

```
In [33]: # initial cell
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.multioutput import MultiOutputRegressor
import shap

DATA_FILE = 'Health index1.csv'

# Load data
df = pd.read_csv(DATA_FILE)

# Drop exact duplicates and any rows with missing values
df = df.drop_duplicates().dropna()
print(f"Data shape after cleaning: {df.shape}")

# Optional: remove obvious outliers (e.g., beyond 3σ in any feature)
# Only apply to feature columns, not targets
feature_cols = df.columns[:-2] # Assuming last 2 are targets
for col in feature_cols:
    mean, std = df[col].mean(), df[col].std()
    df = df[(df[col] >= mean - 3*std) & (df[col] <= mean + 3*std)]

print(f"Data shape after outlier removal: {df.shape}")

# Prepare initial data splits
X = df.drop(columns=["Health index", "Life expectation"])
y = df[["Health index", "Life expectation"]]

# SHAP-based feature selection
# First, standardize for SHAP analysis
scaler_shap = StandardScaler()
X_scaled = pd.DataFrame(scaler_shap.fit_transform(X), columns=X.columns)

# Train model for SHAP analysis
model_shap = MultiOutputRegressor(ExtraTreesRegressor(n_estimators=200, random_s
model_shap.fit(X_scaled, y)

# SHAP analysis - use first estimator for feature importance
explainer = shap.TreeExplainer(model_shap.estimators_[0])
shap_values = explainer.shap_values(X_scaled)

# Calculate feature importance based on mean absolute SHAP values
shap_means = np.abs(shap_values).mean(0)
shap_imports = pd.Series(shap_means, index=X.columns).sort_values(ascending=False)

print("Top SHAP feature importances:")
print(shap_imports.head(8))

# Select top 8 features
top_feats = shap_imports.head(8).index.tolist()

# Apply feature selection
X_selected = X[top_feats]
```

```

# Apply smoothing to selected features and targets
to_smooth = top_feats + ['Health index', 'Life expectation']
df_smooth = df[to_smooth].rolling(window=5, min_periods=1).mean()

# Remove any remaining NaN values after smoothing
df_smooth = df_smooth.dropna()

print(f"Data shape after smoothing: {df_smooth.shape}")

# After all preprocessing, prepare final data
X_processed = df_smooth[top_feats]
y_processed = df_smooth[['Health index', 'Life expectation']]

# Normalize the data column-wise (final step)
X_norm = (X_processed - X_processed.mean()) / X_processed.std()
y_norm = (y_processed - y_processed.mean()) / y_processed.std()

# Split into train and test sets (final variables)
X_train, X_test, y_train, y_test = train_test_split(X_norm, y_norm, test_size=0.

```

Data shape after cleaning: (470, 16)
 Data shape after outlier removal: (379, 16)
 Top SHAP feature importances:

DBDS	5.400746
Interfacial V	3.169584
Acetylene	1.362682
Methane	1.123805
Power factor	0.965366
Water content	0.908013
CO2	0.776445
Hydrogen	0.586350

dtype: float64
 Data shape after smoothing: (379, 10)

In [34]:

```

from sklearn.linear_model import MultiTaskLasso
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd

# Initialize and train MultiTaskLasso
lasso = MultiTaskLasso(alpha=0.1, random_state=42)
lasso.fit(X_train, y_train)

# Predict on test set
y_pred = lasso.predict(X_test)

# Denormalize predictions for evaluation
y_std = y.std()
y_mean = y.mean()
y_pred_denorm = pd.DataFrame(y_pred * y_std.values + y_mean.values,
                             columns=y.columns,
                             index=y_test.index)
y_test_denorm = y_test * y_std + y_mean

# Evaluate the model
mse = mean_squared_error(y_test_denorm, y_pred_denorm, multioutput='raw_values')
r2 = r2_score(y_test_denorm, y_pred_denorm, multioutput='raw_values')
mape = np.mean(np.abs((y_test_denorm - y_pred_denorm) / y_test_denorm), axis=0)
mape = pd.Series(mape, index=y.columns) # Ensure consistent Series with Labels

```

```

print("Model Performance:\n")
print("Health Index:")
print(f"MSE: {mse[0]:.6f}")
print(f"R²: {r2[0]:.6f}")
print(f"MAPE: {mape.iloc[0]:.6f}%\n")

print("Life Expectation:")
print(f"MSE: {mse[1]:.6f}")
print(f"R²: {r2[1]:.6f}")
print(f"MAPE: {mape.iloc[1]:.6f}%")



```

Model Performance:

Health Index:
MSE: 90.600530
R²: 0.684890
MAPE: 29.331015%

Life Expectation:
MSE: 90.884206
R²: 0.707110
MAPE: 161.566892%

```

In [35]: from sklearn.linear_model import MultiTaskElasticNet

# Fit MultiTaskElasticNet and compute R² values
enet = MultiTaskElasticNet(alpha=0.1, random_state=42)
enet.fit(X_train, y_train)
y_pred_enet = enet.predict(X_test)
from sklearn.metrics import r2_score
r2_enet = r2_score(y_test, y_pred_enet, multioutput='raw_values')

# Print R² values from MultiTaskElasticNet
print("\nMultiTaskElasticNet R² values:")
print(f"Health Index R²: {r2_enet[0]:.6f}")
print(f"Life Expectation R²: {r2_enet[1]:.6f}")



```

MultiTaskElasticNet R² values:
Health Index R²: 0.735071
Life Expectation R²: 0.719207

```

In [36]: from sklearn.ensemble import RandomForestRegressor

# Initialize RandomForestRegressor for each target
rf_health = RandomForestRegressor(random_state=42)
rf_life = RandomForestRegressor(random_state=42)

# Fit the models
rf_health.fit(X_train, y_train["Health index"])
rf_life.fit(X_train, y_train["Life expectation"])

# Predict on test set
y1_pred = rf_health.predict(X_test)
y2_pred = rf_life.predict(X_test)

# Evaluate
health_mse = mean_squared_error(y_test["Health index"], y1_pred)
health_r2 = r2_score(y_test["Health index"], y1_pred)
life_mse = mean_squared_error(y_test["Life expectation"], y2_pred)
life_r2 = r2_score(y_test["Life expectation"], y2_pred)



```

```
print("Test Set Performance:")
print(f"Health Index - MSE: {health_mse:.6f}, R2: {health_r2:.6f}")
print(f"Life Expectation - MSE: {life_mse:.6f}, R2: {life_r2:.6f}")
```

Test Set Performance:

Health Index - MSE: 0.133032, R²: 0.893413

Life Expectation - MSE: 0.078548, R²: 0.922013

```
In [37]: # Use already Loaded X, y1, y2 from previous cells
from sklearn.ensemble import ExtraTreesRegressor

# Initialize and train ExtraTrees models
et_health = ExtraTreesRegressor(random_state=42).fit(X_train, y_train["Health index"])
et_life = ExtraTreesRegressor(random_state=42).fit(X_train, y_train["Life expectation"])

# Predict on test data
y1_pred_test = et_health.predict(X_test)
y2_pred_test = et_life.predict(X_test)

# Calculate testing errors
errors_test = {
    "Health Index": {
        "MSE": mean_squared_error(y_test["Health index"], y1_pred_test),
        "R2": r2_score(y_test["Health index"], y1_pred_test)
    },
    "Life Expectation": {
        "MSE": mean_squared_error(y_test["Life expectation"], y2_pred_test),
        "R2": r2_score(y_test["Life expectation"], y2_pred_test)
    }
}

# Print results
print("ExtraTreesRegressor Testing Errors:")
for target, metrics in errors_test.items():
    print(f"{target}: MSE = {metrics['MSE']:.6f}, R2 = {metrics['R2]:.6f}")
```

ExtraTreesRegressor Testing Errors:

Health Index: MSE = 0.113976, R² = 0.908681

Life Expectation: MSE = 0.064018, R² = 0.936439

```
In [38]: from sklearn.ensemble import GradientBoostingRegressor

# Initialize Gradient Boosting models for each target
gb_health = GradientBoostingRegressor(random_state=42)
gb_life = GradientBoostingRegressor(random_state=42)

# Train the models
gb_health.fit(X_train, y_train["Health index"])
gb_life.fit(X_train, y_train["Life expectation"])

# Predict on test data
y1_pred_test = gb_health.predict(X_test)
y2_pred_test = gb_life.predict(X_test)

# Calculate testing errors
errors_test = {
    "Health Index": {
        "MSE": mean_squared_error(y_test["Health index"], y1_pred_test),
        "R2": r2_score(y_test["Health index"], y1_pred_test)
    },
    "Life Expectation": {
```

```

        "MSE": mean_squared_error(y_test["Life expectation"], y2_pred_test),
        "R2": r2_score(y_test["Life expectation"], y2_pred_test)
    }
}

# Print results
print("GradientBoostingRegressor Testing Errors:")
for target, metrics in errors_test.items():
    print(f"{target}: MSE = {metrics['MSE']:.6f}, R2 = {metrics['R2]:.6f}")

```

GradientBoostingRegressor Testing Errors:
 Health Index: MSE = 0.147575, R² = 0.881761
 Life Expectation: MSE = 0.071129, R² = 0.929379

```

In [39]: import xgboost as xgb

# Use the already split X_train, X_test, y_train, y_test from previous cells
# Use columns from y for each target
y_health_train = y_train["Health index"]
y_health_test = y_test["Health index"]
y_life_train = y_train["Life expectation"]
y_life_test = y_test["Life expectation"]

# Initialize XGBoost models
xgb_health = xgb.XGBRegressor(random_state=42)
xgb_life = xgb.XGBRegressor(random_state=42)

# Train models
xgb_health.fit(X_train, y_health_train)
xgb_life.fit(X_train, y_life_train)

# Predict on test data
y_health_pred = xgb_health.predict(X_test)
y_life_pred = xgb_life.predict(X_test)

# Calculate R2 and MSE for test data
health_test_mse = mean_squared_error(y_health_test, y_health_pred)
health_test_r2 = r2_score(y_health_test, y_health_pred)

life_test_mse = mean_squared_error(y_life_test, y_life_pred)
life_test_r2 = r2_score(y_life_test, y_life_pred)

# Print results
print("XGBoost Test Set Performance:\n")
print("Health Index:")
print(f"Test MSE: {health_test_mse:.6f}")
print(f"Test R2: {health_test_r2:.6f}\n")

print("Life Expectation:")
print(f"Test MSE: {life_test_mse:.6f}")
print(f"Test R2: {life_test_r2:.6f}\n")

# Model is overfitting
# The training error is much lower than the cross-validation error
# The model is not generalizing well to unseen data

```

XGBoost Test Set Performance:

Health Index:
 Test MSE: 0.144970
 Test R²: 0.883848

Life Expectation:
 Test MSE: 0.050952
 Test R²: 0.949412

```
In [40]: from lightgbm import LGBMRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error, r2_score
import warnings

# Suppress LightGBM warnings
warnings.filterwarnings("ignore", category=UserWarning)

# Define model with feature_fraction and bagging_fraction only
lgb = LGBMRegressor(
    max_depth=8,
    num_leaves=50,
    min_child_samples=5,
    feature_fraction=0.7, # preferred
    bagging_fraction=0.7, # preferred
    random_state=42,
    verbose=-1, # suppress training logs
    force_row_wise=True
)

multi_lgb = MultiOutputRegressor(lgb)

# Train the model
multi_lgb.fit(X_train, y_train)

# Predict on test data
y_pred_test = multi_lgb.predict(X_test)

# Calculate testing errors
test_mse = mean_squared_error(y_test, y_pred_test, multioutput='raw_values')
test_r2 = r2_score(y_test, y_pred_test, multioutput='raw_values')

# Print results
target_names = ['Health Index', 'Life Expectation']
print("LightGBM MultiOutputRegressor Testing Performance:\n")
for i, name in enumerate(target_names):
    print(f"{name}:")
    print(f"  Test MSE: {test_mse[i]:.6f}")
    print(f"  Test R2: {test_r2[i]:.6f}")
```

LightGBM MultiOutputRegressor Testing Performance:

Health Index:
 Test MSE: 0.140703
 Test R²: 0.887267

Life Expectation:
 Test MSE: 0.071736
 Test R²: 0.928776

```
In [41]: from catboost import CatBoostRegressor

# Initialize CatBoost with MultiRMSE objective
cb = CatBoostRegressor(
    iterations=500,
    learning_rate=0.1,
    depth=6,
    loss_function='MultiRMSE',
    random_seed=42,
    verbose=0
)

# Train the model
cb.fit(X_train, y_train)

# Predict on test data
y_pred_test = cb.predict(X_test)

# Calculate testing errors
test_mse = mean_squared_error(y_test, y_pred_test, multioutput='raw_values')
test_r2 = r2_score(y_test, y_pred_test, multioutput='raw_values')

# Print results
target_names = ['Health Index', 'Life Expectation']
print("CatBoost Testing Performance:\n")
for i, name in enumerate(target_names):
    print(f"{name}:")
    print(f" Test MSE: {test_mse[i]:.6f}")
    print(f" Test R²: {test_r2[i]:.6f}")
```

CatBoost Testing Performance:

```
Health Index:
Test MSE: 0.185252
Test R²: 0.851573
Life Expectation:
Test MSE: 0.063784
Test R²: 0.936672
```

```
In [42]: from sklearn.neighbors import KNeighborsRegressor

# Initialize KNN Regressor
knn = KNeighborsRegressor(
    n_neighbors=5,           # Number of neighbors to consider
    weights='uniform',       # Weighting: 'uniform' or 'distance'
    algorithm='auto',        # Algorithm: 'auto', 'ball_tree', 'kd_tree', 'brute'
    p=2                      # Power parameter (2=euclidean distance)
)

# Train the model
knn.fit(X_train, y_train)

# Predict on test data
y_pred_test = knn.predict(X_test)

# Calculate testing errors
test_mse = mean_squared_error(y_test, y_pred_test, multioutput='raw_values')
test_r2 = r2_score(y_test, y_pred_test, multioutput='raw_values')

# Print results
```

```
target_names = ['Health Index', 'Life Expectation']
print("K-Nearest Neighbors Regressor Testing Performance:\n")
for i, name in enumerate(target_names):
    print(f"{name}:")
    print(f"  Test MSE: {test_mse[i]:.6f}")
    print(f"  Test R²: {test_r2[i]:.6f}")
```

K-Nearest Neighbors Regressor Testing Performance:

Health Index:
 Test MSE: 0.180933
 Test R²: 0.855033
 Life Expectation:
 Test MSE: 0.117758
 Test R²: 0.883082

In [43]:

```
from sklearn.svm import SVR
from sklearn.multioutput import MultiOutputRegressor
from sklearn.preprocessing import StandardScaler

# Scale features (important for SVR)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Initialize SVR with MultiOutputRegressor
svr = SVR(
    kernel='rbf',           # Radial basis function kernel
    C=1.0,                 # Regularization parameter
    epsilon=0.1,            # Epsilon in epsilon-SVR
    gamma='scale'           # Kernel coefficient
)
multi_svr = MultiOutputRegressor(svr)

# Train the model
multi_svr.fit(X_train, y_train)

# Predict on test data
y_pred_test = multi_svr.predict(X_test)

# Calculate testing errors
test_mse = mean_squared_error(y_test, y_pred_test, multioutput='raw_values')
test_r2 = r2_score(y_test, y_pred_test, multioutput='raw_values')

# Print results
target_names = ['Health Index', 'Life Expectation']
print("SVR (MultiOutputRegressor) Testing Performance:\n")
for i, name in enumerate(target_names):
    print(f"{name}:")
    print(f"  Test MSE: {test_mse[i]:.6f}")
    print(f"  Test R²: {test_r2[i]:.6f}")
```

SVR (MultiOutputRegressor) Testing Performance:

Health Index:
 Test MSE: 0.317412
 Test R²: 0.745684
 Life Expectation:
 Test MSE: 0.169287
 Test R²: 0.831921

```
In [44]: from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel
from sklearn.preprocessing import StandardScaler

# Scale features (important for Gaussian Process)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Define kernel for Gaussian Process
kernel = ConstantKernel(1.0) * RBF(length_scale=1.0)

# Initialize GaussianProcessRegressor
gp = GaussianProcessRegressor(
    kernel=kernel,
    alpha=1e-10, # Added to diagonal for numerical stability
    normalize_y=True,
    random_state=42
)

# Train the model
gp.fit(X_train, y_train)

# Predict on test data
y_pred_test = gp.predict(X_test)

# Calculate testing errors
test_mse = mean_squared_error(y_test, y_pred_test, multioutput='raw_values')
test_r2 = r2_score(y_test, y_pred_test, multioutput='raw_values')

# Print results
target_names = ['Health Index', 'Life Expectation']
print("GaussianProcessRegressor Testing Performance:\n")
for i, name in enumerate(target_names):
    print(f"{name}:")
    print(f" Test MSE: {test_mse[i]:.6f}")
    print(f" Test R^2: {test_r2[i]:.6f}")
```

GaussianProcessRegressor Testing Performance:

Health Index:
 Test MSE: 224.171903
 Test R²: 0.202147
 Life Expectation:
 Test MSE: 212.204413
 Test R²: 0.326832

```
In [45]: from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler

# Scale features and targets (critical for neural networks)
X_scaler = StandardScaler()
X_scaled = X_scaler.fit_transform(X)

y_scaler = StandardScaler()
y_scaled = y_scaler.fit_transform(y)

# Split data into training and testing sets
```

```

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.2)

# Initialize MLP Regressor
mlp = MLPRegressor(
    hidden_layer_sizes=(100, 50), # Network architecture
    activation='relu', # Activation function
    solver='adam', # Optimization algorithm
    alpha=0.0001, # L2 regularization
    batch_size=32, # Mini-batch size
    learning_rate_init=0.001, # Initial learning rate
    max_iter=500, # Maximum iterations
    random_state=42,
    early_stopping=True, # Stop if validation score doesn't improve
    validation_fraction=0.1 # Fraction of training data for validation
)

# Train the model
mlp.fit(X_train, y_train)

# Predict on test data
y_pred_test_scaled = mlp.predict(X_test)
y_pred_test = y_scaler.inverse_transform(y_pred_test_scaled)
y_test_original = y_scaler.inverse_transform(y_test)

# Calculate testing errors
test_mse = mean_squared_error(y_test_original, y_pred_test, multioutput='raw_values')
test_r2 = r2_score(y_test_original, y_pred_test, multioutput='raw_values')

# Print results
target_names = ['Health Index', 'Life Expectation']
print("MLP Regressor Testing Performance:\n")
for i, name in enumerate(target_names):
    print(f"{name}:")
    print(f"  Test MSE: {test_mse[i]:.6f}")
    print(f"  Test R^2: {test_r2[i]:.6f}")

```

MLP Regressor Testing Performance:

Health Index:

```

Test MSE: 136.547133
Test R2: 0.514013

```

Life Expectation:

```

Test MSE: 124.291735
Test R2: 0.605714

```

In [46]:

```

from sklearn.linear_model import Ridge
from sklearn.multioutput import RegressorChain
from sklearn.preprocessing import StandardScaler

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,)

# Initialize RegressorChain with Ridge as the base estimator
chain = RegressorChain(
    base_estimator=Ridge(alpha=1.0, random_state=42),
    order='random',
)

```

```

    random_state=42
)

# Train the model
chain.fit(X_train, y_train)

# Predict on test data
y_pred_test = chain.predict(X_test)

# Calculate testing errors
test_mse = mean_squared_error(y_test, y_pred_test, multioutput='raw_values')
test_r2 = r2_score(y_test, y_pred_test, multioutput='raw_values')

# Print results
target_names = ['Health Index', 'Life Expectation']
print("RegressorChain Testing Performance:\n")
for i, name in enumerate(target_names):
    print(f"{name}:")
    print(f"  Test MSE: {test_mse[i]:.6f}")
    print(f"  Test R^2: {test_r2[i]:.6f}")

```

RegressorChain Testing Performance:

Health Index:
 Test MSE: 147.184204
 Test R²: 0.476155
 Life Expectation:
 Test MSE: 186.262563
 Test R²: 0.409127

In [47]:

```

from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.multioutput import MultiOutputRegressor

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,)

# Define base estimators
estimators = [
    ('svr', SVR(kernel='rbf', C=1.0, gamma='scale')),
    ('dt', DecisionTreeRegressor(max_depth=5, random_state=42))
]

# Initialize StackingRegressor with Ridge as final estimator
base_stack = StackingRegressor(
    estimators=estimators,
    final_estimator=Ridge(alpha=1.0),
    cv=5,
    n_jobs=-1
)
stack = MultiOutputRegressor(base_stack)

# Train the model

```

```

stack.fit(X_train, y_train)

# Predict on test data
y_pred_test = stack.predict(X_test)

# Calculate testing errors
test_mse = mean_squared_error(y_test, y_pred_test, multioutput='raw_values')
test_r2 = r2_score(y_test, y_pred_test, multioutput='raw_values')

# Print results
target_names = ['Health Index', 'Life Expectation']
print("StackingRegressor Testing Performance:\n")
for i, name in enumerate(target_names):
    print(f"{name}:")
    print(f"  Test MSE: {test_mse[i]:.6f}")
    print(f"  Test R²: {test_r2[i]:.6f}")

```

StackingRegressor Testing Performance:

Health Index:
 Test MSE: 140.944876
 Test R²: 0.498361

Life Expectation:
 Test MSE: 139.887596
 Test R²: 0.556240

```

In [48]: """
This is a Python script to perform sklearn.ensemble.VotingRegressor analysis
from sklearn.ensemble import VotingRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression

# Initialize individual regressors
lr = LinearRegression()
rf = RandomForestRegressor(random_state=42)
gb = GradientBoostingRegressor(random_state=42)

# Create the VotingRegressor
voting_regressor = VotingRegressor(estimators=[
    ('lr', lr),
    ('rf', rf),
    ('gb', gb)
])

# Train on each output separately using MultiOutputRegressor wrapper
from sklearn.multioutput import MultiOutputRegressor

# MultiOutputRegressor allows us to fit multiple regressors for each target
multi_output_voting = MultiOutputRegressor(voting_regressor)
multi_output_voting.fit(X_train, y_train)

# Predict on test set
y_pred_vote = multi_output_voting.predict(X_test)

# Evaluate the model
mse_vote = mean_squared_error(y_test, y_pred_vote, multioutput='raw_values')
r2_vote = r2_score(y_test, y_pred_vote, multioutput='raw_values')
mape_vote = np.mean(np.abs((y_test - y_pred_vote) / y_test), axis=0) * 100

mse_vote, r2_vote, mape_vote

```

```
# R2 values from VotingRegressor
print("\nVotingRegressor R2 values:")
print(f"Health Index R2: {r2_vote[0]:.6f}")
print(f"Life Expectation R2: {r2_vote[1]:.6f}")

# Lower error than the previous model but not sufficiently low
# The data maybe tree based, may work with MLP Regressor
# Might need to use neural networks
# Might need to use Support Vector machines
```

```
VotingRegressor R2 values:
Health Index R2: 0.603722
Life Expectation R2: 0.598618
```