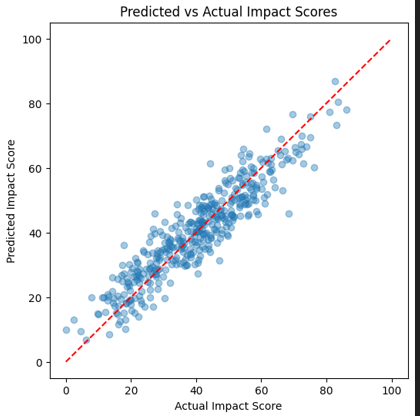
**Problem**: AI-Generated Medical Research Papers

**Graphical Solution**:



**Execution Output:**

# Step 1: Install dependencies

!pip install pandas numpy scikit-learn xgboost shap matplotlib streamlit --quiet

# Step 2: Imports

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_absolute\_error, r2\_score

import xgboost as xgb

import shap

import joblib

# ============================================================

# Step 3: Generate synthetic environmental data

# ============================================================

def generate\_synthetic\_env\_data(n=2000, seed=42):

    np.random.seed(seed)

    emissions = np.random.gamma(2, 10, n)

    water\_use = np.random.normal(500, 200, n).clip(min=10)

    land\_disturb = np.random.exponential(1.5, n)

    proximity = np.random.exponential(10, n)

    pop\_density = np.random.normal(300, 250, n).clip(min=1)

    project\_scale = np.random.choice(["small","medium","large"], size=n, p=[0.5,0.35,0.15])

    ndvi = np.random.uniform(0,1,n)

    is\_protected = (proximity < 2).astype(int)

    # synthetic impact score (target)

    impact\_score = (

        0.4 \* (emissions / (emissions.max()+1)) +

        0.2 \* (water\_use / (water\_use.max()+1)) +

        0.25 \* (land\_disturb / (land\_disturb.max()+1)) +

        0.3 \* (1 - ndvi) +

        0.15 \* (is\_protected) +

        0.2 \* (np.where(project\_scale=="large", 1,

                        np.where(project\_scale=="medium", 0.5, 0)))

    )

    impact\_score = (impact\_score + np.random.normal(0,0.05,n))

    impact\_score = (impact\_score - impact\_score.min()) / (impact\_score.max()-impact\_score.min())

    impact\_score = (impact\_score \* 100).round(2)

    df = pd.DataFrame({

        "emissions\_ton\_per\_year": emissions,

        "water\_use\_m3\_per\_day": water\_use,

        "land\_disturb\_ha": land\_disturb,

        "proximity\_km": proximity,

        "population\_density": pop\_density,

        "project\_scale": project\_scale,

        "ndvi": ndvi,

        "is\_protected": is\_protected,

        "impact\_score": impact\_score

    })

    return df

df = generate\_synthetic\_env\_data()

df.head()

# ============================================================

# Step 4: Preprocess & Split

# ============================================================

df = pd.get\_dummies(df, columns=["project\_scale"], drop\_first=True)

X = df.drop("impact\_score", axis=1)

y = df["impact\_score"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

num\_cols = X\_train.select\_dtypes(include=[np.number]).columns

X\_train[num\_cols] = scaler.fit\_transform(X\_train[num\_cols])

X\_test[num\_cols] = scaler.transform(X\_test[num\_cols])

# ============================================================

# Step 5: Train XGBoost model

# ============================================================

dtrain = xgb.DMatrix(X\_train, label=y\_train)

dtest = xgb.DMatrix(X\_test, label=y\_test)

params = {

    "objective": "reg:squarederror",

    "max\_depth": 6,

    "eta": 0.1,

    "subsample": 0.8,

    "seed": 42

}

bst = xgb.train(params, dtrain, num\_boost\_round=150)

preds = bst.predict(dtest)

mae = mean\_absolute\_error(y\_test, preds)

r2 = r2\_score(y\_test, preds)

print(f"✅ Model trained — MAE: {mae:.3f} | R²: {r2:.3f}")

# ============================================================

# Step 6: Visualize predicted vs actual

# ============================================================

plt.figure(figsize=(6,6))

plt.scatter(y\_test, preds, alpha=0.4)

plt.xlabel("Actual Impact Score")

plt.ylabel("Predicted Impact Score")

plt.title("Predicted vs Actual Impact Scores")

plt.plot([0,100],[0,100],"r--")

plt.show()

# ============================================================

# Step 7: Explainability with SHAP

# ============================================================

explainer = shap.TreeExplainer(bst)

sample\_X = X\_test.sample(200, random\_state=42)

shap\_values = explainer.shap\_values(sample\_X)

plt.title("Feature importance summary (SHAP values)")

shap.summary\_plot(shap\_values, sample\_X, show=True)