!pip install shap

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.preprocessing import StandardScaler

# Sample dataset (replace with your dataset)

data = pd.DataFrame({

    'Age': np.random.randint(18, 80, size=100),

    'BMI': np.round(np.random.uniform(15, 40, size=100), 1),

    'BP Systolic': np.random.randint(90, 180, size=100),

    'BP Diastolic': np.random.randint(60, 120, size=100),

    'Cholesterol': np.random.randint(100, 300, size=100),

    'Disease': np.random.choice([0, 1], size=100)  # Binary target variable

})

# Splitting the dataset into features and target variable

X = data.drop('Disease', axis=1)

y = data['Disease']

# Standardize the 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=0.3, random\_state=42)

# 1. Logistic Regression

logreg = LogisticRegression(random\_state=42)

logreg.fit(X\_train, y\_train)

y\_pred\_logreg = logreg.predict(X\_test)

logreg\_score = accuracy\_score(y\_test, y\_pred\_logreg)

# 2. Decision Tree

tree = DecisionTreeClassifier(random\_state=42)

tree.fit(X\_train, y\_train)

y\_pred\_tree = tree.predict(X\_test)

tree\_score = accuracy\_score(y\_test, y\_pred\_tree)

# 3. Random Forest

forest = RandomForestClassifier(random\_state=42)

forest.fit(X\_train, y\_train)

y\_pred\_forest = forest.predict(X\_test)

forest\_score = accuracy\_score(y\_test, y\_pred\_forest)

# 4. Support Vector Machine (SVM)

svm = SVC(kernel='linear', random\_state=42)

svm.fit(X\_train, y\_train)

y\_pred\_svm = svm.predict(X\_test)

svm\_score = accuracy\_score(y\_test, y\_pred\_svm)

# Print accuracy scores

print(f"Logistic Regression accuracy: {logreg\_score:.4f}")

print(f"Decision Tree accuracy: {tree\_score:.4f}")

print(f"Random Forest accuracy: {forest\_score:.4f}")

print(f"SVM accuracy: {svm\_score:.4f}")

Logistic Regression accuracy: 0.5667

Decision Tree accuracy: 0.5667

Random Forest accuracy: 0.4667

SVM accuracy: 0.5333

# Detailed classification report

print("\nClassification Report for Random Forest:")

print(classification\_report(y\_test, y\_pred\_forest))

Classification Report for Random Forest:

precision recall f1-score support

0 0.25 0.17 0.20 12

1 0.55 0.67 0.60 18

accuracy 0.47 30

macro avg 0.40 0.42 0.40 30

weighted avg 0.43 0.47 0.44 30

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

from sklearn.preprocessing import StandardScaler

# Sample dataset (replace with your dataset)

data = pd.DataFrame({

    'Age': np.random.randint(18, 80, size=100),

    'BMI': np.round(np.random.uniform(15, 40, size=100), 1),

    'BP Systolic': np.random.randint(90, 180, size=100),

    'BP Diastolic': np.random.randint(60, 120, size=100),

    'Cholesterol': np.random.randint(100, 300, size=100),

    'Disease': np.random.choice([0, 1], size=100)  # Binary target variable

})

# Splitting the dataset into features and target variable

X = data.drop('Disease', axis=1)

y = data['Disease']

# Standardize the 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=0.3, random\_state=42)

# Initialize models

models = {

    "Logistic Regression": LogisticRegression(random\_state=42),

    "Decision Tree": DecisionTreeClassifier(random\_state=42),

    "Random Forest": RandomForestClassifier(random\_state=42),

    "SVM": SVC(kernel='linear', random\_state=42)

}

# Dictionary to store evaluation metrics

metrics = {

    "Model": [],

    "Accuracy": [],

    "Precision": [],

    "Recall": [],

    "F1-Score": []

}

# Train, predict and evaluate each model

for model\_name, model in models.items():

    # Train the model

    model.fit(X\_train, y\_train)

    # Predict on the test set

    y\_pred = model.predict(X\_test)

    # Calculate metrics

    accuracy = accuracy\_score(y\_test, y\_pred)

    precision = precision\_score(y\_test, y\_pred)

    recall = recall\_score(y\_test, y\_pred)

    f1 = f1\_score(y\_test, y\_pred)

    # Store metrics

    metrics["Model"].append(model\_name)

    metrics["Accuracy"].append(accuracy)

    metrics["Precision"].append(precision)

    metrics["Recall"].append(recall)

    metrics["F1-Score"].append(f1)

    # Print classification report

    print(f"\nClassification Report for {model\_name}:\n")

    print(classification\_report(y\_test, y\_pred))

Classification Report for SVM:

precision recall f1-score support

0 0.67 0.50 0.57 16

1 0.56 0.71 0.63 14

accuracy 0.60 30

macro avg 0.61 0.61 0.60 30

weighted avg 0.61 0.60 0.60 30

# Convert metrics dictionary to DataFrame for easier viewing

metrics\_df = pd.DataFrame(metrics)

# Print the evaluation metrics for each model

print("\nModel Performance Metrics:")

print(metrics\_df)

Model Performance Metrics:

Model Accuracy Precision Recall F1-Score

0 SVM 0.6 0.555556 0.714286 0.625

import pandas as pd

import numpy as np

from sklearn.model\_selection import KFold, cross\_val\_score, StratifiedKFold

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

# Sample dataset (replace with your dataset)

data = pd.DataFrame({

    'Age': np.random.randint(18, 80, size=100),

    'BMI': np.round(np.random.uniform(15, 40, size=100), 1),

    'BP Systolic': np.random.randint(90, 180, size=100),

    'BP Diastolic': np.random.randint(60, 120, size=100),

    'Cholesterol': np.random.randint(100, 300, size=100),

    'Disease': np.random.choice([0, 1], size=100)  # Binary target variable

})

# Splitting the dataset into features and target variable

X = data.drop('Disease', axis=1)

y = data['Disease']

# Standardize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Define the models

models = {

    "Logistic Regression": LogisticRegression(random\_state=42),

    "Decision Tree": DecisionTreeClassifier(random\_state=42),

    "Random Forest": RandomForestClassifier(random\_state=42),

    "SVM": SVC(kernel='linear', random\_state=42)

}

# Number of folds

K = 5

# Perform K-Fold Cross-Validation

for model\_name, model in models.items():

    kfold = KFold(n\_splits=K, shuffle=True, random\_state=42)

    cv\_results = cross\_val\_score(model, X\_scaled, y, cv=kfold, scoring='accuracy')

    print(f"{model\_name}: Mean Accuracy = {cv\_results.mean():.4f}, Std = {cv\_results.std():.4f}")

Logistic Regression: Mean Accuracy = 0.5000, Std = 0.0894

Decision Tree: Mean Accuracy = 0.6200, Std = 0.0980

Random Forest: Mean Accuracy = 0.4700, Std = 0.0812

SVM: Mean Accuracy = 0.5100, Std = 0.0800

# Perform Stratified K-Fold Cross-Validation

for model\_name, model in models.items():

    skfold = StratifiedKFold(n\_splits=K, shuffle=True, random\_state=42)

    cv\_results = cross\_val\_score(model, X\_scaled, y, cv=skfold, scoring='accuracy')

    print(f"{model\_name} (Stratified): Mean Accuracy = {cv\_results.mean():.4f}, Std = {cv\_results.std():.4f}")

Logistic Regression (Stratified): Mean Accuracy = 0.5000, Std = 0.1304

Decision Tree (Stratified): Mean Accuracy = 0.5000, Std = 0.1049

Random Forest (Stratified): Mean Accuracy = 0.4800, Std = 0.0678

SVM (Stratified): Mean Accuracy = 0.5400, Std = 0.1428

import numpy as np

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

# Sample dataset (replace with your dataset)

data = pd.DataFrame({

    'Age': np.random.randint(18, 80, size=100),

    'BMI': np.round(np.random.uniform(15, 40, size=100), 1),

    'BP Systolic': np.random.randint(90, 180, size=100),

    'BP Diastolic': np.random.randint(60, 120, size=100),

    'Cholesterol': np.random.randint(100, 300, size=100),

    'Disease': np.random.choice([0, 1], size=100)  # Binary target variable

})

# Splitting the dataset into features and target variable

X = data.drop('Disease', axis=1)

y = data['Disease']

# Standardize the 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=0.3, random\_state=42)

# Define the hyperparameter grids for each model

param\_grid\_logreg = {

    'C': [0.01, 0.1, 1, 10, 100],

    'solver': ['liblinear', 'lbfgs']

}

param\_grid\_rf = {

    'n\_estimators': [100, 200, 300],

    'max\_depth': [None, 10, 20, 30],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 4]

}

param\_grid\_svm = {

    'C': [0.1, 1, 10, 100],

    'kernel': ['linear', 'rbf'],

    'gamma': ['scale', 'auto']

}

# Initialize the models

logreg = LogisticRegression(random\_state=42)

rf = RandomForestClassifier(random\_state=42)

svm = SVC(random\_state=42)

# Set up GridSearchCV for each model

grid\_logreg = GridSearchCV(logreg, param\_grid\_logreg, cv=5, scoring='accuracy')

grid\_rf = GridSearchCV(rf, param\_grid\_rf, cv=5, scoring='accuracy')

grid\_svm = GridSearchCV(svm, param\_grid\_svm, cv=5, scoring='accuracy')

# Fit the models and find the best hyperparameters

grid\_logreg.fit(X\_train, y\_train)

grid\_rf.fit(X\_train, y\_train)

grid\_svm.fit(X\_train, y\_train)

# Best hyperparameters and corresponding score

best\_logreg = grid\_logreg.best\_estimator\_

best\_rf = grid\_rf.best\_estimator\_

best\_svm = grid\_svm.best\_estimator\_

print("Best Logistic Regression Model:", best\_logreg)

print("Best Logistic Regression Parameters:", grid\_logreg.best\_params\_)

print("Best Logistic Regression Accuracy:", grid\_logreg.best\_score\_)

print("\nBest Random Forest Model:", best\_rf)

print("Best Random Forest Parameters:", grid\_rf.best\_params\_)

print("Best Random Forest Accuracy:", grid\_rf.best\_score\_)

print("\nBest SVM Model:", best\_svm)

print("Best SVM Parameters:", grid\_svm.best\_params\_)

print("Best SVM Accuracy:", grid\_svm.best\_score\_)

Best Logistic Regression Model: LogisticRegression(C=0.01, random\_state=42)

Best Logistic Regression Parameters: {'C': 0.01, 'solver': 'lbfgs'}

Best Logistic Regression Accuracy: 0.5857142857142857

Best Random Forest Model: RandomForestClassifier(random\_state=42)

Best Random Forest Parameters: {'max\_depth': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 100}

Best Random Forest Accuracy: 0.5428571428571429

Best SVM Model: SVC(C=0.1, kernel='linear', random\_state=42)

Best SVM Parameters: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}

Best SVM Accuracy: 0.5857142857142857