection import train_test_split
from sklearn.inspection import permutation_importance

#데이터셋 분할
X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, random_state=42)

# 모델 학습
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Permutation Importance 계산
result = permutation_importance(model, X_test, y_test, n_repeats=20, random_state=42)

# 결과 정리
importance_df = pd.DataFrame({
    'Feature': df_X.columns,
    'MeanImportance': result.importances_mean,
    'StdImportance': result.importances_std
}).sort_values(by='MeanImportance', ascending=False)

print(importance_df)
```

	Feature	MeanImportance	StdImportance
1	Sex	0.158969	0.029863
0	Pclass	0.064126	0.020603
6	Embarked	0.017937	0.020104
2	Age	0.011435	0.016011
3	SibSp	0.009865	0.012572
5	Fare	0.007848	0.014243
4	Parch	-0.011883	0.010543

Local interpretation

- 주어진 입력 샘플 x 에 대해서 왜 y 가 계산되었는지 해석
- 대표적으로 LIME, SHAP 등의 방법론

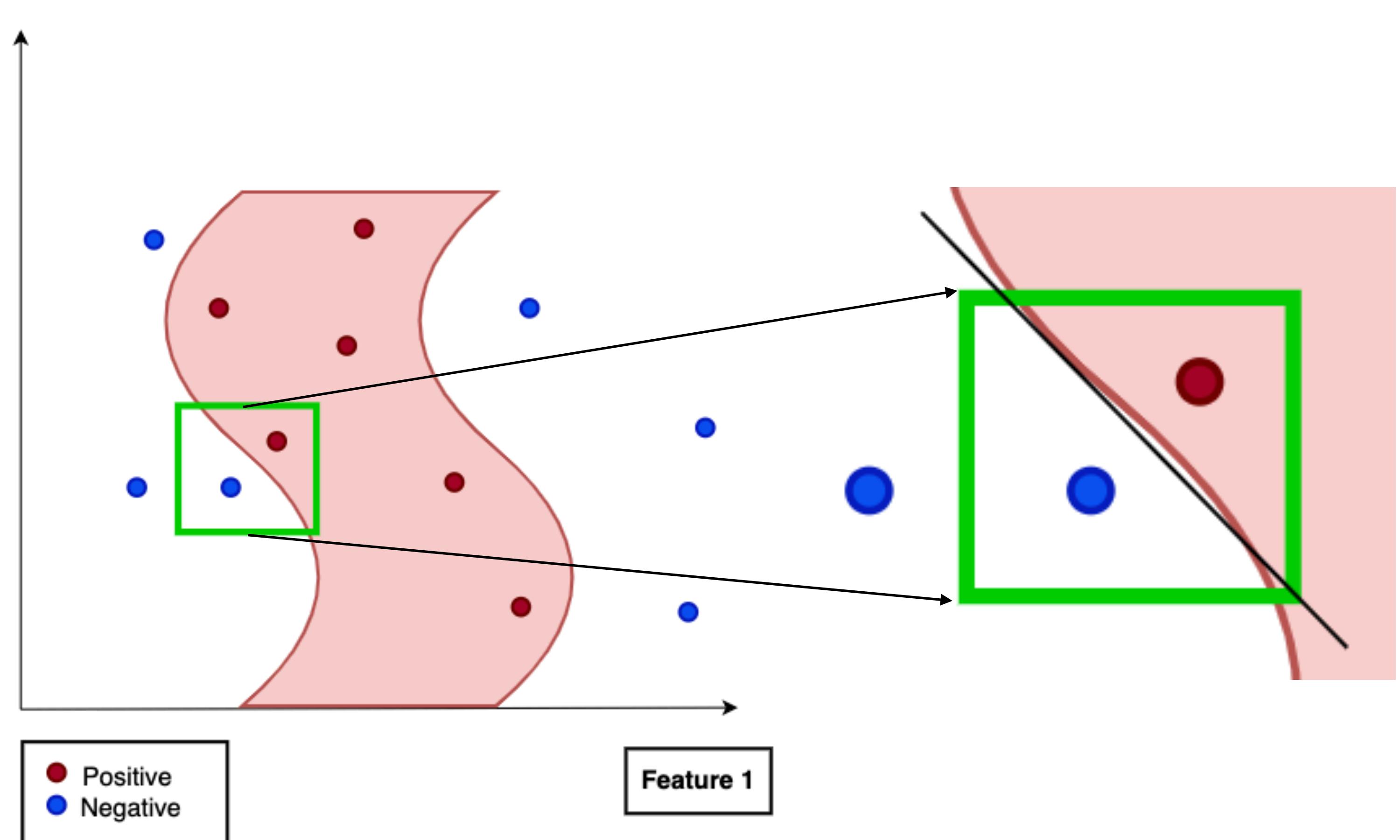
LIME

- LIME : Local Interpretable Model-agnostic Explanations
- Local Interpretable : 개별 샘플에 대한 결과 해석
- Model-agnostic : 모델에 무관하게 적용 가능
- Post Hoc : 모델 학습 이후 사후적으로 해석



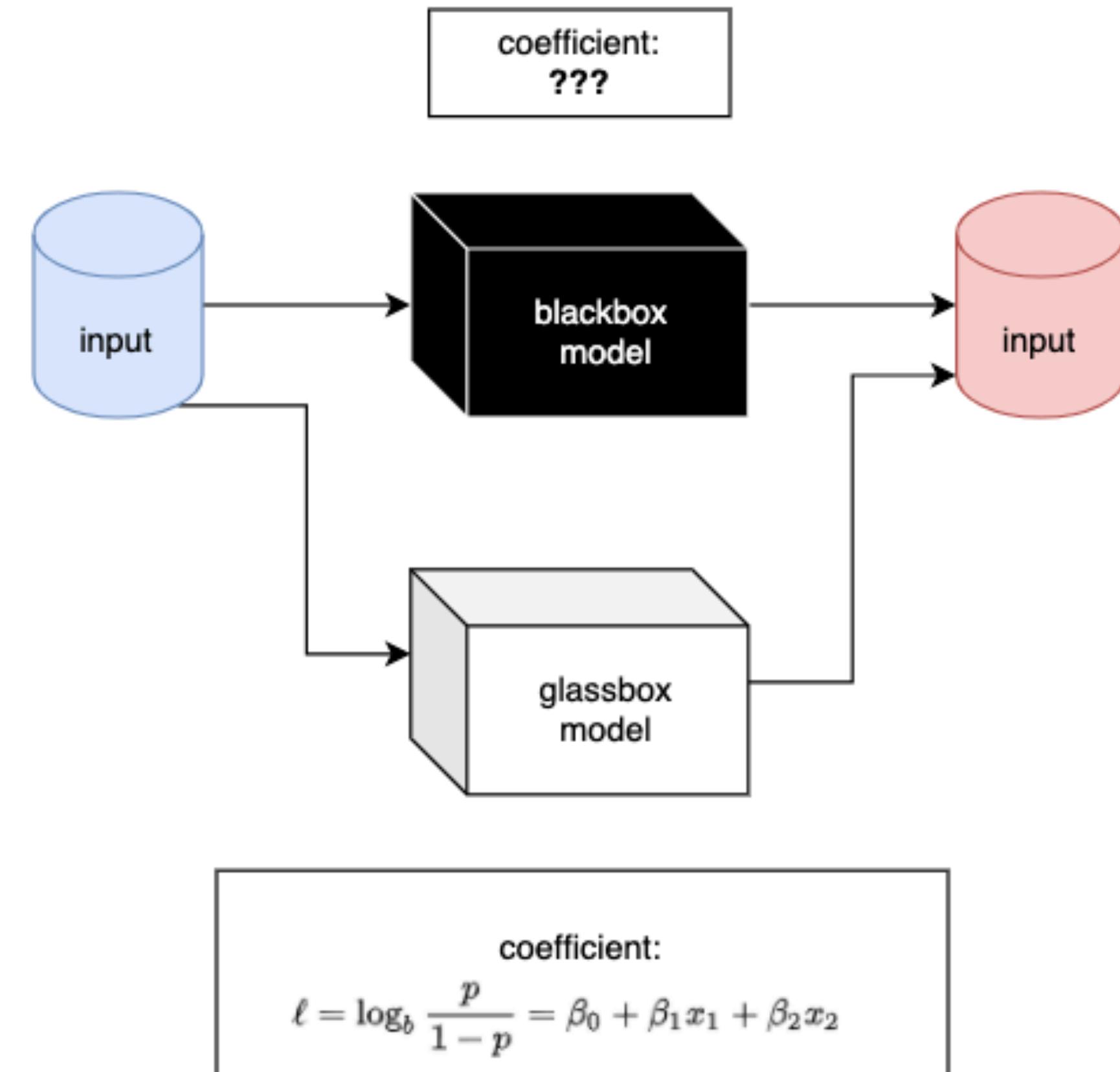
Surrogate Model

- 복잡한 비선형의 결정 경계
- 샘플 단위에서는?
- 샘플 근처에서는 복잡한 모델도 단순한 선형 모델로 근사 가능
- 국소지역의 설명가능한 대리모델



LIME의 구성

- 입력 데이터
- 블랙박스 모델 (복잡한 원래 모델)
- 출력 데이터
- 해석하기 쉬운 입력 표현
- 화이트박스 모델 (설명 가능한 대리모델)
 - 선형모델



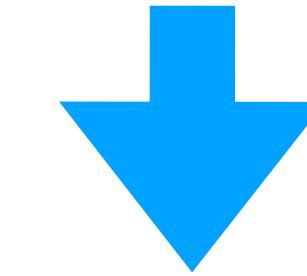
입력 데이터와 해석하기 쉬운 입력 표현

- 입력 데이터 (Original input)

- 해석하고자 하는 원본 입력 x

- e.g., 이미지, 텍스트, 테이블 데이터 등

나이	연봉	직업	신용등급
45	60k	연구원	B



- 해석하기 쉬운 입력 표현 (Interpretable input representation)

- 원본 입력을 사람이 해석하기 쉬운 형태로 변환한 것

- e.g.,

- 텍스트: 단어 단위 on/off

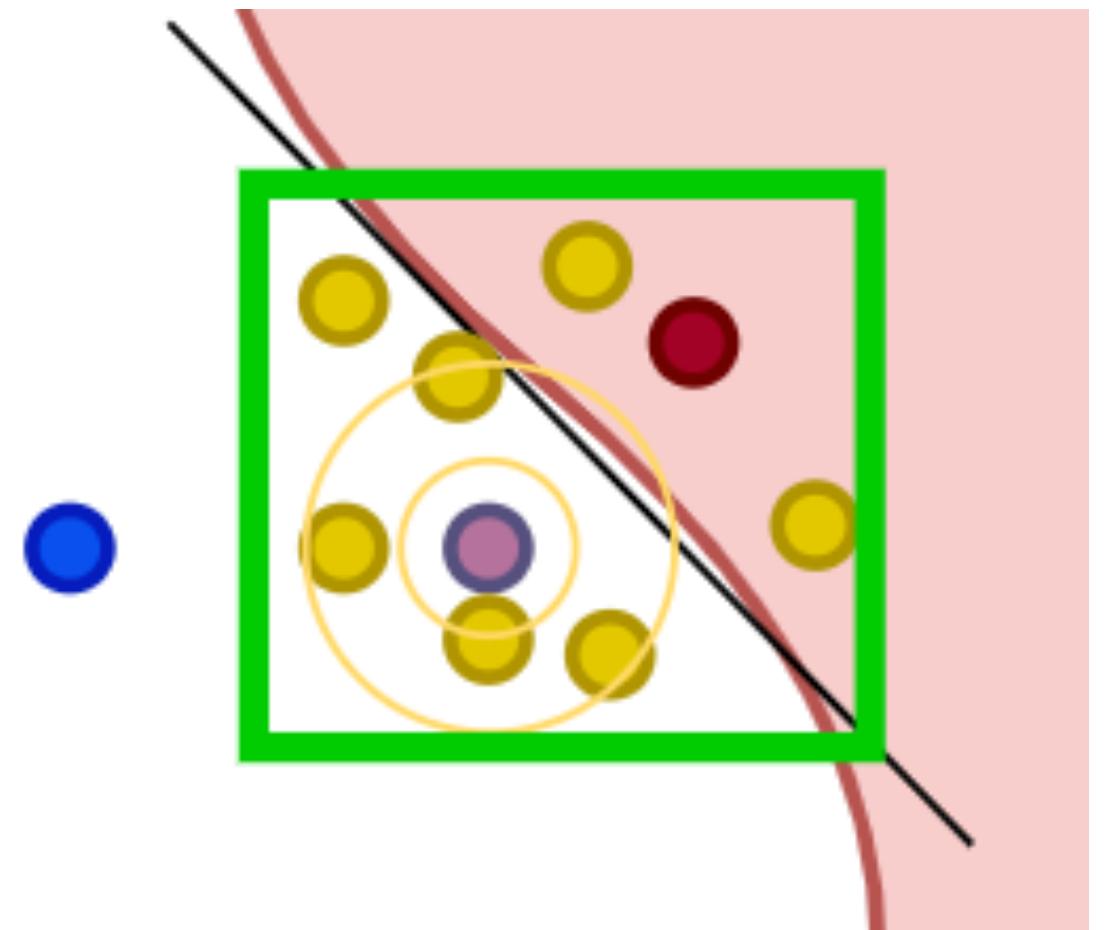
- 테이블: binning, one-hot 등 간소화된 feature 표현

10대	20대	30대	40대
0	0	0	1

연구원	변호사	학생	무직
1	0	0	0

LIME의 단계

- 어떤 샘플 x 에 대해서 설명하고 싶은 경우
- 샘플 x 에 대응하는 설명가능한 입력데이터 x' 으로 변환
- x' 의 주변 값(z')들을 샘플링 후 원래 차원의 데이터(z)로 변환
- 블랙박스 모델 f 의 출력 결과 $f(z)$ 를 확보
- 화이트 박스 모델 g 를 학습, 여기서 g 의 입력은 설명가능한 입력데이터임. $g(z')$
 - e.g. K-Lasso regression (L1 penalty - Sparse use of feature)



LIME의 학습 과정

- 블랙박스 모델 f

- 화이트박스 모델 g

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \quad \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

- π_x 주변 샘플들과의 유사도

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z) (f(z) - g(z'))^2$$

- Ω 모델 복잡성 계산 함수

- L: 가까운 샘플에 대해서 g 가 f 를 잘 근사 할 수 있도록 학습

- Ω : g 가 너무 복잡해져서 설명력을 잃지 않도록

LIME 의 설명

- 이진 분류 모델 f 가 주어졌을 때 True일 확률을 출력함
- predict_proba 함수를 통해 여러 샘플들의 클래스 분류 확률을 정답으로 g 학습
- 이를 선형 모델로 학습하면?
- 각 계수들에 대응하는 feature가 최종 출력확률에 어떤 영향을 미쳤는지 수치적으로 표현 가능

$$\text{e.g., } 0.78 = \frac{w_1x_1 + w_3x_3 + w_kx_k}{\text{출력에 기여한 feature들}} + \sum_{x_j=0} w_jx_j$$

데이터 전처리

```
› !pip install lime  
!pip install lightgbm  
from lime.lime_tabular import LimeTabularExplainer  
  
df = pd.read_csv('./Titanic-Dataset.csv')  
  
mean_imputer = SimpleImputer(strategy='mean')  
df['Age'] = mean_imputer.fit_transform(df[['Age']])  
mode_imputer = SimpleImputer(strategy='most_frequent')  
df[['Embarked']] = mode_imputer.fit_transform(df[['Embarked']])  
df_X = df.drop(['PassengerId', 'Name',  
                 'Ticket', 'Cabin', 'Survived'], axis=1)  
df_y = df.Survived
```

범주형 변수 처리

```
#numeric_features = ['Age', 'Fare', 'Parch', 'SibSp']
#categorical_features = ['Pclass','Sex', 'Embarked']
feature_names = df_X.columns.tolist()
categorical_feat_index = [0,1,6]
categorical_names = {}
for i in categorical_feat_index:
    le = LabelEncoder()
    df_X.iloc[:,i] = le.fit_transform(df_X.iloc[:,i])
    categorical_names[i] = le.classes_
df_X= df_X.astype('float')
```

모델 학습 및 준비

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import lightgbm as lgb
#데이터셋 분할
X_train, X_test, y_train, y_test = train_test_split(
    df_X, df_y, random_state=42)

lgbm = lgb.LGBMClassifier()
lgbm.fit(X_train, y_train)

y_hat = lgbm.predict(X_test)

acc = accuracy_score(y_test, y_hat)
print(acc)
```

Explainer 구현

```
explainer = LimeTabularExplainer(  
    #numpy 2d array  
    training_data=X_train.values,  
    feature_names = feature_names,  
    class_names = ['Not Survived', 'Survived'],  
    #categorical feature column index  
    categorical_features = [0, 1, 6],  
    #categorical_names[i][j], lable j for feature i  
    categorical_names = categorical_names)
```

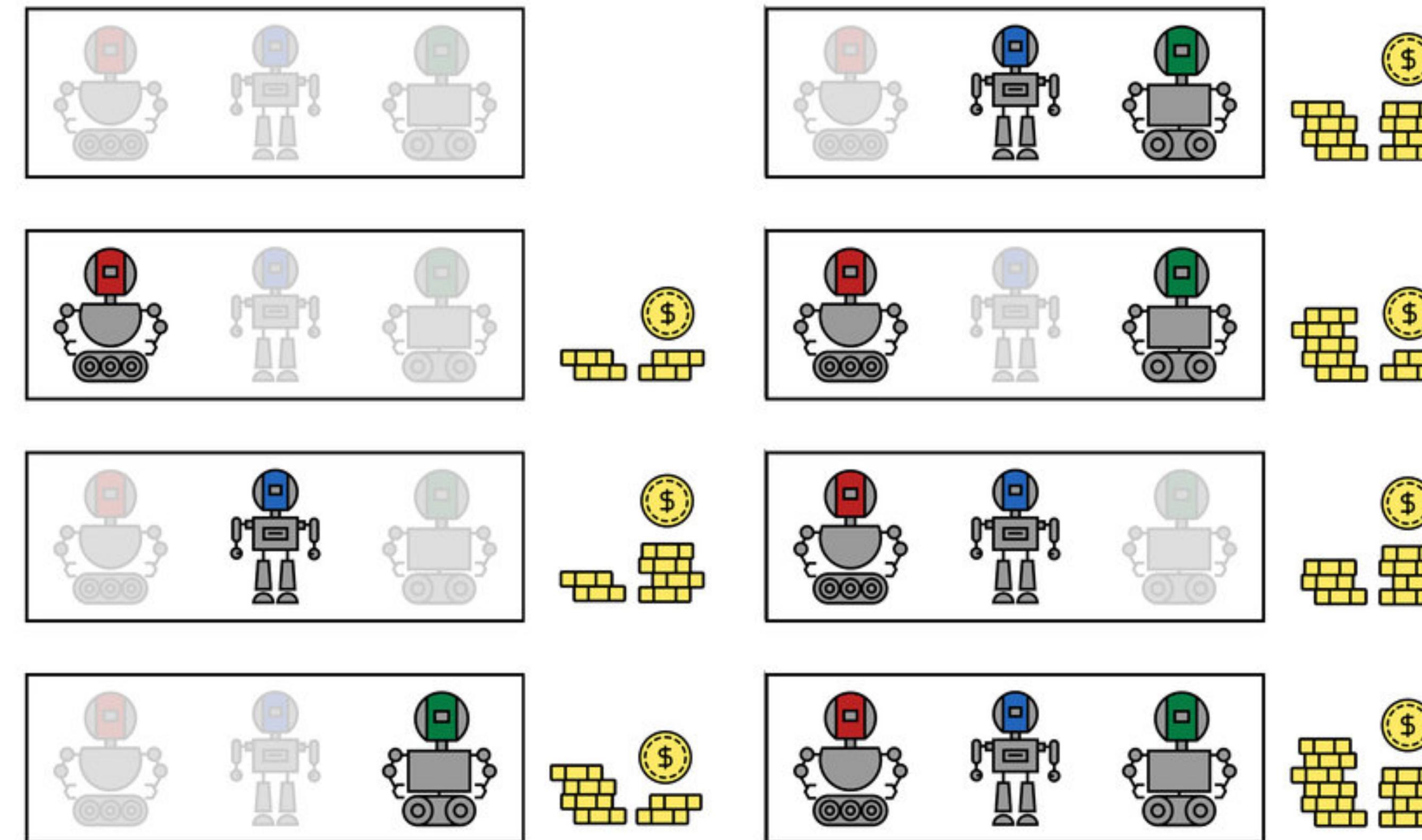
특정 샘플에 대한 설명 생성

```
#샘플 인덱스  
i=0  
#원래 설명하려던 모델의 출력을 얻기위한 함수  
predict_fn = lambda x: lgbm.predict_proba(x).astype(float)  
  
exp = explainer.explain_instance(  
    X_test.iloc[i].values, predict_fn)  
  
exp.show_in_notebook(show_all=True)
```

Shapley value

- Shapley value
- n명의 참가자 존재
- 개별 참여로 얻을 수 있는 이익 vs 그룹 참여로 얻을 수 있는 이익
- 그룹을 구성함으로써 이익이 증가했을 때, 각 참여자의 공정한 이익분배는?

Shapley value



Coalitional Game

- 플레이어의 집합 N
- 플레이어들의 서브셋 s
- 특성함수 $v(s) \rightarrow \mathbb{R}$, 플레이어들이 협력하여 s 를 구성했을때 얻을 수 있는 보상

$$v(s \cup \{i\}) - v(s)$$

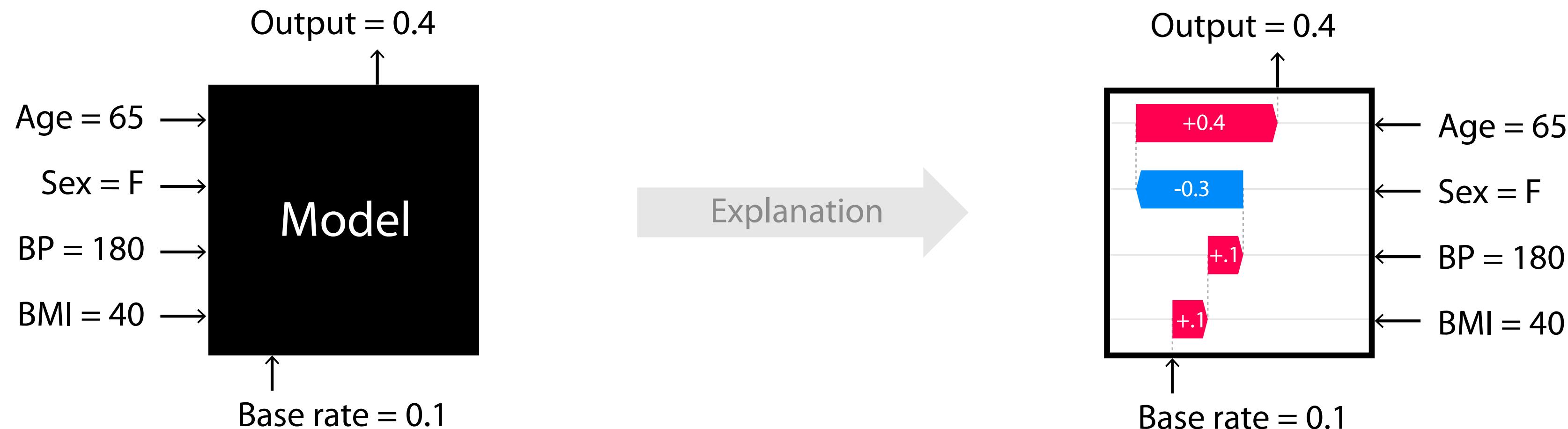
Shapley value and coalitional game

- 플레이어 i의 참여에 따른 평균 이익 변화량을 공정한 기여분으로 간주

$$\begin{aligned}\varphi_i(v) &= \sum_{S \subseteq N \setminus \{i\}} \frac{|S|! (n - |S| - 1)!}{n!} (v(S \cup \{i\}) - v(S)) \\ &= \frac{1}{n} \sum_{S \subseteq N \setminus \{i\}} \binom{n-1}{|S|}^{-1} (v(S \cup \{i\}) - v(S))\end{aligned}$$

SHAP

- SHAP: SHapley Additive exPlanation
- 특정 feature의 결과에 대한 기여는?



SHAP

Definition 1 Additive feature attribution methods have an explanation model that is a linear function of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i, \quad (1)$$

where $z' \in \{0, 1\}^M$, M is the number of simplified input features, and $\phi_i \in \mathbb{R}$.

- ϕ_i 를 Shapley value와 같이 feature i 사용에 따른 결과 변화량으로 계산

SHAP

- 로컬 정확성(Local Accuracy)
 - 개별 인스턴스에 대한 예측값을, 모든 피처별 Shapley 값의 합(+기저값)으로 재현
- 누락성(Missingness)
 - 해석 가능한 입력에서의 특정 피처의 중요도가 0이라면, 그 피처의 Shapley 값은 0
- 일관성(Consistency)
 - 두 모델을 비교했을 때, 어떤 피처가 한 모델에서 예측에 크게 기여한다면, 해당 피처의 Shapley 값도 커야함.

SHAP

- Additive explanation + properties 1,2,3 = SHAP

Theorem 1 *Only one possible explanation model g follows Definition 1 and satisfies Properties 1, 2, and 3:*

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)] \quad (8)$$

where $|z'|$ is the number of non-zero entries in z' , and $z' \subseteq x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries in x' .

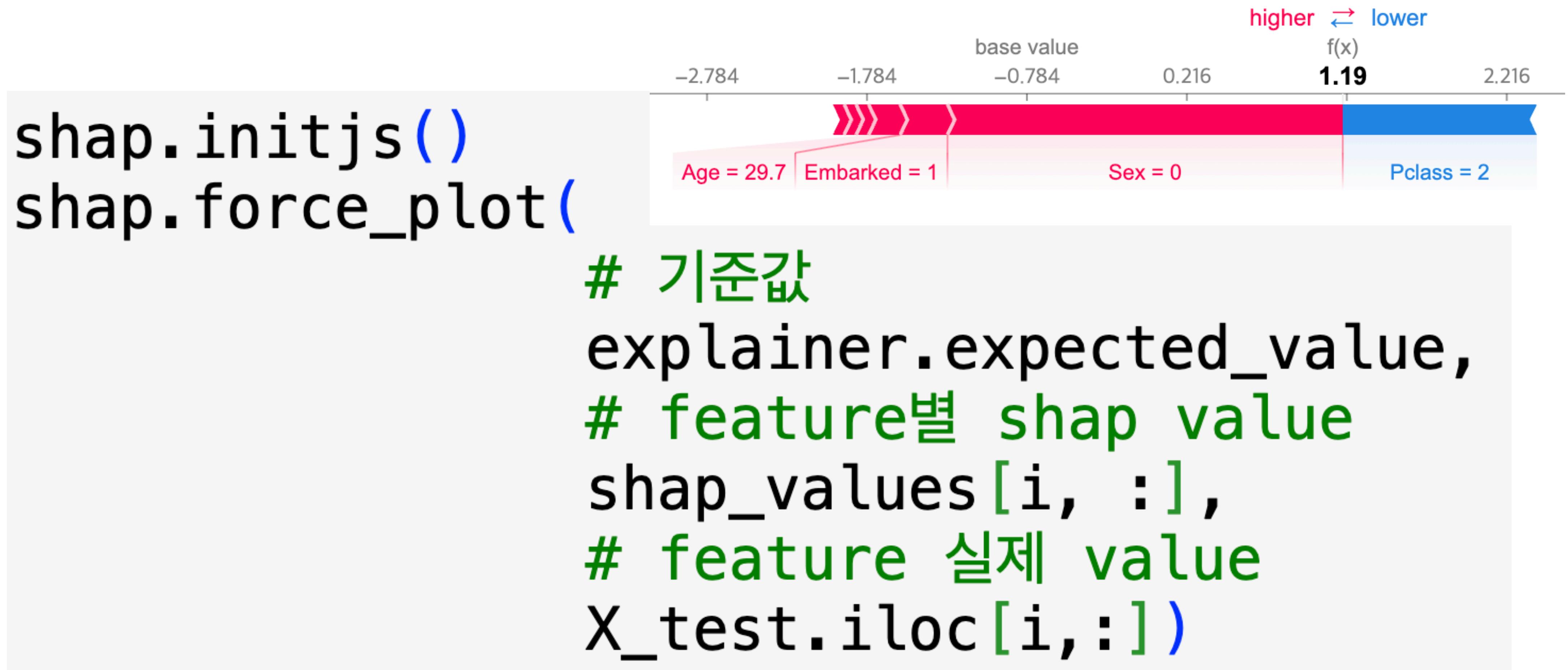
SHAP 계산 과정

- x 에 대해 궁금한 경우 \gg interpretable feature x' 으로 변형
- e.g., 동물(개, 고양이, 토끼), 몸무게 데이터일 때 $x = [\text{개}, 14] \gg x' = [1, 0, 0, 1]$
- Interpretable feature 가 1인경우 -> 실제 값을 사용 (실제 정보가 주어졌을때)
- Interpretable feature 가 0인 경우 -> background data에서 샘플링
(실제 정보가 주어지지 않은 상황이므로 기존 분포의 값으로 처리)
- 동물 데이터를 사용할 때, 사용하지 않을때의 각각의 모델 출력 결과를 알 수 있음.
- 즉 다음 네가지 interpretable feature 조합에 대한 비교 가능
- $[1, 0, 0, 1], [1, 0, 0, 0], [0, 0, 0, 1], [0, 0, 0, 0]$
- $\gg [x, 0, 0, 1]$ 일때의 출력 결과 차이와 $[x, 0, 0, 0]$ 일때의 출력 결과 차이의 가중합으로 동물 정보에 대한 SHAP 계산

SHAP value 계산

```
!pip install shap  
import shap  
  
#Tree SHAP algorithms for ensemble tree models.  
  
explainer = shap.TreeExplainer(lgbm)  
shap_values = explainer.shap_values(X_test)
```

개별 샘플에 대한 그래프 - force plot

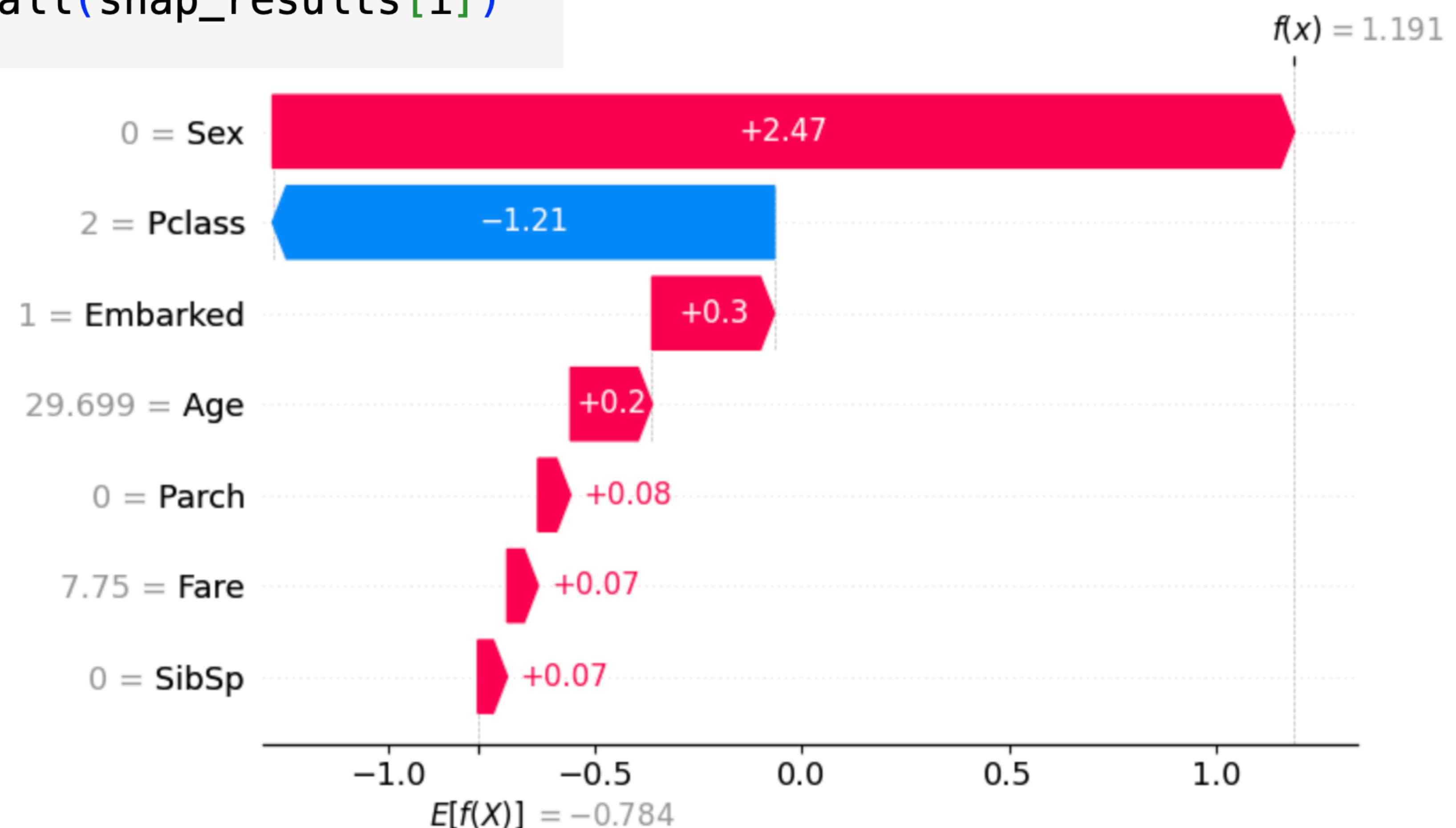


개별 샘플에 대한 그래프 - force plot

```
shap.initjs()  
shap.force_plot(  
    # 기준값  
    explainer.expected_value,  
    # feature별 shap value  
    shap_values[i, :],  
    # feature 실제 value  
    X_test.iloc[i,:])
```

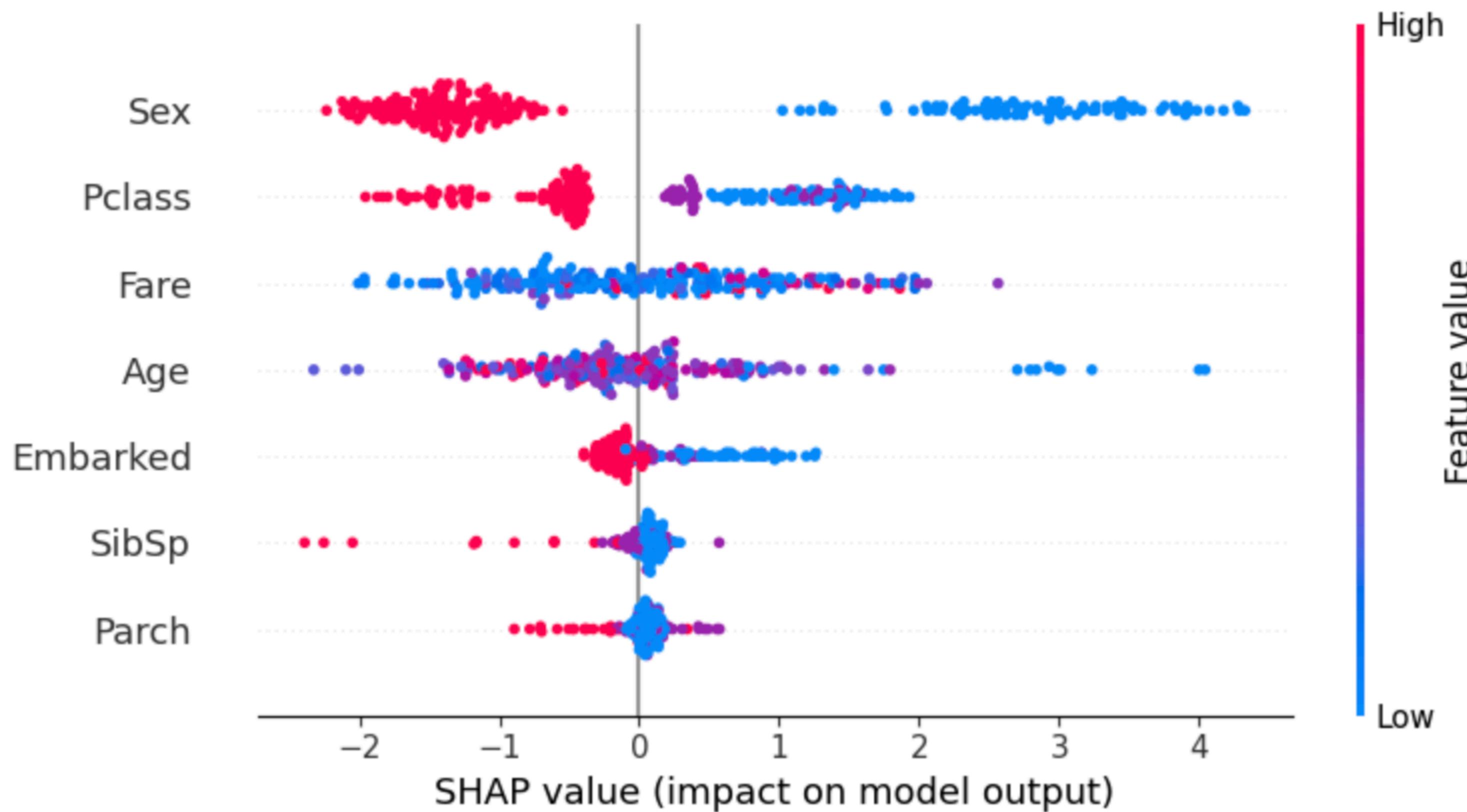
개별 샘플에 대한 그래프 - waterfall

```
shap_results = explainer(X_test)  
shap.plots.waterfall(shap_results[i])
```



전체 데이터에 플롯

```
shap.summary_plot(shap_values, X_test)
```



XAI with ChatGPT

- **Sex** 변수의 영향이 가장 큼
→ 성별은 생존 예측에 매우 큰 영향을 미치며, 여성(값이 높음, 빨간색)이 생존 가능성을 높이는 쪽(양의 SHAP값)으로 작용함.
- **Pclass**는 생존 확률과 음의 상관관계
→ 높은 Pclass(=1등석, 값이 낮음, 파란색)은 생존 확률을 높이며, 낮은 Pclass(=3등석, 값이 높음, 빨간색)은 생존 확률을 낮추는 쪽으로 작용함.
- **Fare**는 높은 요금일수록 생존 확률을 높임
→ 요금(Fare)이 높을수록 SHAP 값이 양수에 가까워지며, 이는 고액 승객일수록 생존 확률이 높다는 것을 의미함.
- **Age**는 전반적으로 생존 예측에 덜 민감하지만 경향 존재
→ 나이가 어린 승객(값이 낮음, 파란색)은 다소 생존 확률을 높이는 경향이 있음.