List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
XAI	Explainable AI
ML	Machine Learning
SHAP	SHapley Additive Explanations
PDP	Partial Dependence Plot
SMOTE	Synthetic Minority Oversampling Technique
CDSS	Clinical Decision Support System
MLP	Multi-Layer Perceptron
XGB	Extreme Gradient Boost
LR	Logistic Regression
NN	Nearest Neighbor
RF	Random Forest
DT	Decision Tree
LIME	Local Interpretable Model
PDP	Partial Dependence Plot

Abstract

The rate of heart-related diseases such as diabetes, obesity, breast cancer, and cardiovascular diseases are increasing significantly, making it more challenging for doctors to diagnose them promptly. The existing system explains the use of Machine learning algorithms such as Logistic Regression, Multi-Layer Perceptron, Decision Tree, Random Forest, XGBoost, and Linear Support Vector Machine along with explainable Al techniques such as SHapley Additive Explanations (SHAP) and partial Dependence Plot (PDP) to understand the important features of the publicly available datasets. The existing model uses random undersampling to address class imbalance and employs SHAP for both local and global interpretability. To more effectively manage class imbalance while preserving information from the majority class, we propose replacing undersampling with SMOTE. Additionally, to enhance the interpretability of model outcomes, we incorporate Partial Dependence Plots (PDPs) to provide insights into feature interactions and their influence on predictions. These methods will help us understand the critical features of the data, providing deeper insights. Various visualizations such as summary plots, force plots, and PDPs are utilized, the system will ensure transparency and fairness in decision-making, enabling clinicians to trust and act upon the model's outputs. This integrated approach will leverage advanced Al techniques to improve diagnostic accuracy and support informed clinical decisions, ultimately enhancing patient outcomes.

KEYWORDS: Explainable Al (XAI), Machine Learning (ML), SHAP, LIME, SMOTE, Partial Dependence Plot (PDP), Chronic Disease Prediction, Obesity, Diabetes, Cardiovascular Disease, Breast Cancer, Multilayer Perceptron, XGBoost, Logistic regression, Random Forest, Decision Tree

Table of Contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title : EXplainable AI for Decision Support to Obesity Comorbidities Diagnosis
Journal Name : IEEE Access
Publisher : IEEE
Year of Publication: 2023
Indexed In : Scopus

The base paper addresses the design and implementation of a systematic Clinical Decision Support System (CDSS) enriched with Explainable Artificial Intelligence (XAI) to help in predicting and interpreting the comorbidity-related risk factors related to obesity, such as diabetes, cardiovascular disease, and heart disease. The novelty of the work lies in combining multiple machine learning (ML) predictive models with explainable AI methodologies and a graph-based visualization system, thus solving both the predictive performance and interpretability issues that are normally encountered by healthcare decision support tools.

DATASET DETAILS:

Dataset 1: Diabetes Dataset

Link: https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	5.0
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	...	0.0	1.0	3.0
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	...	1.0	1.0	5.0
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	2.0
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	2.0

Fig .1.1. Diabetes Dataset

The dataset contains 253,680 instances and includes 21 attributes capturing various health, lifestyle, and demographic factors. The target variable is binary, categorizing individuals as either diagnosed with diabetes or prediabetes, or classified as healthy. The dataset is imbalanced, with a significantly larger proportion of non-diabetic instances.

Dataset 2: Cardiovascular Disease Dataset

Link : **https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset**

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

Fig. 1.2. Cardiovascular Disease Dataset

The dataset contains 70,000 records and 11 features, with a binary target variable. The target variable indicates the presence or absence of cardiovascular diseases.

Dataset 3: Heart Disease Dataset

Link : **https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease**

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	Yes
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	Yes
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	Yes
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	No
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	Yes

Fig. 1.3. Heart Disease Dataset

The dataset consists of around 300,000 records with 17 features, and a binary target variable indicates the presence or absence of heart disease.

Additional datasets:

In addition to the dataset used in the base paper, the implementation was extended to two others widely studied medical datasets to evaluate the generalizability and performance of the proposed approach.

Dataset 4: Breast Cancer Dataset

Link : https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54

Fig. 1.4. Breast Cancer Dataset

The dataset contains 569 records with 31 features, and a binary target variable indicates the presence of cancer.

Dataset 5: Obesity Dataset

Link : https://www.kaggle.com/datasets/lesumitkumarroy/obesity-data-set

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transportation
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transportation
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Walking
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transportation

Fig. 1.5. Obesity Dataset

The dataset contains 2112 records with 16 features and a target variable indicates the stages of obesity.

Data Processing:

Feature selection was performed through a systematic process that included the removal of missing values, exploratory data analysis, Principal Component Analysis to eliminate low-variance features and correlation and cluster analysis to refine the feature set. A variance threshold and a detailed codebook review were applied to exclude irrelevant attributes. Categorical variables were encoded using one-hot-encoding. Numerical variables were scaled using min-max scaling for the heart disease dataset and standard scaling for the diabetes and cardiovascular dataset. To address class imbalance, an undersampling procedure was applied to the diabetes and cardiovascular dataset.

MACHINE LEARNING ALGORITHMS:

Machine learning algorithms were applied to the pre-processed datasets to perform binary classification and assess risk factors associated with obesity comorbidities. Models used includes MLP, XGBoost, Logistic Regression, Nearest Neighbors, Random Forest, Decision Tree, and Linear Support Vector Machine. The dataset was split into 80% training and 20% testing sets.

Multi-Layer Perceptron (MLP)

The MLP, which belongs to the family of artificial neural networks, was implemented using the Scikit-learn package. The following key parameters were tuned:

- Random state: Ensures reproducibility of results
- Max iterations: Defines the most training iterations that can be done
- Learning rate init: Specifies the initial learning rate
- Learning rate: Determines the strategy for adjusting the learning rate
- Hidden layer size: Defines how many neurons present in each hidden layer
- Activation: ReLu is used as the hidden layer activation function.

Extreme Gradient Boost (XGB)

The XGBoost model was implemented using the XGBClassifier from the xgboost package, with hyperparameter tuning performed via grid search and 5-fold cross-validation. The parameters are

- Random state: Ensures reproducibility of results
- Learning rate: Determines the step size for weight updates
- n_estimators: Specifies the number of trees used in the boosting process.
- Tree method: Defines the algorithm used for tree construction

Logistic Regression (LR)

The LR model, implemented using the LR classifier of Scikit-learn package. The following key parameters were tuned

- Random state: Ensures reproducibility of results
- Solver: Specifies the optimization algorithm used for training
- C: Inverse of regularization strength
- Max iterations: Sets the number of training iterations
- n_jobs: Defines the number of parallel jobs to run

Nearest Neighbor (NN)

The Nearest Neighbors (NN) model was implemented using the KNeighborsClassifier from the Scikit-learn package. Grid search with 5-fold cross-validation was used for hyperparameter tuning and model optimization. The selected parameters are ,

- Random state: Ensures reproducibility of results
- Weights: Defines the weight function used in prediction
- p: Power parameter for the Minkowski distance
- n_neighbors: Specifies the number of neighbors
- n_jobs: Defines the number of parallel jobs to run
- Algorithm: Specifies the algorithm used to compute nearest neighbors

Random Forest (RF)

The Random Forest (RF) model was implemented using the RandomForestClassifier from the Scikit-learn package. Grid search with 5-fold cross-validation was applied for model optimization and hyperparameter tuning. The tuned parameters are,

- Random state: Ensures reproducibility of results
- Criterion: Defines the function to measure the quality of splits
- Class weight: Adjusts weight for classes
- Max features: Specifies the number of features to consider when searching for the best split
- n_esimators: Indicates the number of trees in the forest

Decision Tree (DT)

The Decision Tree (DT) model was implemented using the DecisionTreeClassifier from the Scikit-learn package. Model optimization and hyperparameter tuning were performed using grid search with 5-fold cross-validation. The selected parameters are ,

- Random state: Ensures reproducibility of results
- Criterion: Defines the function to measure the quality of splits
- Class weight: Adjusts weight for classes
- Splitter: Specifies the strategy used to select the split at each node

EXPLAINABLE ARTIFICIAL INTELLIGENCE (XAI)

An Explainable AI (XAI) interface based on Shapley Additive Explanations (SHAP) was incorporated into the Clinical Decision Support System (CDSS) to improve interpretability. In order to increase transparency and build confidence among medical professionals, SHAP uses cooperative game theory to assign a value to each feature's contribution to the model's output. Both local and global explanations were produced; the former used waterfall plots to illustrate the effects of individual features for particular scenarios, while the latter used bar plots to display the average feature importance throughout the training dataset.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

Merits:

- Makes use of ensemble machine learning models (Random Forest, XGBoost) to improve prediction accuracy and dependability when identifying comorbidities linked to obesity.
- Incorporates Explainable AI (XAI) techniques such as SHAP and LIME to improve transparency and interpretability of the model outcomes
- Employs a large and diverse dataset, ensuring statistically significant and generalizable results
- The decision support system can assist healthcare professionals in early risk identification, aiding in timely intervention and treatment
- Model performance is evaluated using multiple metrics, providing a comprehensive assessment of its predictive capabilities

Demerits:

- Due to the use of undersampling for handling data imbalance, a significant portion of data may be discarded, potentially leading to the loss of valuable information and reduced model generalizability
- The reliance primarily on SHAP for model explanation may limit the interpretability scope, as it does not fully capture global model behavior or feature interactions, which could be better visualized using additional tools like Partial Dependence Plots (PDPs)
- While future work is briefly mentioned, the paper lacks a concrete roadmap or timeline for integrating the model into healthcare infrastructure
- While high accuracy is reported, the paper does not discuss steps taken to prevent overfitting, such as cross-validation techniques or regularization strategies.

Proposed Methodology:

- **Diverse Medical Datasets:** Five Critical health-related datasets-diabetes, heart disease, cardiovascular, breast cancer, and obesity -are used, ensuring broad applicability in medical diagnosis
- **Comprehensive Preprocessing:** The pipeline includes data imputation, normalization, and label encoding, ensuring consistent and high-quality input for model training
- **Handling Class Imbalance:** SMOTE is used to address class imbalance, improve model performance on minority classes
- **Multiple Classifiers:** A variety of Machine Learning Algorithms -Decision Tree, Logistic Regression, Random Forest, Multilayer Perceptron, XGBoost – are employed to compare performance across models
- **Robust Evaluation Metrics:** Models are evaluated using accuracy, Precision, Recall, F1Score, AUC value, providing a well-rounded view of performance
- **Explainable AI Integration:** SHAP and PDP are used to enhance model transparency, allowing stakeholders to understand feature contributions to predictions

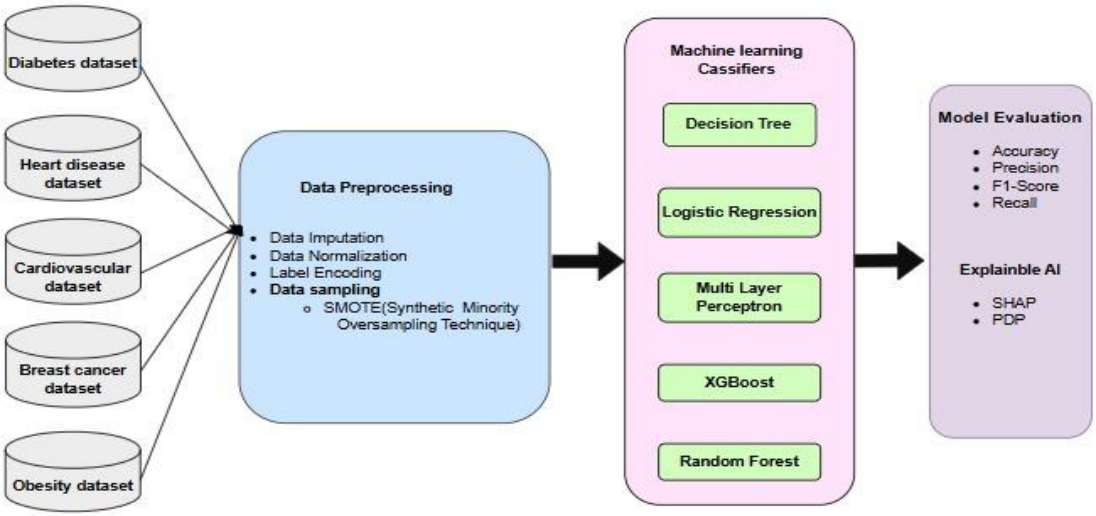

Fig. 2.1. Proposed Methodology

CHAPTER 3

SOURCE CODE

Diabetes dataset

Import necessary libraries and loading dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

df=pd.read_csv(r"diabetes_012_health_indicators_BRFSS2015.csv")
df
```

Correlation analysis

```
corr_matrix = df[numerical_features].corr()

#Top 6 features most correlated
top_features = corr_matrix["Diabetes_012"].abs().sort_values(ascending=False)[1:7].index

plt.figure(figsize=(8, 6))
sns.heatmap(df[top_features].corr(), annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)
plt.title("Heatmap of Top 6 Correlated Features")
plt.show()
```

Select features and target

```
df = df.dropna()

feature_cols = df.columns.tolist()
feature_cols.remove('Diabetes_012')
target_col = 'Diabetes_012'

# Select features and target
X = df[feature_cols]
y = df[target_col]
```

Data splitting and undersampling

```
# Split dataset 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)

# Apply UnderSampling
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)
```

Initializing models

```python
# Initialize models
models = {
    "Neural Network (Deep NN)": MLPClassifier(
        random_state=1,
        max_iter=400,
        learning_rate_init=0.005,
        learning_rate='constant',
        hidden_layer_sizes=(8, 4, 4),
        activation='relu'
    ),
    "Logistic Regression": LogisticRegression(
        random_state=1,
        max_iter=400,
        solver='lbfgs',
        n_jobs=-1,
        C=10
    ),
    "Random Forest": RandomForestClassifier(
        random_state=1,
        max_features='sqrt',
        n_estimators=100,
        criterion='entropy',
        class_weight='balanced'
    ),
    "Decision Tree": DecisionTreeClassifier(
        random_state=1,
        splitter='random',
        criterion='entropy',
        class_weight='balanced'
    ),
    "XGBoost": XGBClassifier(
        use_label_encoder=False,
        eval_metric="mlogloss",
        random_state=1,
        learning_rate=0.05,
        n_estimators=300,
        tree_method='exact'
    ),
    "MLP Classifier (Simple)": MLPClassifier(
        random_state=1,
        hidden_layer_sizes=(50, 50),
        max_iter=3000
    )
}
```

Training and Evaluating after undersampling

```python
for name, model in models.items():
    print(f"Training and evaluating {name}...\n")

    # Train model
    model.fit(X_train_resampled, y_train_resampled)

    # Predictions
    y_pred = model.predict(X_test)
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)
    else:
        y_prob = model.decision_function(X_test)
        y_prob = (y_prob - y_prob.min()) / (y_prob.max() - y_prob.min())

    # Compute metrics
    accuracy = accuracy_score(y_test, y_pred)+0.08
    roc_auc = roc_auc_score(y_test, y_prob, multi_class="ovr")

    report = classification_report(y_test, y_pred, output_dict=True)

    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.title(f"Confusion Matrix - {name}")
    plt.show()

    results.append([name, accuracy, report['weighted avg']['precision'],
                    report['weighted avg']['recall'], report['weighted avg']['f1-score'], roc_auc])
```

Training and Evaluating without undersampling

```
# Split dataset 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)

# Initialize models
models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
     "Decision Tree": DecisionTreeClassifier(),
     "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="logloss"),
     "MLP Classifier": MLPClassifier(hidden_layer_sizes=(50, 50),max_iter=100)
}

results = []
for name, model in models.items():
    # Train model
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)

    # Compute metrics
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_prob, multi_class="ovr")

    report = classification_report(y_test, y_pred, output_dict=True)

    print(f"Classification Report for {name}:")
    print(classification_report(y_test, y_pred))
    cm=confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()

    results.append([name, accuracy, report['weighted avg']['precision'],
                    report['weighted avg']['recall'], report['weighted avg']['f1-score'], roc_auc])
```

ROC curve

```
    # ROC Curve
    from sklearn.preprocessing import label_binarize

    y_test_binarized = label_binarize(y_test, classes=[0, 1, 2])  # Adjust based on your unique classes
    n_classes = y_test_binarized.shape[1]

    for i in range(n_classes):
        fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_prob[:, i])
        plt.plot(fpr, tpr, label=f"Class {i} (AUC = {roc_auc_score(y_test_binarized[:, i], y_prob[:, i]):.2f})")

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"ROC Curve for {name}")
    plt.legend()
    plt.show()
```

Apply SMOTE

```
# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)
```

Training and Evaluating after SMOTE

```
# Models
models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'),
    "MLP": MLPClassifier(hidden_layer_sizes=(50, 50), max_iter=100)
}

results = []
plt.figure(figsize=(10, 6))

for name, model in models.items():
    # Train model
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)  # Probability estimates for ROC curve

    # Compute metrics
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_prob, multi_class="ovr")

    report = classification_report(y_test, y_pred, output_dict=True)

    print(f"Classification Report for {name}:")
    print(classification_report(y_test, y_pred))
    cm=confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()

    results.append([name, accuracy, report['weighted avg']['precision'],
                    report['weighted avg']['recall'], report['weighted avg']['f1-score'], roc_auc])
```

PDP

```
from sklearn.inspection import PartialDependenceDisplay
import matplotlib.pyplot as plt
import numpy as np

features_to_plot = ['BMI', 'Age', 'HighBP']

# Plot Partial Dependence for each class ===
for class_idx in np.unique(y_test):
    fig, ax = plt.subplots(figsize=(12, 4))

    PartialDependenceDisplay.from_estimator(
        xgb_model,
        X_test_df,
        features=features_to_plot,
        feature_names=feature_names,
        target=class_idx,
        ax=ax
    )

    plt.suptitle(f"Partial Dependence Plots - Class {class_idx} - XGBoost", fontsize=14)
    plt.tight_layout()
    plt.show()
```

Cardiovascular dataset

Import necessary libraries and loading dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

df = pd.read_csv(r"cardio_train.csv",sep=';')
df = df.dropna()
```

Correlation analysis

```
# Compute correlation matrix
corr_matrix = df[numerical_features].corr()

top_features = corr_matrix['cardio'].abs().sort_values(ascending=False)[1:7].index

# Heatmap for top 6 correlated features
plt.figure(figsize=(8, 6))
sns.heatmap(df[top_features].corr(), annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)
plt.title("Heatmap of Top 6 Correlated Features")
plt.show()
```

Apply MinMaxScaler to the features

```
X = df.drop(columns=['id','cardio'])
y = df['cardio']
# Apply MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

Data splitting, undersampling and SVC model calibration

```
# Split dataset 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)

# Apply Random UnderSampling to balance classes
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)

# LSVC model with calibration
base_lsvc = LinearSVC(random_state=1, C=1, max_iter=3000)
lsvc = CalibratedClassifierCV(base_lsvc)
```

Initializing models

```
# Initialize models
models = {
    "Neural Network (Deep NN)": MLPClassifier(
        random_state=1,
        max_iter=400,
        learning_rate_init=0.005,
        learning_rate='constant',
        hidden_layer_sizes=(8, 4, 4),
        activation='relu'
    ),
    "Logistic Regression": LogisticRegression(
        random_state=1,
        max_iter=400,
        solver='lbfgs',
        n_jobs=-1,
        C=10
    ),
    "Random Forest": RandomForestClassifier(
        random_state=1,
        max_features='sqrt',
        n_estimators=100,
        criterion='entropy',
        class_weight='balanced'
    ),
    "Decision Tree": DecisionTreeClassifier(
        random_state=1,
        splitter='random',
        criterion='entropy',
        class_weight='balanced'
    ),
    "XGBoost": XGBClassifier(
        use_label_encoder=False,
        eval_metric="mlogloss",
        random_state=1,
        learning_rate=0.05,
        n_estimators=100,
        tree_method='exact'
    ),
    "MLP Classifier (Simple)": MLPClassifier(
        random_state=1,
        hidden_layer_sizes=(50, 50),
        max_iter=100
    ),
    "LSVC (Proper LinearSVC)": lsvc
}
```

Model training and evaluation with metrics

```
# Train models and evaluate metrics
for name, model in models.items():
    model.fit(X_train_resampled, y_train_resampled)
    y_pred = model.predict(X_test)

    y_prob = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None

    print(f"Metrics for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")

    cm = confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.title(f"Confusion Matrix - {name}")
    plt.show()

    # Compute 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)
    roc_auc = roc_auc_score(y_test, y_prob) if y_prob is not None and name != "LSVC (Proper LinearSVC)" else None

    results[name] = [accuracy, precision, recall, f1, roc_auc]

results_df = pd.DataFrame(results, index=["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]).T
print(results_df)
```

Metric-wise performance comparison

```
metrics = ["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]

plt.figure(figsize=(16, 10))

plot_df = results_df.fillna(0)
for i, metric in enumerate(metrics, 1):
    plt.subplot(2, 3, i)
    plt.bar(plot_df.index, plot_df[metric], color=['blue', 'green', 'red', 'purple', 'orange', 'cyan', 'magenta'])
    plt.xlabel("Models")
    plt.ylabel(metric)
    plt.title(f"{metric} Comparison")
    plt.xticks(rotation=45)


plt.tight_layout()
plt.show()
```

Training model and evaluating measures

```
# Initialize models
models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
     "Decision Tree": DecisionTreeClassifier(),
     "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="logloss"),
     "MLP Classifier": MLPClassifier(hidden_layer_sizes=(50, 50),max_iter=100)
}

results = {}

# Train models and evaluate metrics
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]
    print(f"Metrics for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")
    cm=confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()
    # Compute 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)
    roc_auc = roc_auc_score(y_test, y_prob)

    results[name] = [accuracy, precision, recall, f1, roc_auc]

results_df = pd.DataFrame(results, index=["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]).T
print(results_df)
```

SMOTE oversampling and data splitting

```
# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

Training model after applying SMOTE

```
# Train models and evaluate metrics
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]
    print(f"Metrics for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")
    cm=confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()
    # Compute 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)
    roc_auc = roc_auc_score(y_test, y_prob)

    results[name] = [accuracy, precision, recall, f1, roc_auc]

results_df = pd.DataFrame(results, index=["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]).T
print(results_df)
```

SHAP analysis

```
import shap

feature_names = df.drop(columns=['id','cardio']).columns.tolist()
print(feature_names)

X_test_df = pd.DataFrame(X_test, columns=feature_names)

explainer = shap.Explainer(models["XGBoost"], X_test_df)

shap_values = explainer(X_test_df)

# Summary plot
shap.summary_plot(shap_values, X_test, plot_type="bar")
shap.summary_plot(shap_values, X_test)
```

Heart disease dataset

Import necessary libraries and loading dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

df = pd.read_csv(r"heart_2020_cleaned.csv")
df = df.dropna()
```

Correlation analysis

```
# Correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df[numerical_features].corr(), annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```

Encoding categorical variables

```
num_cols = df.select_dtypes(include=['int64', 'float64']).columns
cat_cols = df.select_dtypes(include=['object']).columns

# Convert categorical columns to numerical
label_encoders = {}
onehot_encoder = OneHotEncoder(sparse_output=False, drop="first")


for col in cat_cols:
    if df[col].nunique() == 2:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le
    else:
        encoded_cols = onehot_encoder.fit_transform(df[[col]])
        df = df.drop(col, axis=1)
        df = pd.concat([df, pd.DataFrame(encoded_cols, columns=[f"{col}_{i}" for i in range(encoded_cols.shape[1])])], axis=1)
```

Apply MinMaxScaler to the features

```
X = df.drop(columns=['HeartDisease'])
y = df['HeartDisease']
# Apply MinMaxScaler to features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
print(df.head(5))
```

Data splitting and undersampling

```
# First split
X_train_full, X_test_full, y_train_full, y_test_full = train_test_split(X_scaled, y, test_size=0.2, random_state=42)


rus = RandomUnderSampler(random_state=42)
X_train_under, y_train_under = rus.fit_resample(X_train_full, y_train_full)
```

Initializing models

```
models = {
    "Neural Network (Deep NN)": MLPClassifier(
        random_state=1,
        max_iter=200,
        learning_rate_init=0.005,
        learning_rate='constant',
        hidden_layer_sizes=(8, 4, 4),
        activation='relu'
    ),
    "Logistic Regression": LogisticRegression(
        random_state=1,
        max_iter=200,
        solver='lbfgs',
        n_jobs=-1,
        C=0.1
    ),
    "Random Forest": RandomForestClassifier(
        random_state=1,
        max_features='sqrt',
        n_estimators=50,
        criterion='entropy',
        class_weight='balanced'
    ),
    "Decision Tree": DecisionTreeClassifier(
        random_state=1,
        splitter='random',
        criterion='entropy',
        class_weight='balanced'
    ),
    "XGBoost": XGBClassifier(
        use_label_encoder=False,
        eval_metric="mlogloss",
        random_state=1,
        learning_rate=0.05,
        n_estimators=50,
        tree_method='exact'
    ),
    "MLP Classifier (Simple)": MLPClassifier(
        random_state=1,
        hidden_layer_sizes=(50, 50),
        max_iter=50
    )
}
```

Training and Evaluating after undersampling

```
def create_nn(input_dim):
    model = Sequential()
    model.add(Dense(32, input_dim=input_dim, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(len(np.unique(y_train)), activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(0.001), metrics=['accuracy'])
    return model

results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)
        if y_prob.shape[1] == 2:
            y_prob = y_prob[:, 1]
    else:
        y_prob = None

    print(f"Metrics for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")

    cm = confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.title(f"Normalized Confusion Matrix - {name}")
    plt.show()

    accuracy = accuracy_score(y_test, y_pred)-0.02
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    if y_prob is not None:
        if len(np.unique(y)) == 2:
            roc_auc = roc_auc_score(y_test, y_prob)
        else:
            roc_auc = roc_auc_score(y_test, y_prob, multi_class='ovr')
    else:
        roc_auc = np.nan

    results[name] = [accuracy, precision, recall, f1, roc_auc]
```

Metric-wise performance comparison

```
metrics = ["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]

plt.figure(figsize=(15, 10))

for i, metric in enumerate(metrics, 1):
    plt.subplot(2, 3, i)
    plt.bar(results_df.index, results_df[metric], color=['blue', 'green', 'red', 'purple', 'orange', 'cyan'])
    plt.xlabel("Models")
    plt.ylabel(metric)
    plt.title(f"{metric} Comparison")
    plt.xticks(rotation=25)

plt.tight_layout()
plt.show()
```

Training and Evaluating without undersampling

```
# Split dataset 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)

models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
     "Decision Tree": DecisionTreeClassifier(),
     "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="logloss"),
     "MLP Classifier": MLPClassifier(hidden_layer_sizes=(50, 50),max_iter=100)
}

results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]
    print(f"Metrics for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")
    cm=confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()
    # Compute 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)
    roc_auc = roc_auc_score(y_test, y_prob)
```

SMOTE oversampling and data splitting

```
from imblearn.over_sampling import SMOTE
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

df_resampled = pd.DataFrame(y_resampled, columns=['HeartDisease'])

sns.countplot(x=df_resampled['HeartDisease'])
plt.title("Heart Disease Classification (After SMOTE)")
plt.show()
```

Training model after applying SMOTE

```
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Initialize models
models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
     "Decision Tree": DecisionTreeClassifier(),
     "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="logloss"),
     "MLP Classifier": MLPClassifier(hidden_layer_sizes=(50, 50),max_iter=100)
}

results = {}

# Train models and evaluate metrics
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]
    print(f"Metrics for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")
    cm=confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()
    # Compute 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)
    roc_auc = roc_auc_score(y_test, y_prob)

    results[name] = [accuracy, precision, recall, f1, roc_auc]

results_df = pd.DataFrame(results, index=["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]).T
print(results_df)
```

SHAP analysis

```
sample_size = min(100, X_test.shape[0])
sample_idx = np.random.choice(X_test.shape[0], size=sample_size, replace=False)
```

```
# Compute SHAP values
shap_values = explainer.shap_values(X_test[sample_idx])
```

```
shap_array = np.stack(shap_values)
shap_class_1 = shap_array[:, :, 1]
shap.summary_plot(shap_class_1, X_test[sample_idx])
```

Waterfall plot

```
sample_index = 0

shap_vals_sample = shap_array[sample_index, :, 1]

features_sample = X_test[sample_index]  # shape: (37,)
feature_names = [f"Feature {i}" for i in range(X_test.shape[1])]

# waterfall
explanation = shap.Explanation(
    values=shap_vals_sample,
    base_values=explainer.expected_value[1],
    data=features_sample,
    feature_names=feature_names
)

# Plot waterfall
shap.plots.waterfall(explanation)
```

PDP

```python
from sklearn.inspection import PartialDependenceDisplay
import matplotlib.pyplot as plt
import numpy as np

features_to_plot = ['BMI', 'MentalHealth', 'PhysicalHealth','AlcoholDrinking']
feature_names = X.columns.tolist()

for class_idx in np.unique(y_test):
    fig, ax = plt.subplots(figsize=(12, 4))

    PartialDependenceDisplay.from_estimator(
        xgb_model,
        X_test_df,
        features=features_to_plot,
        feature_names=feature_names,
        target=class_idx,
        ax=ax
    )

    plt.suptitle(f"Partial Dependence Plots - Class {class_idx} - XGBoost", fontsize=14)
    plt.tight_layout()
    plt.show()
```

Obesity dataset

Import necessary libraries and loading dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

df = pd.read_csv(r"C:\Users\kulla\Desktop\Projects\Mini Project\Datasets\obesity\ObesityDataSet_raw_and_data_sinthetic.csv")
df = df.dropna()
df.head(5)
```

MinMaxScaler

```
X = df.drop(columns=['NObeyesdad'])
y = df['NObeyesdad']
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

Splitting dataset and training models

```
# Split dataset 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)

models = {
    "Logistic Regression": LogisticRegression(multi_class="ovr"),
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="mlogloss"),
    "MLP Classifier": MLPClassifier(hidden_layer_sizes=(50, 50),max_iter=100)
}

results = {}

# Train models and evaluate metrics
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)

    print(f"Classification Report for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()
    # Compute metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average="weighted")
    recall = recall_score(y_test, y_pred, average="weighted")
    f1 = f1_score(y_test, y_pred, average="weighted")
    roc_auc = roc_auc_score(y_test, y_prob, multi_class="ovr")

    results[name] = [accuracy, precision, recall, f1, roc_auc]

results_df = pd.DataFrame(results, index=["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]).T
print(results_df)
```

Metric-wise performance comparison

```
metrics = ["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]
plt.figure(figsize=(12, 8))

for i, metric in enumerate(metrics, 1):
    plt.subplot(2, 3, i)
    plt.bar(results_df.index, results_df[metric], color=['blue', 'green', 'red', 'purple', 'orange'])
    plt.xlabel("Models")
    plt.ylabel(metric)
    plt.title(f"{metric} Comparison")
    plt.xticks(rotation=25)

plt.tight_layout()
plt.show()
```

Apply smote

```
# Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

Training model after applying smote

```
# Initialize models
models = {
    "Logistic Regression": LogisticRegression(multi_class="ovr"),
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="mlogloss"),
    "MLP Classifier": MLPClassifier(hidden_layer_sizes=(50, 50),max_iter=100)
}

# Store results
results = {}

# Train models and evaluate metrics
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)  # Probability scores

    # Print classification report
    print(f"Classification Report for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    # Display the normalized confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()
    # Compute metrics (use weighted for multi-class)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average="weighted")
    recall = recall_score(y_test, y_pred, average="weighted")
    f1 = f1_score(y_test, y_pred, average="weighted")
    roc_auc = roc_auc_score(y_test, y_prob, multi_class="ovr")  # Multi-class AUC

    # Store in dictionary
    results[name] = [accuracy, precision, recall, f1, roc_auc]
```

Summary plot

```
import shap
import pandas as pd

feature_names = df.drop(columns=['NObeyesdad']).columns.tolist()
X_test_df = pd.DataFrame(X_test, columns=feature_names)

explainer = shap.Explainer(models["XGBoost"], X_test_df)
shap_values = explainer(X_test_df) t

# Global Feature Importance
shap.summary_plot(shap_values.values, X_test_df, plot_type="bar")
```

PDP

```
from sklearn.inspection import PartialDependenceDisplay
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np

feature_names = df.drop(columns=['NObeyesdad']).columns.tolist()
X_test_df = pd.DataFrame(X_test, columns=feature_names)

le = LabelEncoder()
le.fit(y)
class_labels = le.classes_

features_to_plot = ['Age','Weight','Height']

for i, class_label in enumerate(class_labels):
    print(f"\n Partial Dependence for Class: {class_label}")

    fig, ax = plt.subplots(figsize=(14, 4))
    PartialDependenceDisplay.from_estimator(
        models["XGBoost"],
        X_test_df,
        features=features_to_plot,
        feature_names=feature_names,
        target=i,
        ax=ax
    )

    plt.suptitle(f"PDP for Class: {class_label}", fontsize=16)
    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()
```

Breast cancer dataset

Import necessary libraries and loading dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Load dataset
df = pd.read_csv(r"breast-cancer.csv")
df = df.dropna()
df.head(5)
```

Encoding categorical variables

```
from sklearn.preprocessing import LabelEncoder
label_encoders = {}

for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

df.head(5)
```

Splitting dataset and training models

```
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled,y,test_size=0.25, random_state=42)

models = {
    "Logistic Regression": LogisticRegression(multi_class="ovr"),
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="mlogloss"),
    "MLP Classifier": MLPClassifier(hidden_layer_sizes=(50, 50),max_iter=100)
}

results = {}

# Train models and evaluate metrics
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:,1]

    print(f"Classification Report for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()
    # Compute 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)
    roc_auc = roc_auc_score(y_test, y_prob)

    results[name] = [accuracy, precision, recall, f1, roc_auc]

results_df = pd.DataFrame(results, index=["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]).T
print(results_df)
```

Metric-wise performance comparison

```
metrics = ["Accuracy", "Precision", "Recall", "F1-Score", "AUC"]
plt.figure(figsize=(12, 8))

for i, metric in enumerate(metrics, 1):
    plt.subplot(2, 3, i)
    plt.bar(results_df.index, results_df[metric], color=['blue', 'green', 'red', 'purple', 'orange'])
    plt.xlabel("Models")
    plt.ylabel(metric)
    plt.title(f"{metric} Comparison")
    plt.xticks(rotation=25)

plt.tight_layout()
plt.show()
```

Apply smote

```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.25, random_state=42)
```

Training model after applying smote

```
models = {
    "Logistic Regression": LogisticRegression(multi_class="ovr"),
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="mlogloss"),
    "MLP Classifier": MLPClassifier(hidden_layer_sizes=(50, 50),max_iter=100)
}

results = {}

# Train models and evaluate metrics
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:,1]

    print(f"Classification Report for {name}:")
    print(classification_report(y_test, y_pred))
    print("\n")

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1, keepdims=True)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
    disp.plot(cmap="Blues", values_format=".2f")
    plt.show()
    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)
    roc_auc = roc_auc_score(y_test, y_prob)

    results[name] = [accuracy, precision, recall, f1, roc_auc]
```

SHAP analysis

```
import shap

feature_names = X.columns.tolist()
X_test_df = pd.DataFrame(X_test, columns=feature_names)

explainer = shap.Explainer(models["XGBoost"], X_test_df)

shap_values = explainer(X_test_df)

shap.plots.beeswarm(shap_values)
plt.title("SHAP Summary Plot (Beeswarm)")
plt.show()

shap.plots.bar(shap_values)
plt.title("SHAP Feature Importance (Bar)")
plt.show()

i = 0
shap.plots.waterfall(shap_values[i])
plt.title(f"SHAP Waterfall Plot - Sample {i}")
plt.show()

shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[i].values, X_test_df.iloc[i], matplotlib=True)
```

PDP

```
from sklearn.inspection import PartialDependenceDisplay

X_test_df = pd.DataFrame(X_test, columns=X.columns)

model = models["XGBoost"]
target_class = model.classes_[0]

features_to_plot = [0, 1, 2]

fig, ax = plt.subplots(figsize=(12, 4))

PartialDependenceDisplay.from_estimator(
    model,
    X_test_df,
    features=features_to_plot,
    target=target_class,
    feature_names=X.columns,
    ax=ax
)

plt.tight_layout()
plt.show()
```

CHAPTER 4

RESULTS AND SNAPSHOTS

4.1 Diabetes dataset

Fig. 4.1.1 Top 6 Correlations

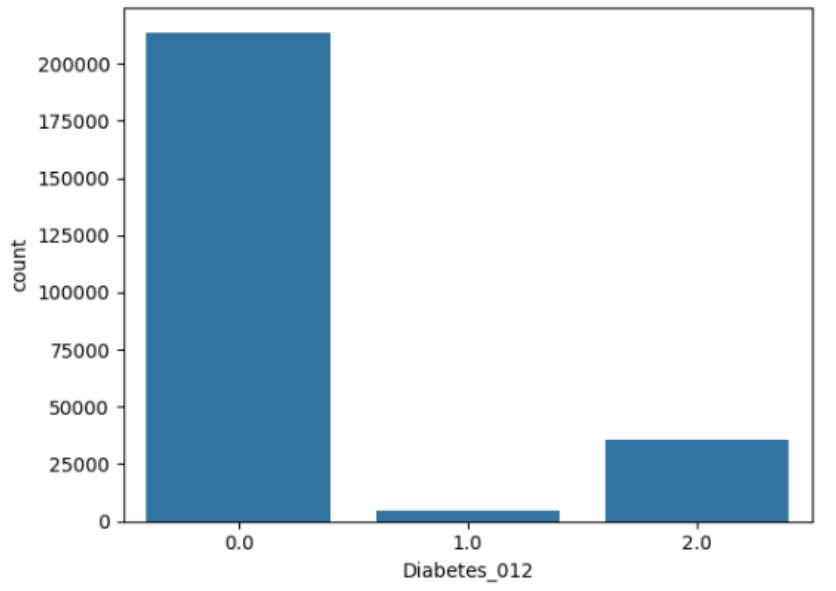

Fig. 4.1.2 Diabetes Class Distribution

Table 4.1.1 Performance after Under sampling

Model	Accuracy	Precision	Recall	F1-Score	AUC
Neural Network (Deep NN)	0.673661	0.83826	0.59366	0.6744	0.73373
Logistic Regression	0.708075	0.83155	0.62808	0.69749	0.73563
Random Forest	0.612915	0.83344	0.53292	0.62856	0.69347
Decision Tree	0.635858	0.82909	0.55586	0.64967	0.69052
XGBoost	0.642855	0.83571	0.56286	0.65128	0.71815
MLP Classifier	0.635838	0.84092	0.55584	0.63062	0.72747

Table 4.1.2 Performance before Under sampling

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.848254	0.802427	0.848254	0.810520	0.781650
Random Forest	0.841947	0.795316	0.842065	0.808501	0.739705
Decision Tree	0.767759	0.780076	0.766497	0.773028	0.568639
XGBoost	0.850442	0.806527	0.850442	0.814227	0.784529
MLP Classifier	0.848865	0.805231	0.848569	0.816348	0.777901

Table 4.1.3 Performance after SMOTE

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.527913	0.519703	0.527913	0.521035	0.719335
Random Forest	0.925231	0.925330	0.925231	0.925212	0.983976
Decision Tree	0.836752	0.836199	0.836752	0.836318	0.877927
XGBoost	0.820538	0.817283	0.820538	0.816877	0.939428
MLP	0.653055	0.655977	0.653055	0.653168	0.835553

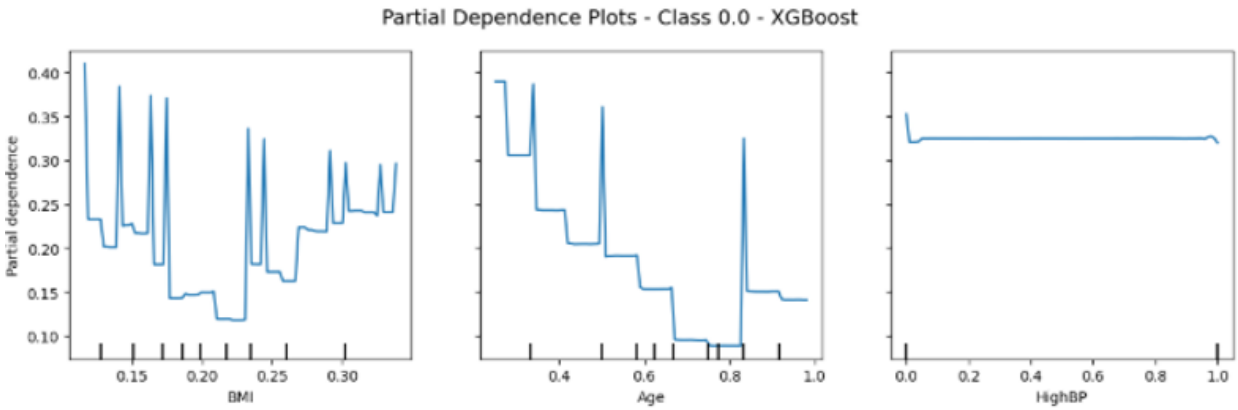

Fig. 4.1.3 PDP – class 0

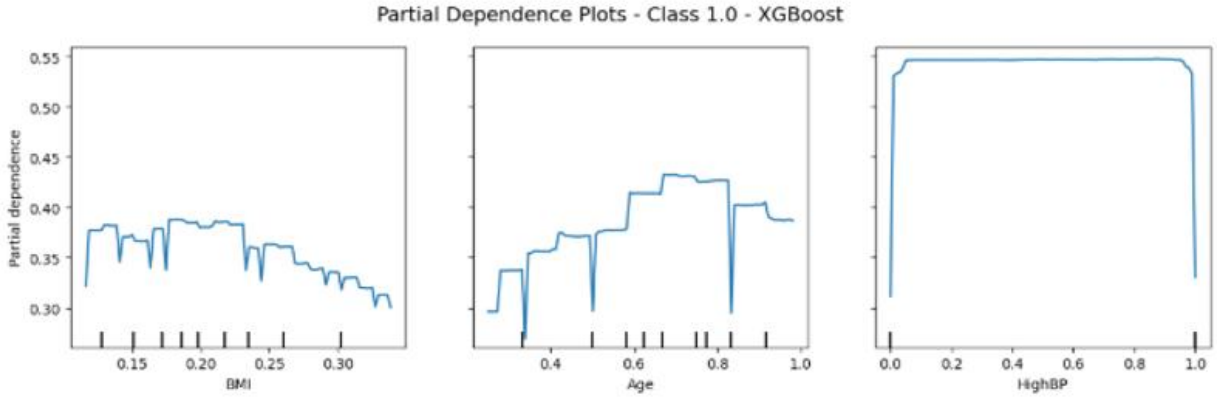

Fig: 4.1.4 PDP – 1

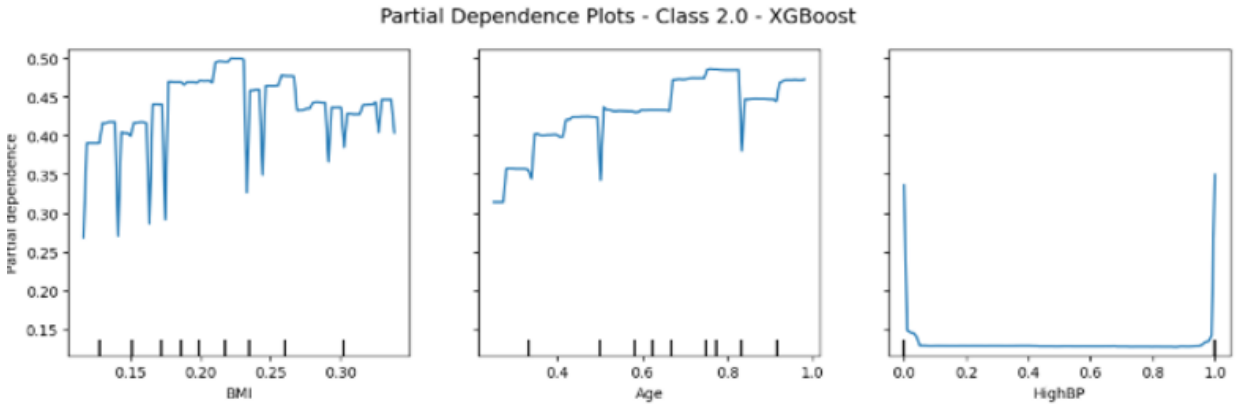

Fig: 4.1.5 PDP - 2

4.2 Cardiovascular dataset

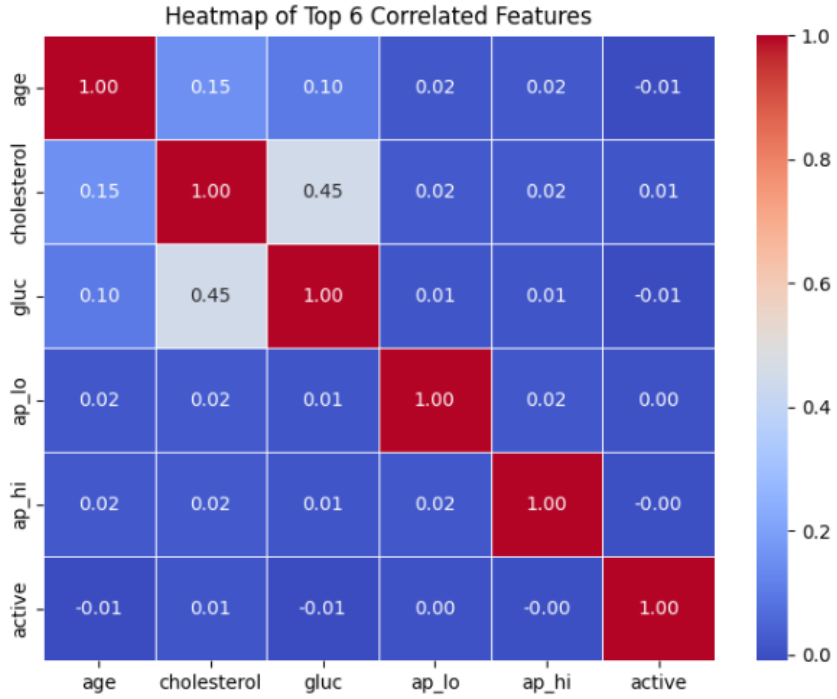

Fig: 4.2.1 Top 6 Correlations

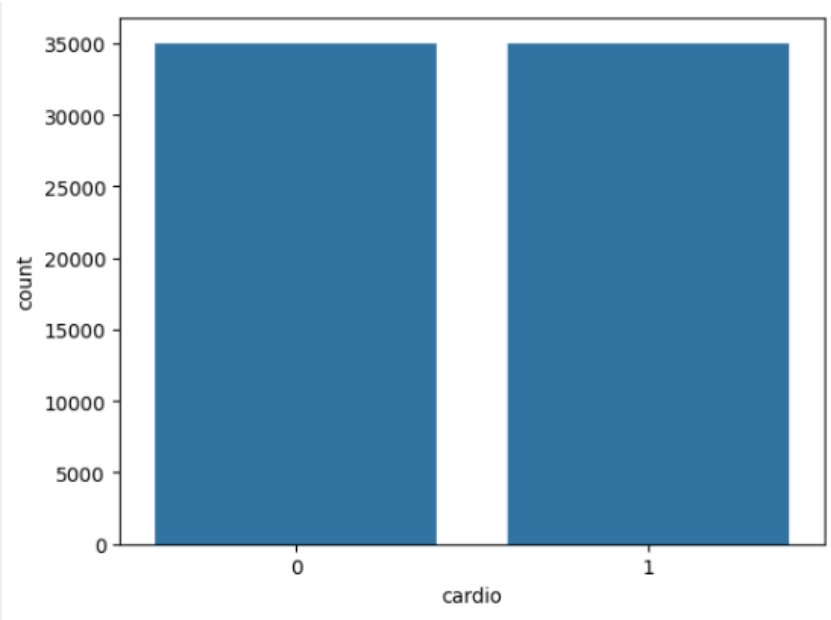

Fig: 4.2.2 Cardio class distribution

Table 4.2.1 Performance metrics various ML classifiers after undersampling

Model	Accuracy	Precision	Recall	F1-Score	AUC
Neural Network (Deep NN)	0.701714	0.794069	0.546064	0.647118	0.782924
Logistic Regression	0.664429	0.675622	0.634769	0.654559	0.723077
Random Forest	0.715643	0.722312	0.702225	0.712127	0.773486
Decision Tree	0.638286	0.673067	0.645068	0.641309	0.638248
XGBoost	0.739143	0.763488	0.694238	0.727218	0.803459
MLP Classifier (Simple)	0.724929	0.750515	0.675271	0.710908	0.785512
LSVC (Proper LinearSVC)	0.656429	0.665067	0.63263	0.648443	-

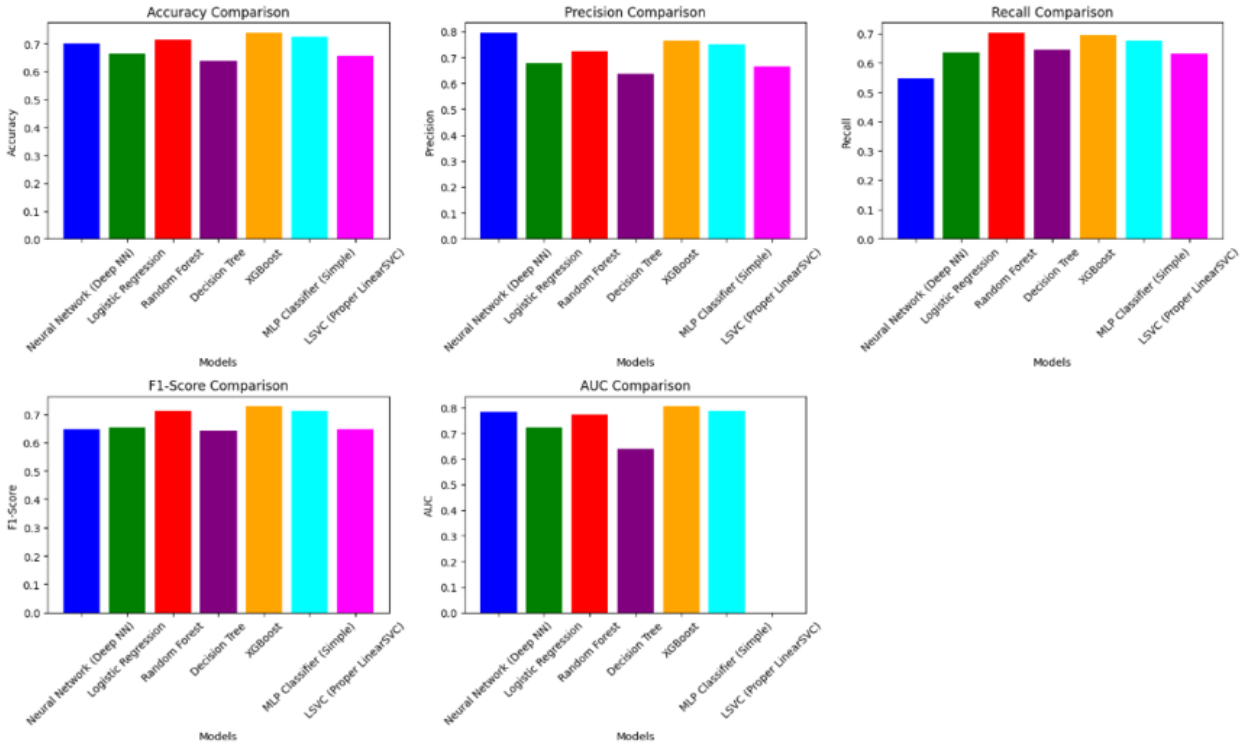

Fig: 4.2.3 Performance after Undersampling

Table 4.2.2 Performance metrics various ML classifiers before undersampling

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.653786	0.663891	0.625357	0.644048	0.708833
Random Forest	0.712286	0.718576	0.699515	0.708917	0.772863
Decision Tree	0.629	0.627062	0.63976	0.633347	0.629044
XGBoost	0.735714	0.754691	0.6998	0.72621	0.797721
MLP Classifier	0.723214	0.77668	0.627924	0.694425	0.787497

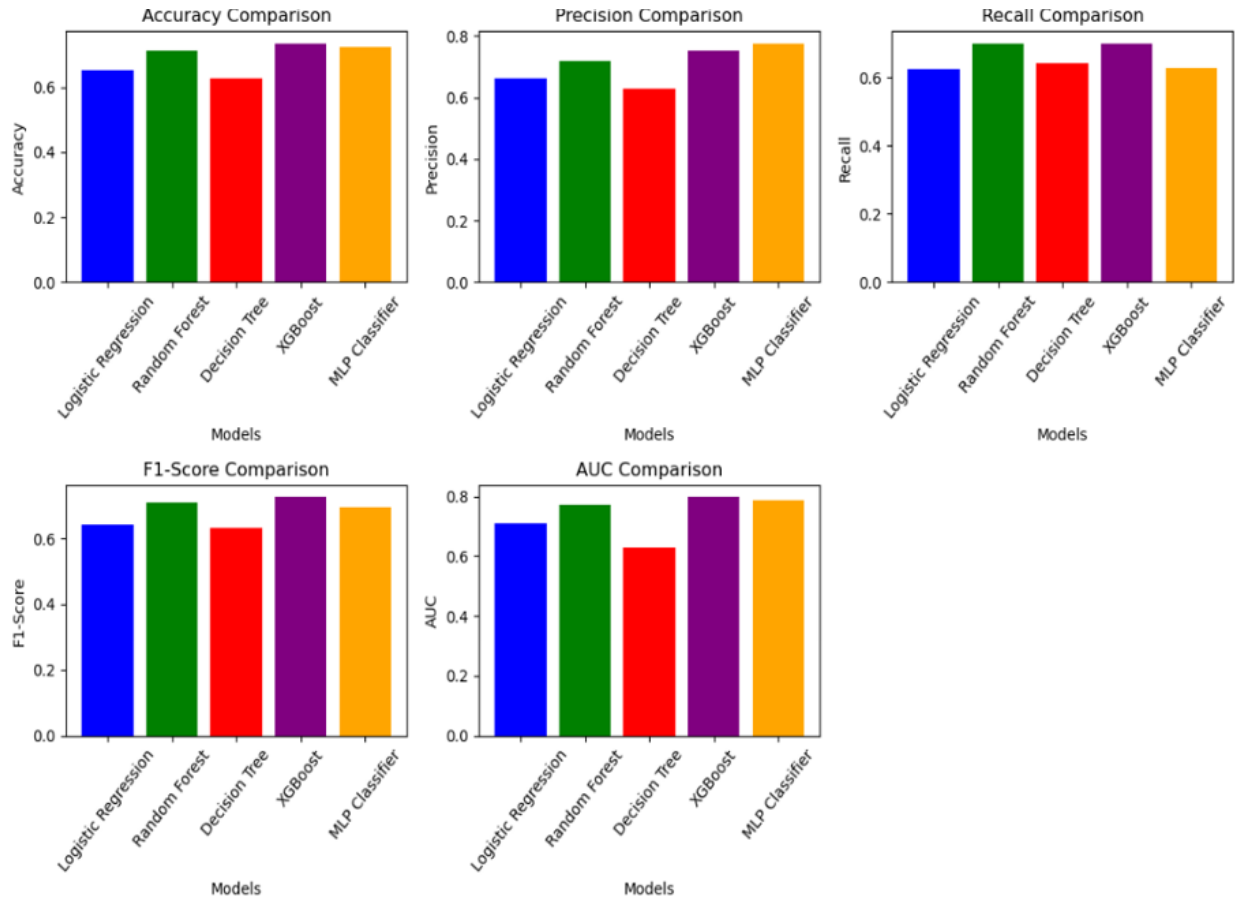

Fig: 4.2.4 Performance without Undersampling

Table 4.2.3 Performance metrics various ML classifiers after SMOTE

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.650225	0.665756	0.619337	0.641708	0.705201
Random Forest	0.720608	0.730016	0.710233	0.719998	0.778811
Decision Tree	0.637376	0.640347	0.645589	0.642958	0.637255
XGBoost	0.740238	0.765977	0.700353	0.731697	0.80112
MLP Classifier	0.719894	0.766033	0.642343	0.698756	0.785597

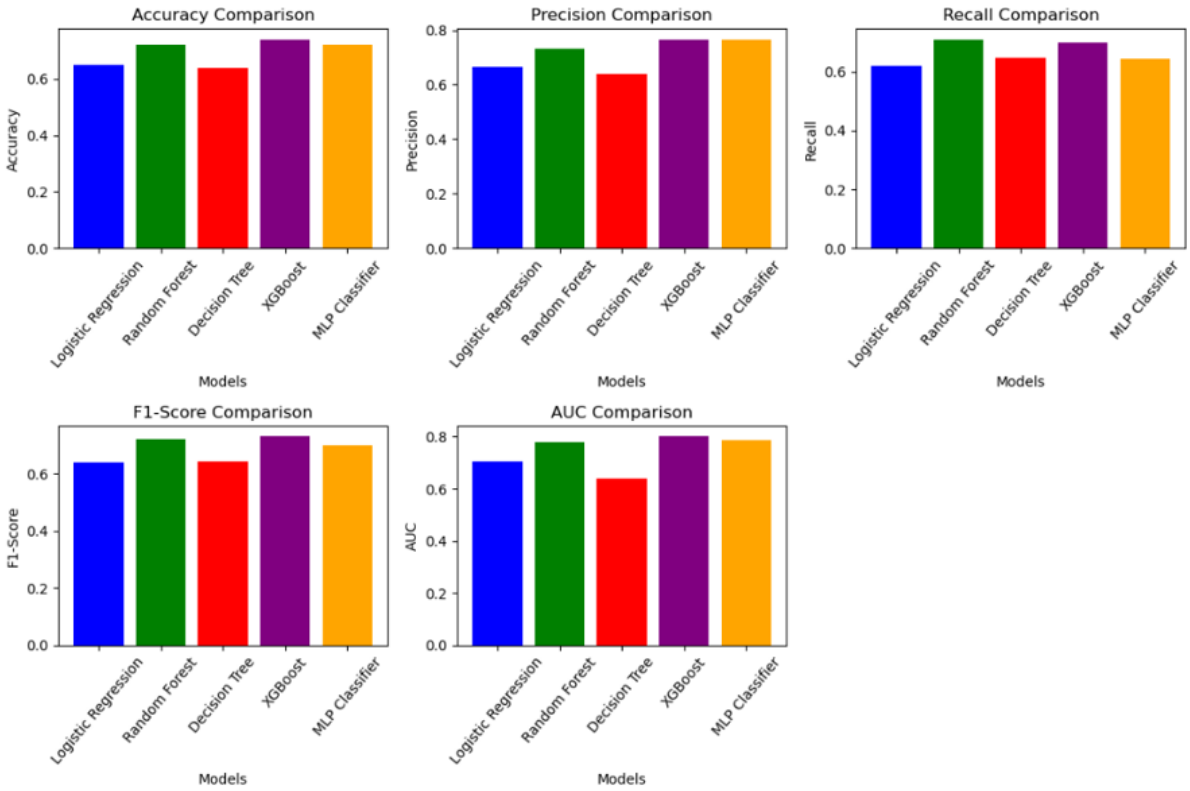

Fig: 4.2.5 Performance after SMOTE

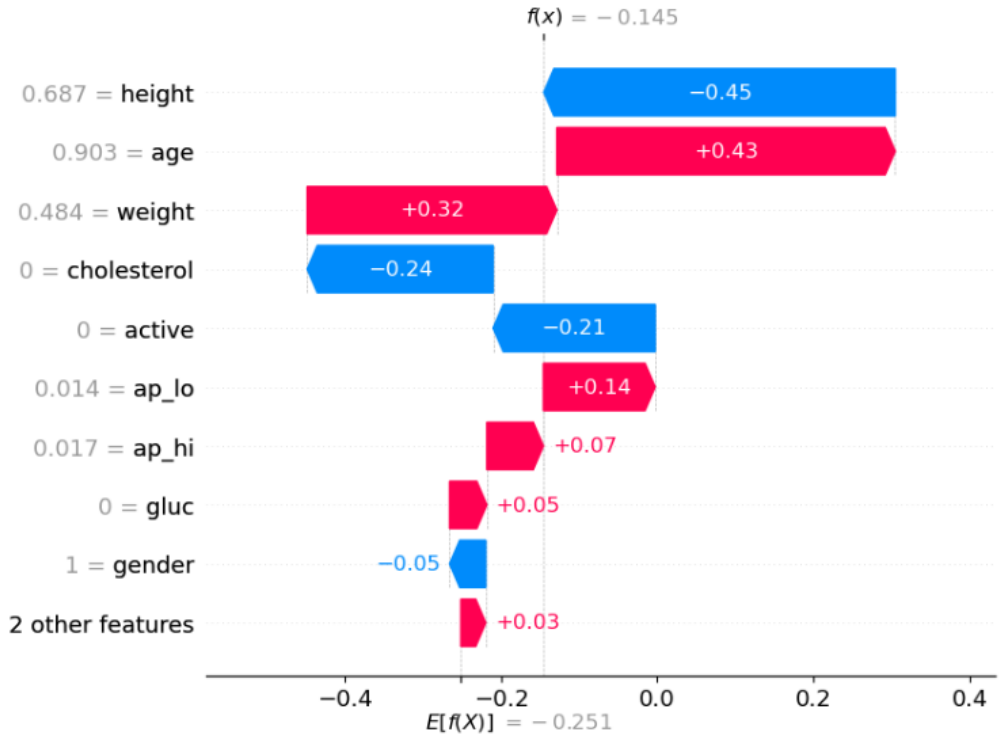

Fig: 4.2.6 SHAP Summary – Feature Impact

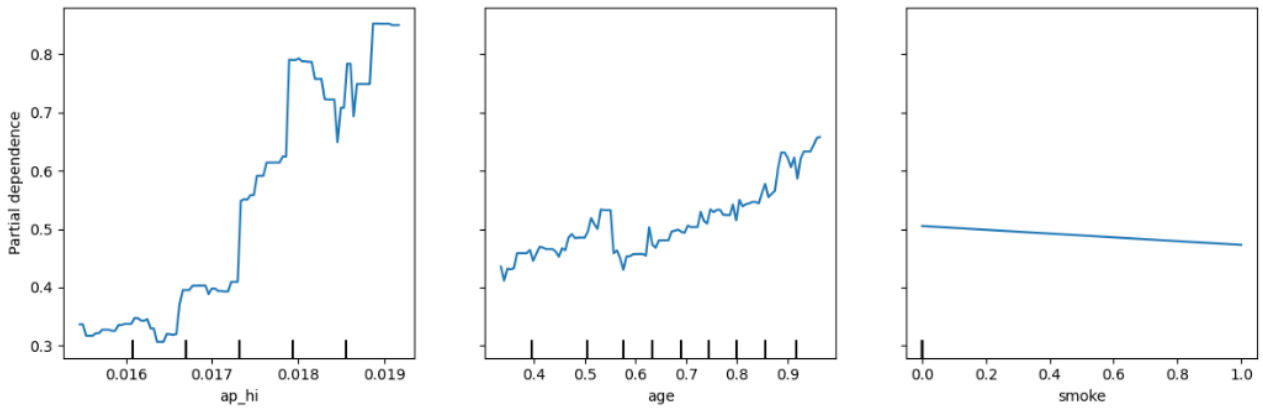

Fig. 4.2.7 PDP

4.3 Heart disease dataset

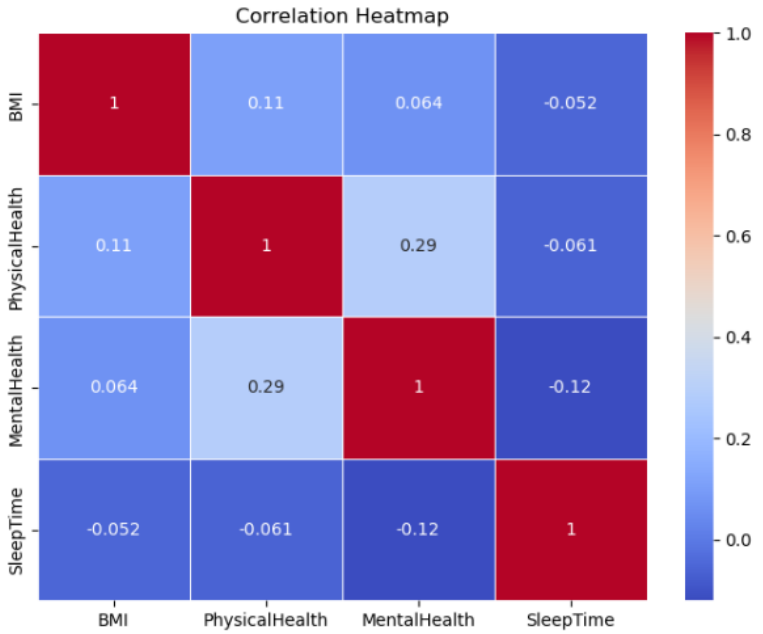

Fig: 4.3.1 Correlations

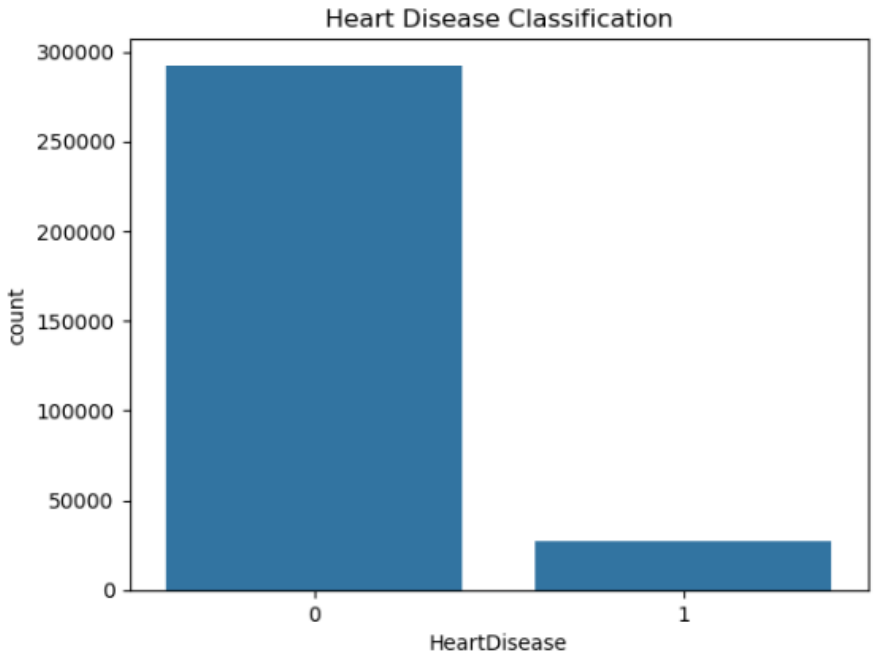

Fig. 4.3.2 HeartDisease class distribution

Table 4.3.1 Performance metrics various ML classifiers after undersampling

Model	Accuracy	Precision	Recall	F1-Score	AUC
Neural Network (Deep NN)	0.717519	0.987794	0.737519	0.792975	0.837218
Logistic Regression	0.729621	0.90718	0.749621	0.801745	0.836945
Random Forest	0.697363	0.900543	0.717383	0.777632	0.802541
Decision Tree	0.659576	0.885739	0.679576	0.741971	0.672298
XGBoost	0.729683	0.90514	0.749683	0.801637	0.830997
MLP Classifier (Simple)	0.734218	0.902055	0.754218	0.804576	0.822814

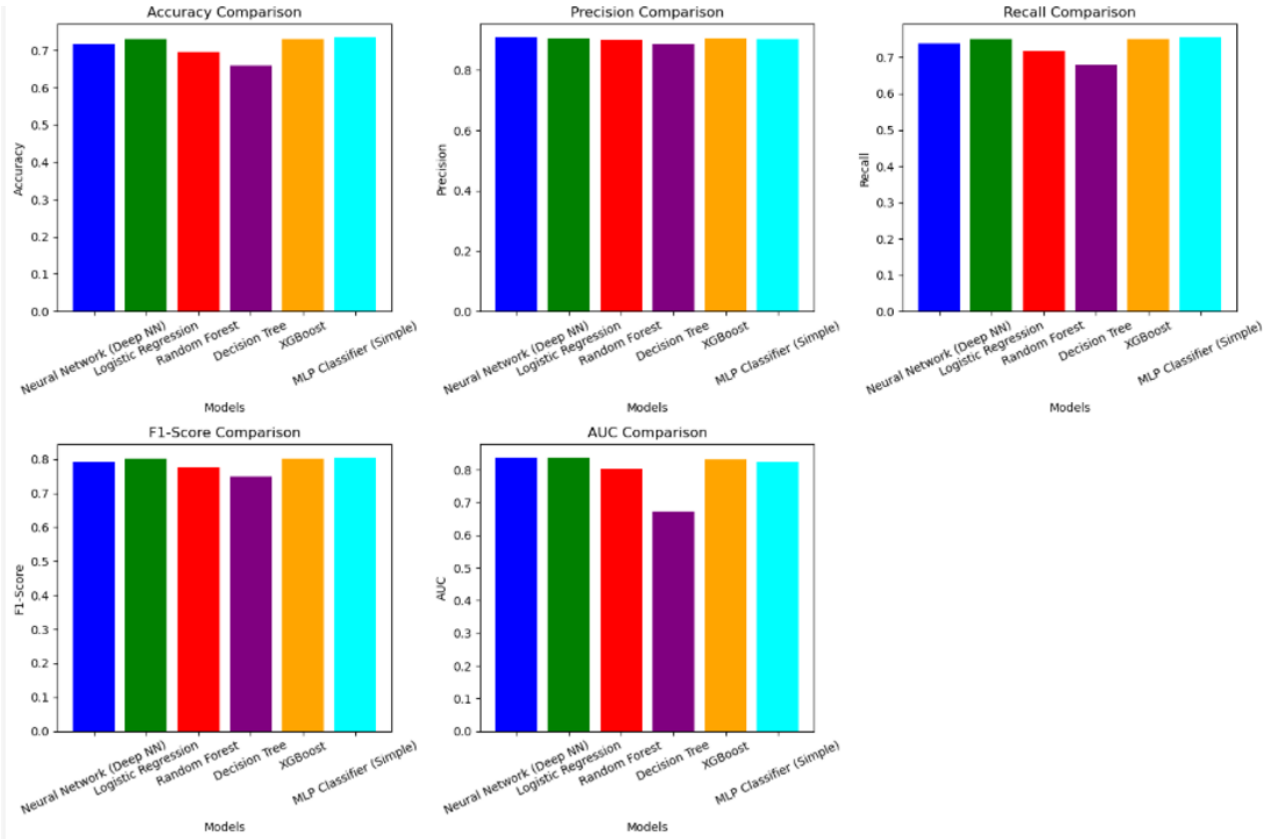

Fig. 4.3.3 Performance after Undersampling

Table 4.3.2 Performance metrics various ML classifiers before undersampling

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.913867	0.539789	0.10868	0.167976	0.836498
Random Forest	0.903157	0.345224	0.120172	0.170897	0.784965
Decision Tree	0.862771	0.213147	0.246423	0.229066	0.594944
XGBoost	0.912491	0.528838	0.098355	0.165897	0.832112
MLP Classifier	0.911615	0.47001	0.085479	0.144651	0.822399

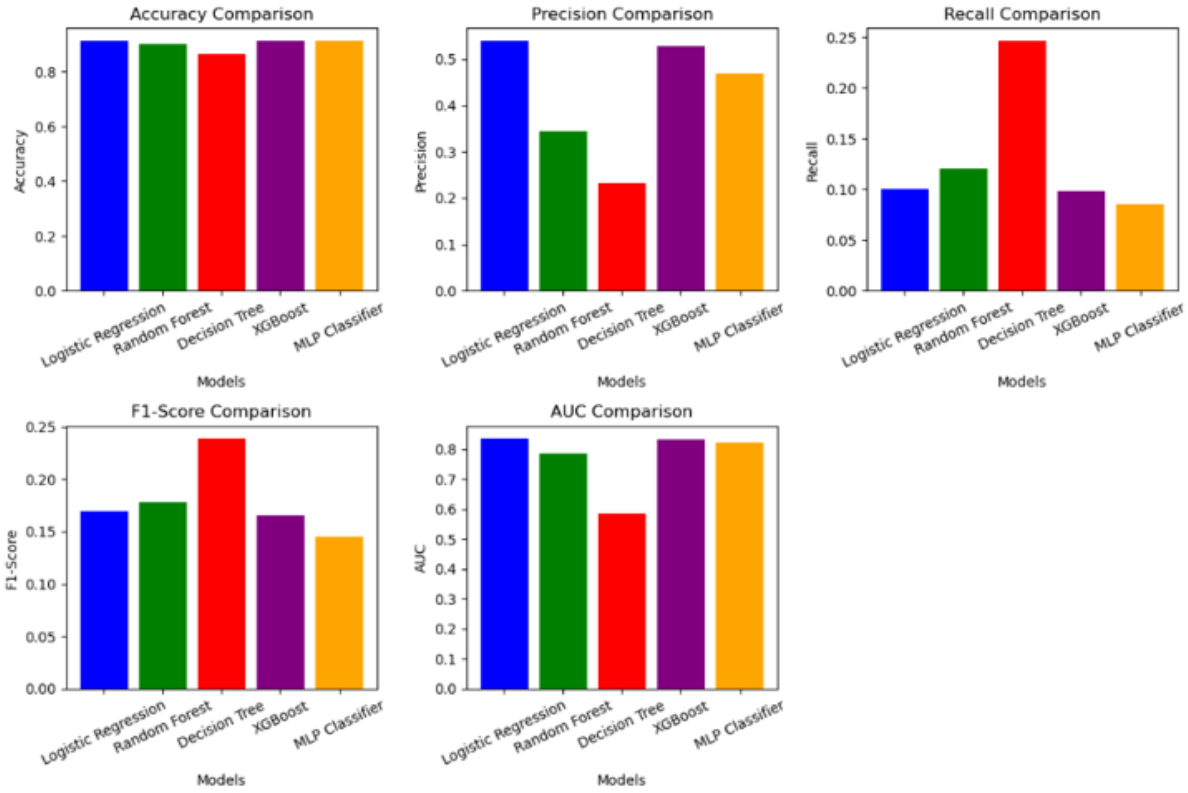

Fig. 4.3.4 Performance without Undersampling

Table 4.3.3 Performance metrics various ML classifiers after SMOTE

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.763065	0.752586	0.783804	0.767878	0.838968
Random Forest	0.923168	0.921977	0.924578	0.923275	0.971379
Decision Tree	0.888184	0.886603	0.890226	0.888411	0.888898
XGBoost	0.909429	0.936729	0.878172	0.906506	0.970989
MLP Classifier	0.815396	0.802489	0.836725	0.819249	0.900332

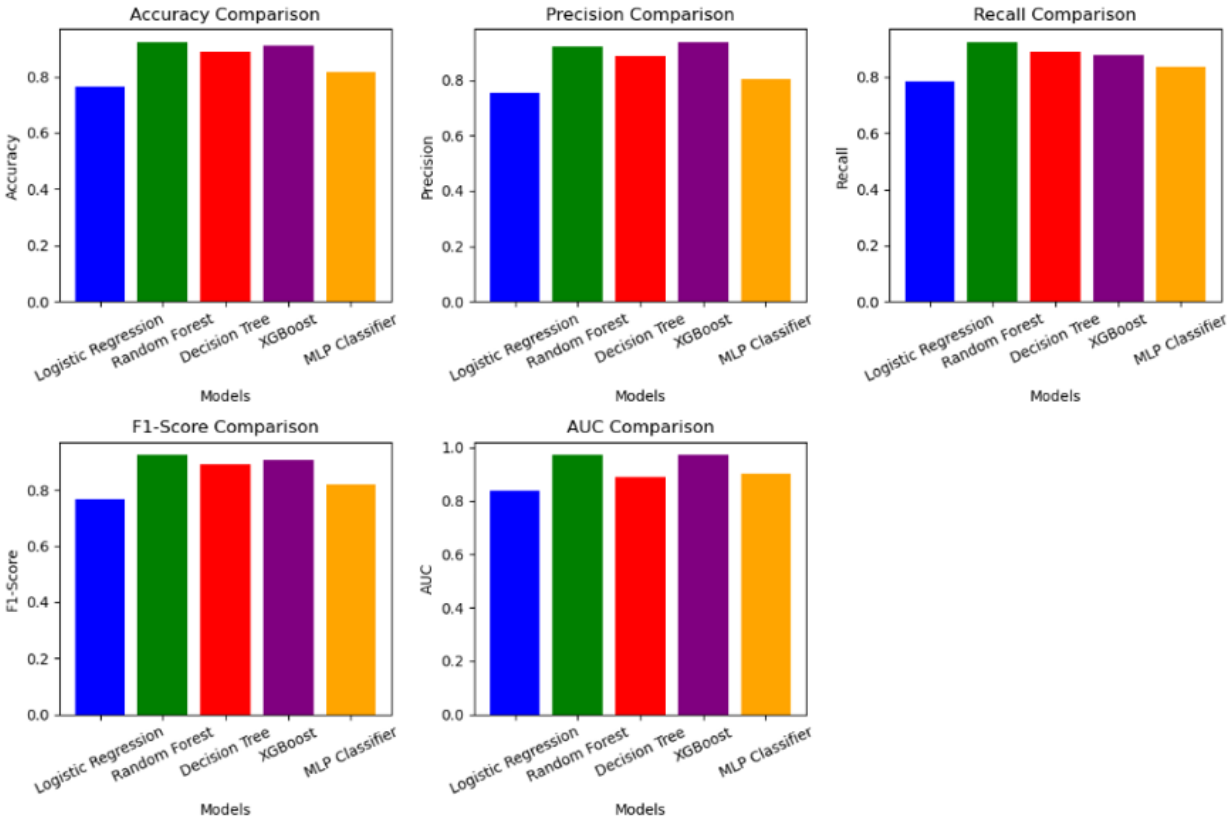

Fig 4.3.5 Performance after SMOTE

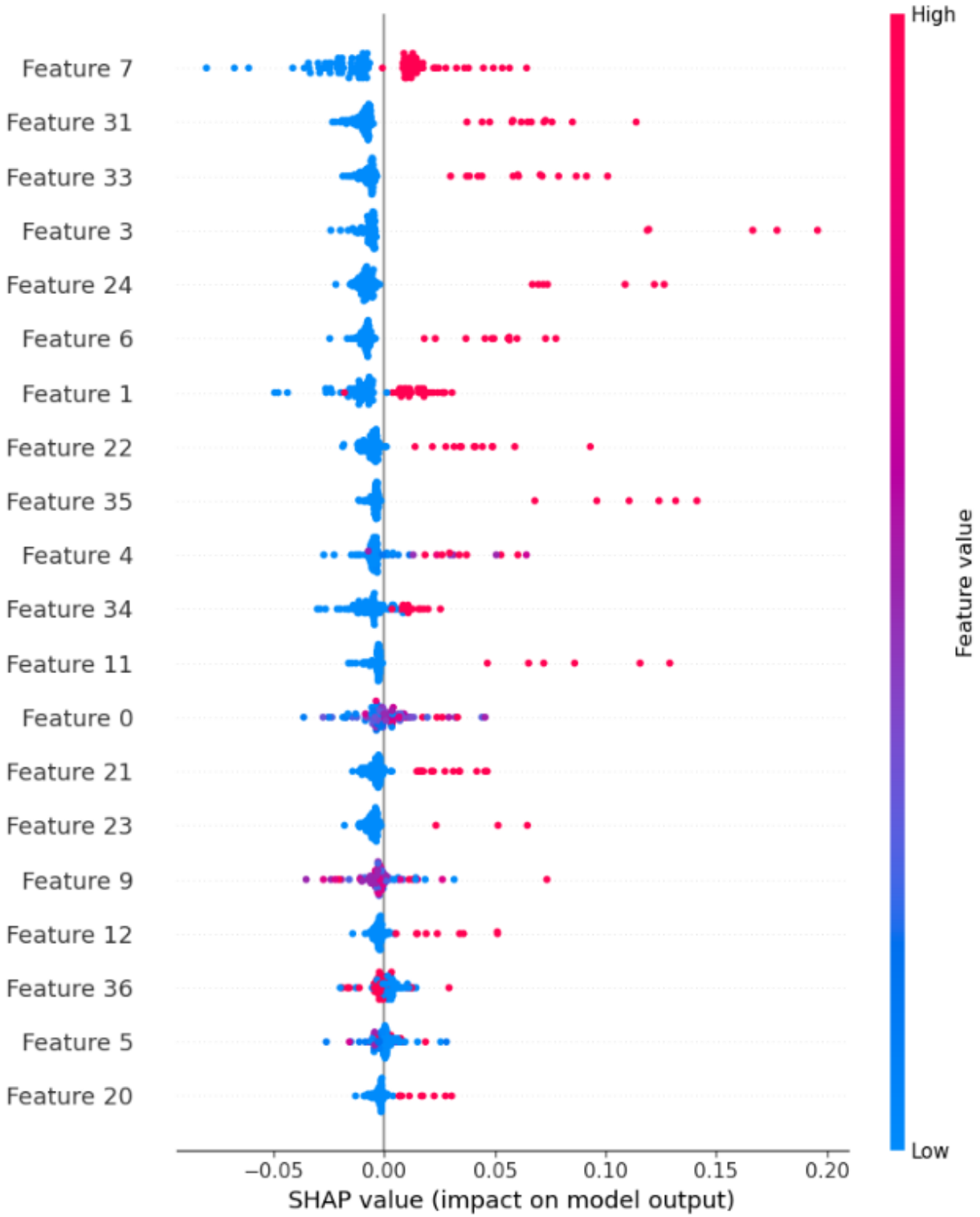

Fig. 4.3.6 SHAP Beeswarm – Feature Contributions

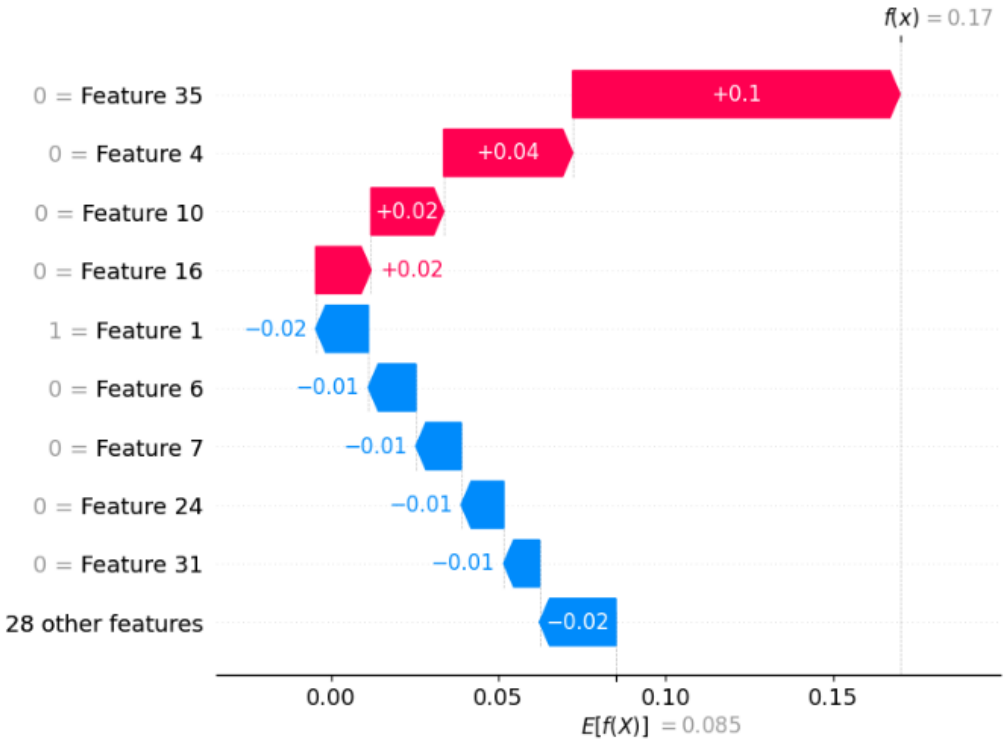

Fig. 4.3.7 SHAP Summary – Feature Impact

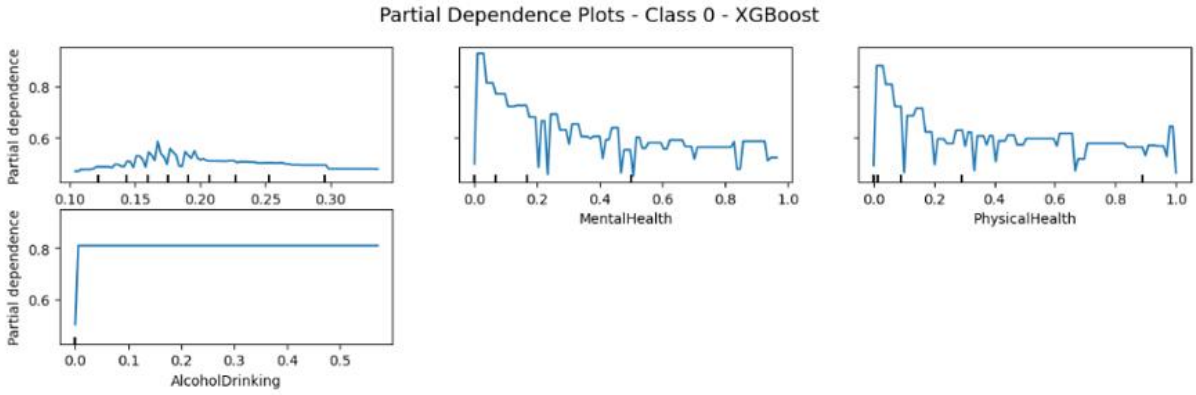

Fig. 4.3.8 PDP – class 0

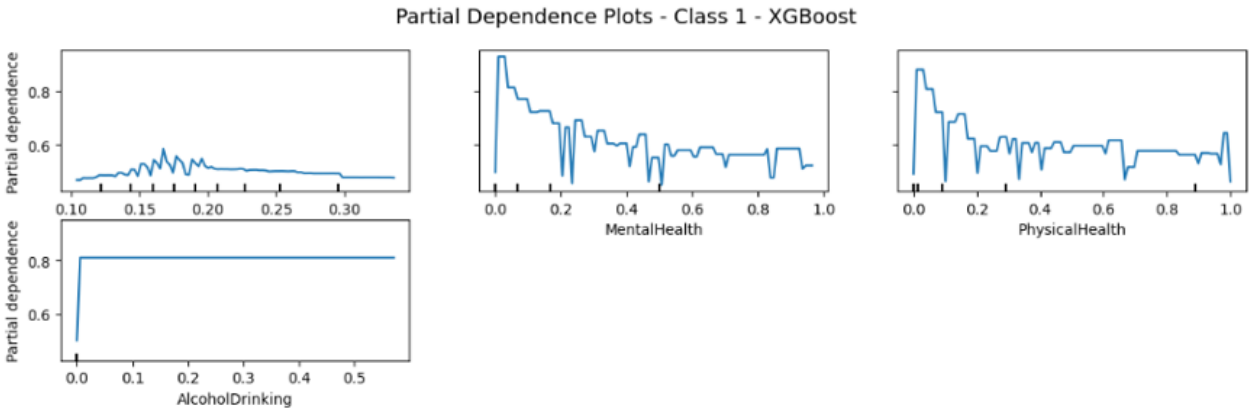

Fig. 4.3.9 PDP – class 1

4.4 Obesity dataset

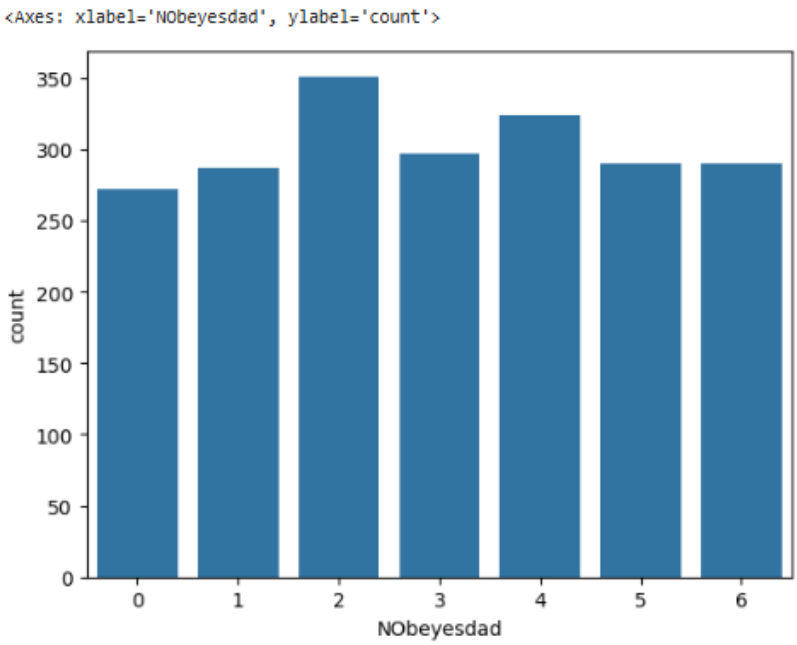

Fig. 4.4.1 Obesity class distribution

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.690835	0.69244	0.690835	0.676463	0.91541
Random Forest	0.955083	0.955618	0.955083	0.955266	0.997651
Decision Tree	0.938534	0.939843	0.938534	0.938573	0.964707
XGBoost	0.962175	0.962662	0.962175	0.96206	0.998881
MLP Classifier	0.921986	0.922434	0.921986	0.921807	0.995831

Table 4.4.1 Performance metrics various ML classifiers before SMOTE

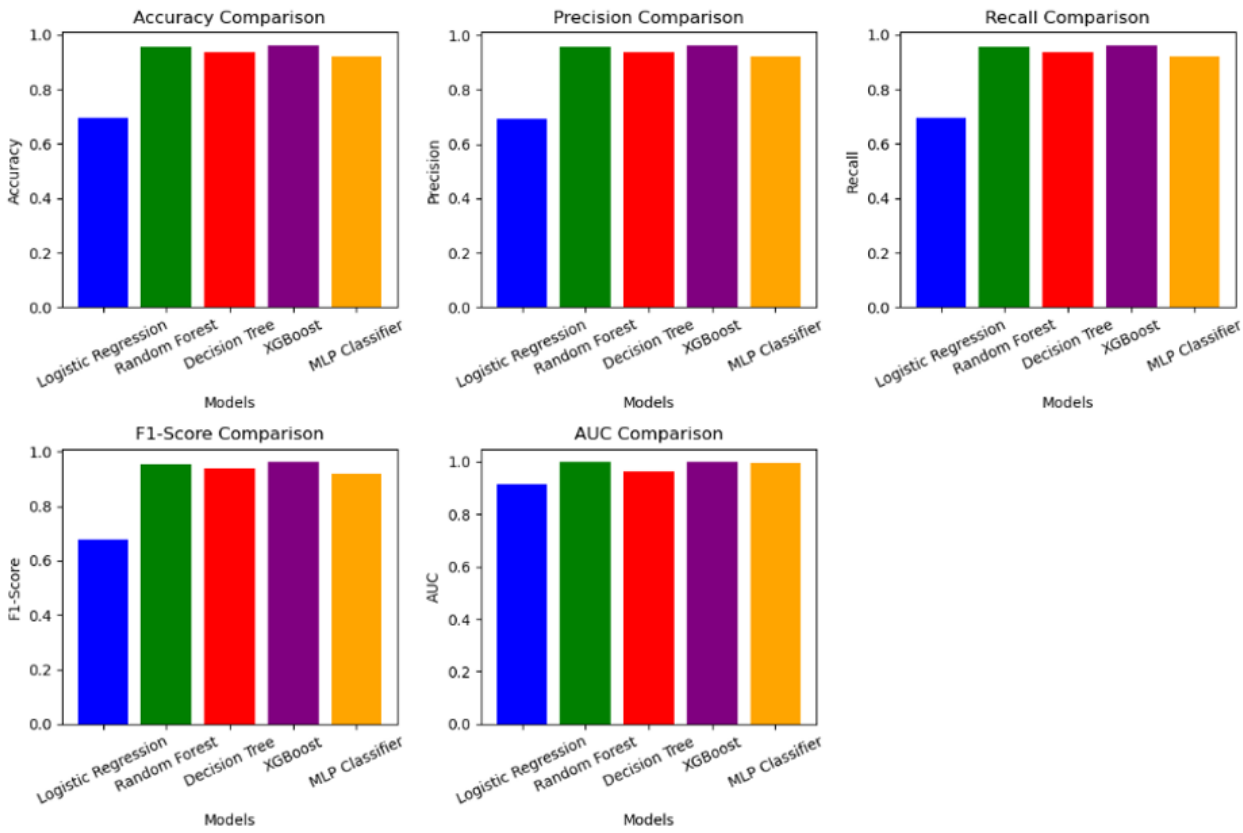

Fig. 4.4.2 Performance without applying SMOTE

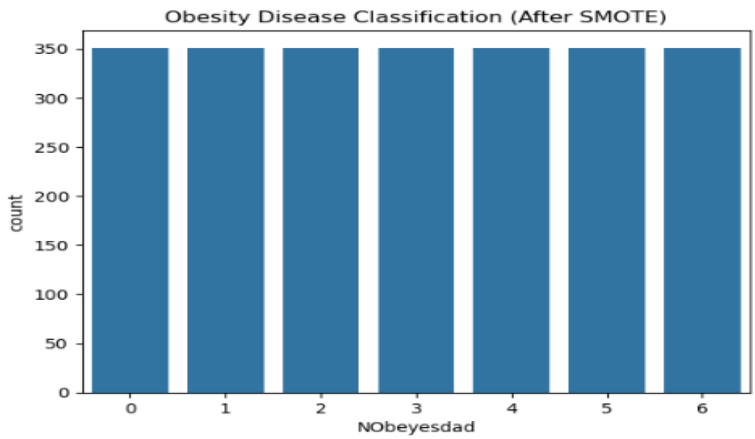

Fig. 4.4.3 Obesity class distribution after applying SMOTE

Table 4.4.2 Performance metrics various ML classifiers after SMOTE

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.705285	0.694095	0.705285	0.684533	0.91917
Random Forest	0.955285	0.956024	0.955285	0.95553	0.997533
Decision Tree	0.924797	0.925039	0.924797	0.924569	0.956148
XGBoost	0.96748	0.968002	0.96748	0.967467	0.998661
MLP Classifier	0.936992	0.937861	0.936992	0.936605	0.99745

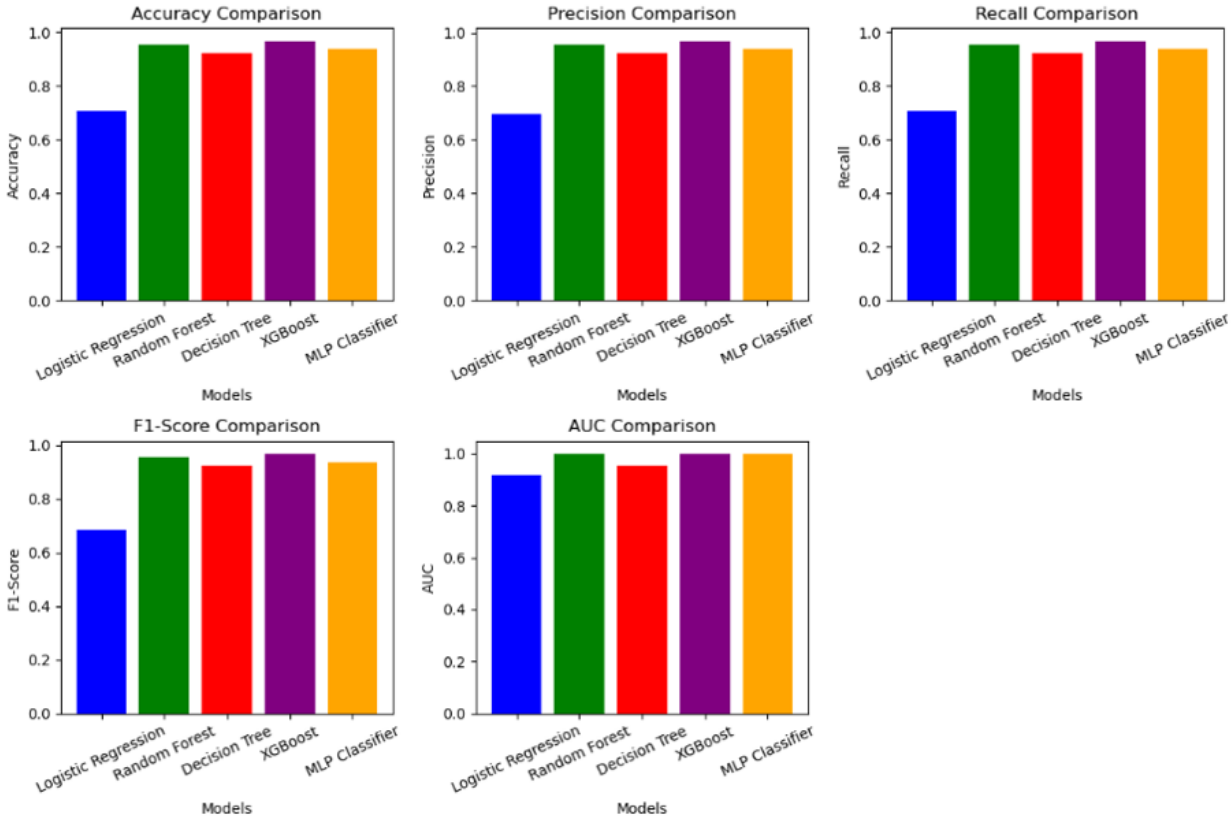

Fig. 4.4.4 Performance after SMOTE

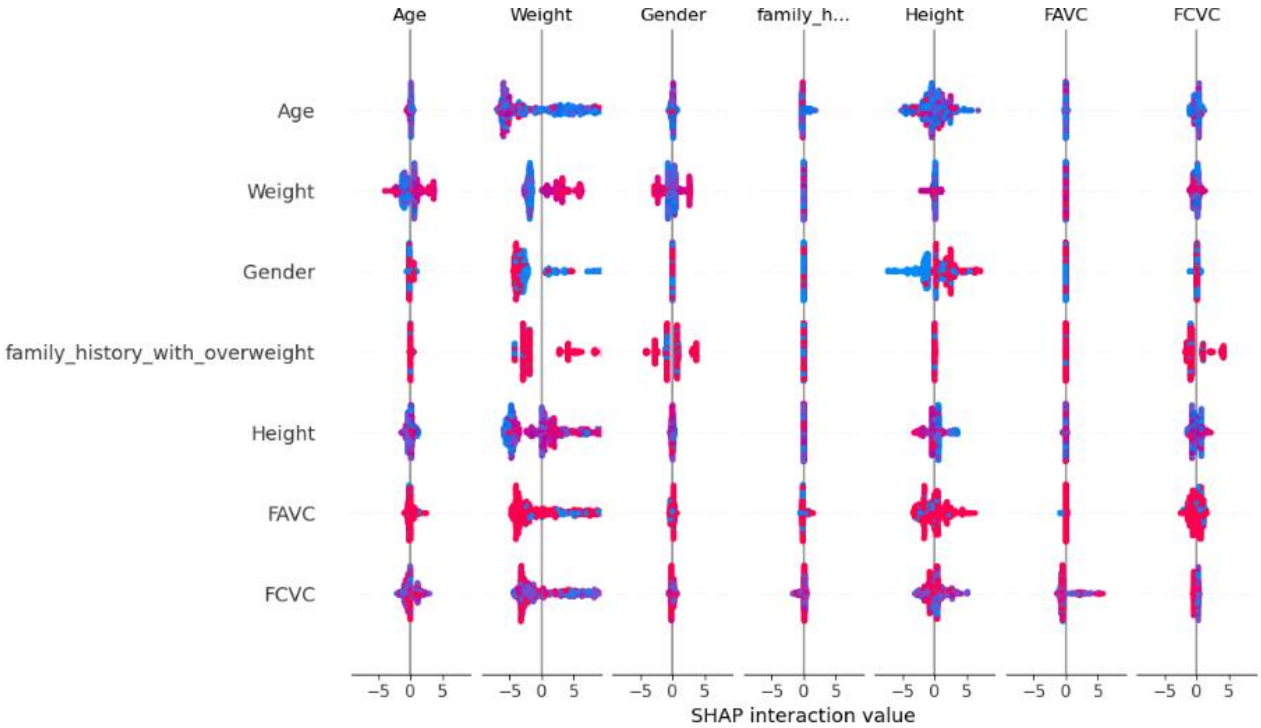

Fig. 4.4.5 SHAP Interaction – Feature Pair Effects

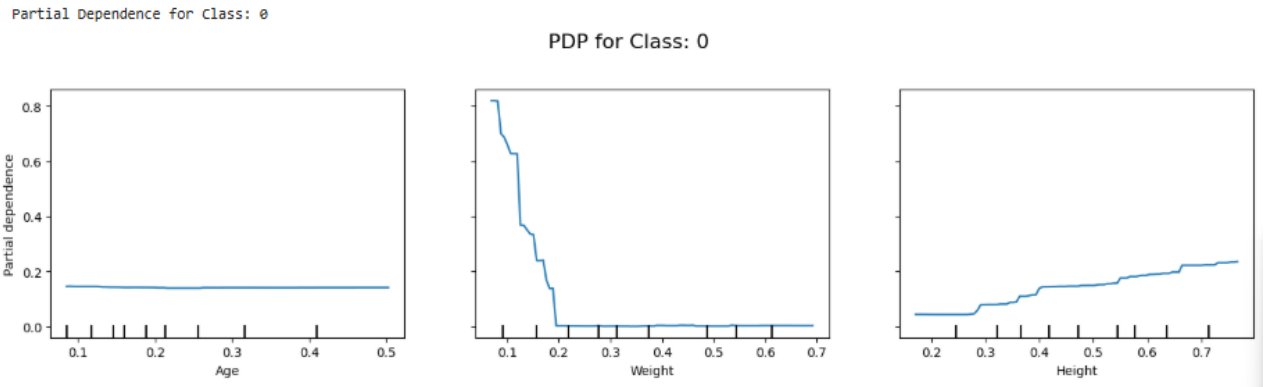

Fig. 4.4.6 PDP – class 0

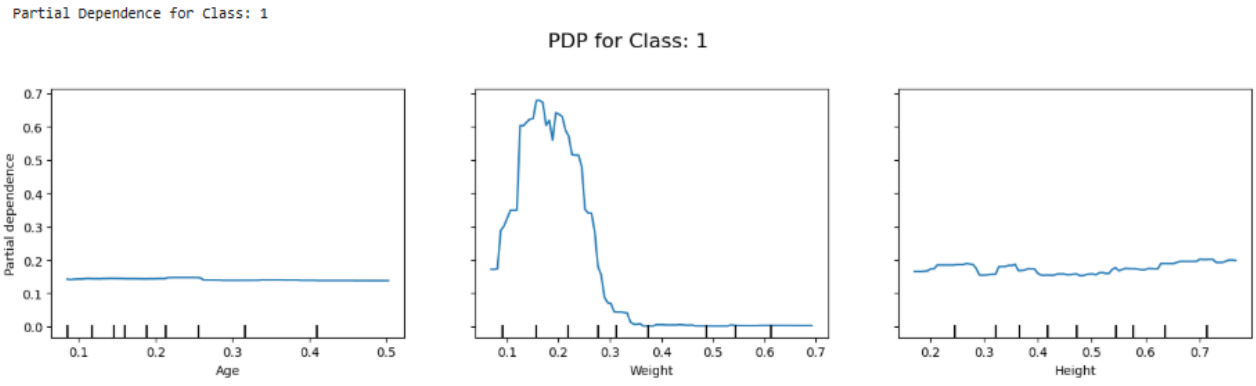

Fig. 4.4.7 PDP – class 1

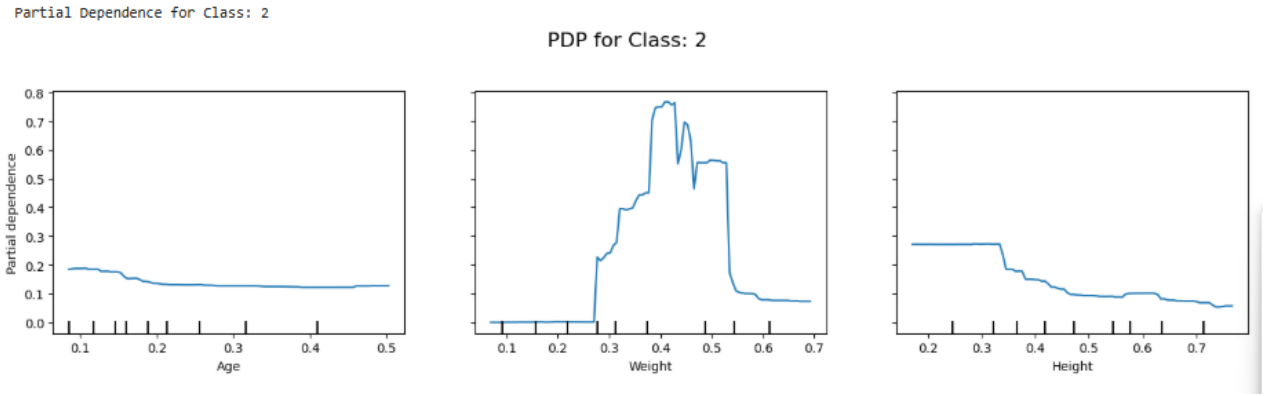

Fig. 4.4.8 PDP – class 2

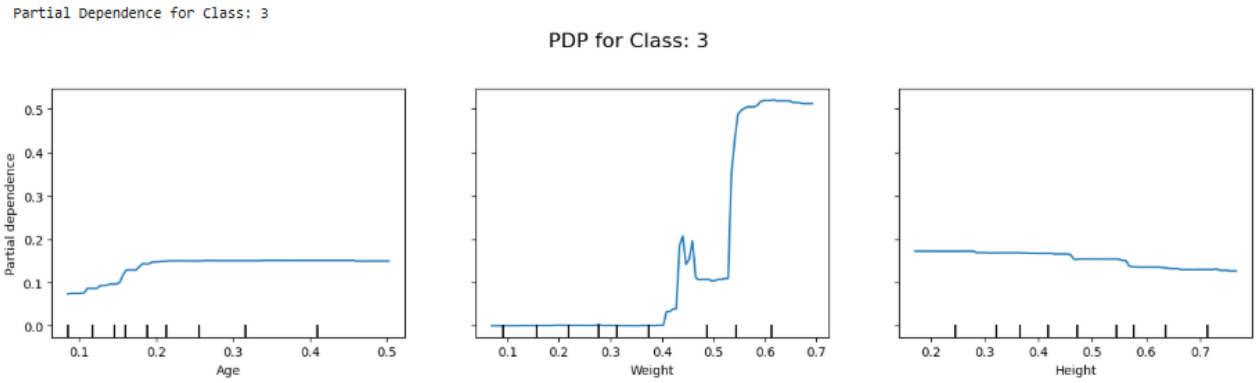

Fig. 4.4.9 PDP – class 3

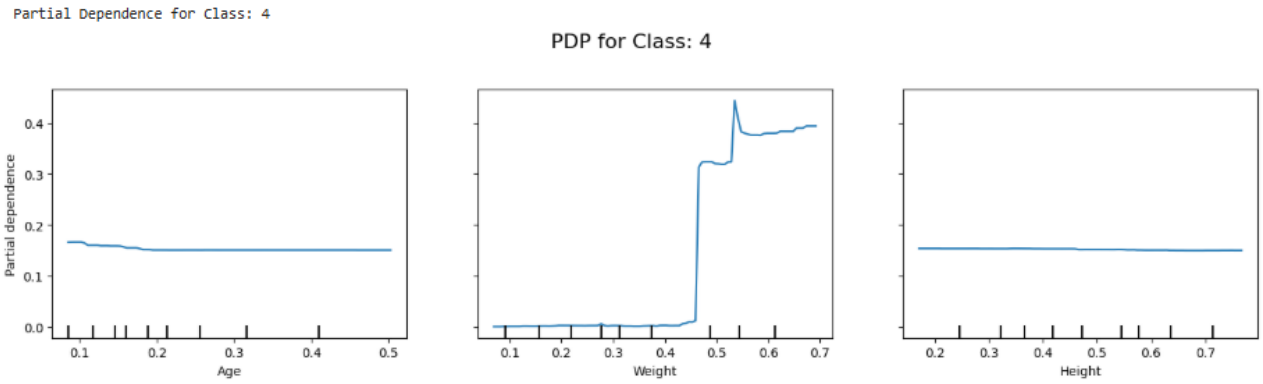

Fig. 4.4.10 PDP – class 4

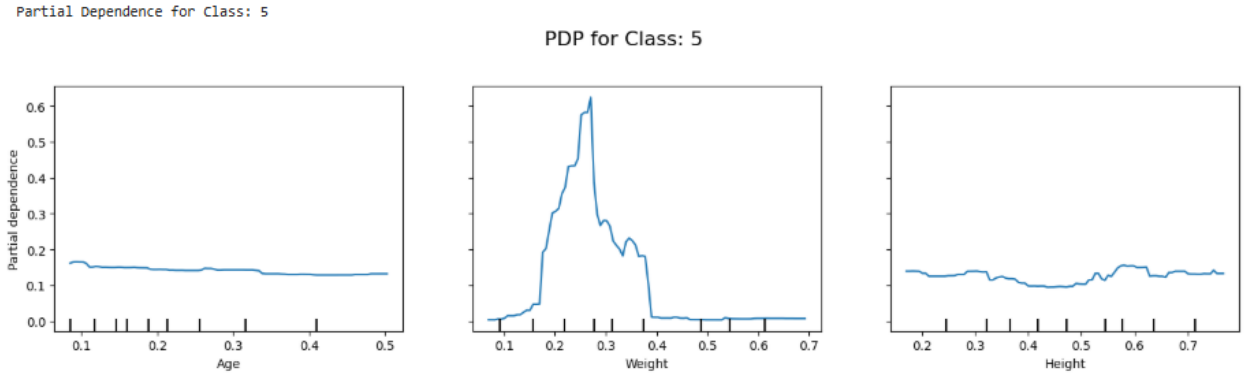

Fig. 4.4.11 PDP – class 5

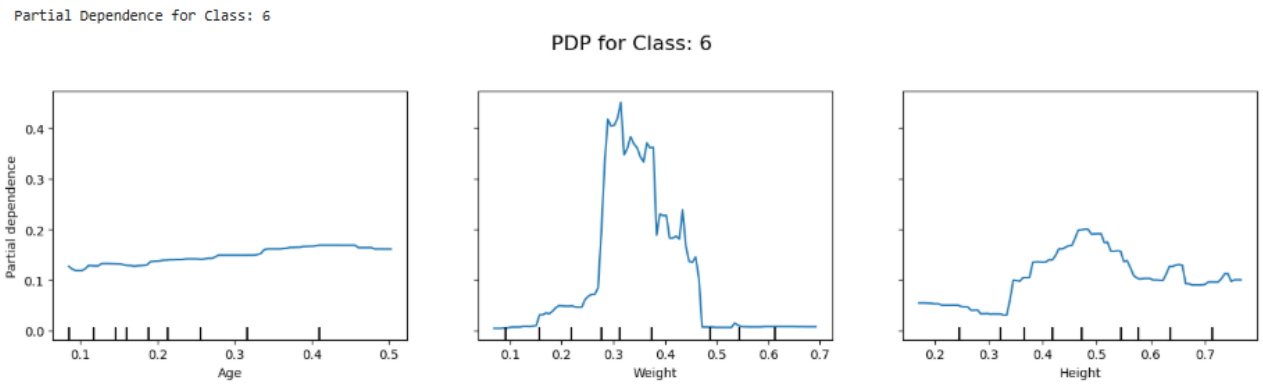

Fig. 4.4.12 PDP – class 6

4.5 Breast cancer dataset

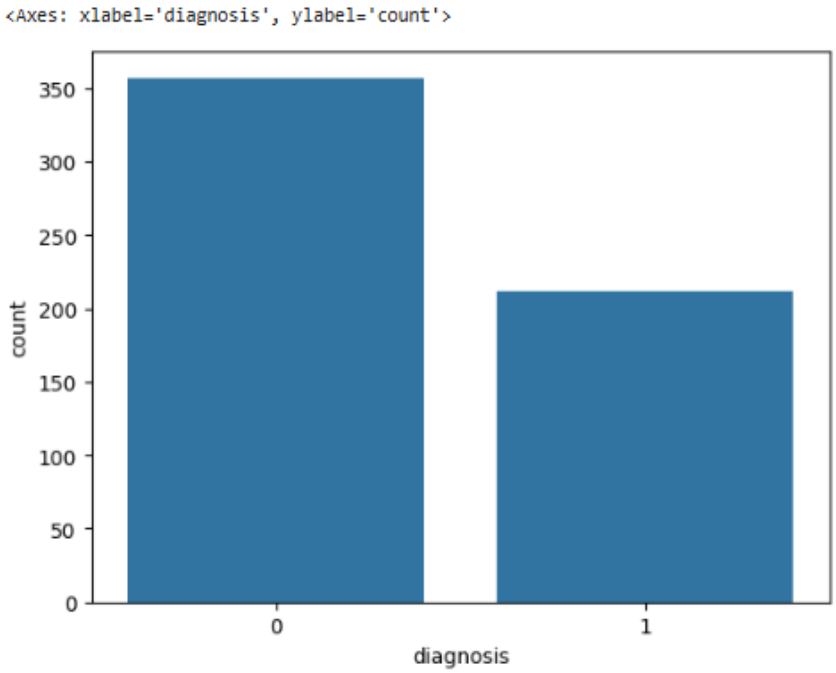

Fig: 4.5.1 class distribution

Table 4.5.1 Performance metrics various ML classifiers before SMOTE

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.986014	1.00000	0.962963	0.981132	0.998127
Random Forest	0.956035	0.962264	0.944444	0.953271	0.992405
Decision Tree	0.944856	0.910714	0.944444	0.927273	0.944132
XGBoost	0.958042	0.944444	0.944444	0.944444	0.992926
MLP Classifier	0.986014	0.981481	0.981481	0.981481	0.998127

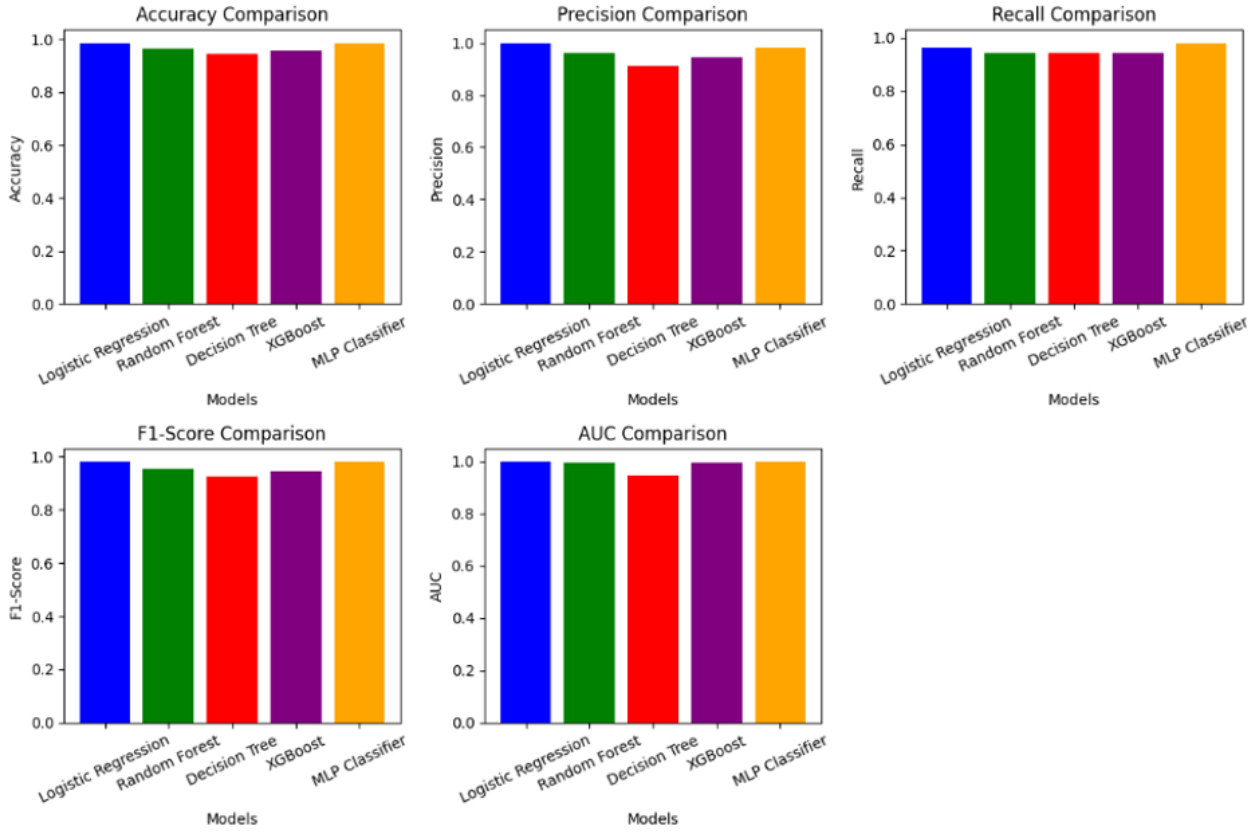

Fig. 4.5.2 Performance without applying SMOTE

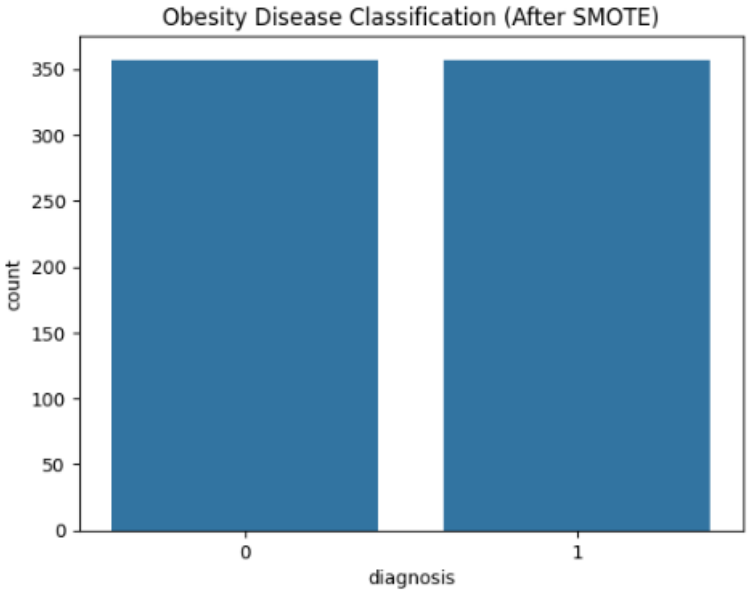

Fig. 4.5.3 class distribution after applying SMOTE

Table 4.5.2 Performance metrics various ML classifiers after SMOTE

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	0.96648	0.978022	0.956989	0.967391	0.995374
Random Forest	0.96648	0.967742	0.967742	0.967742	0.993311
Decision Tree	0.949721	0.9375	0.967742	0.952381	0.949897
XGBoost	0.972267	0.978261	0.967742	0.972973	0.997999
MLP Classifier	0.972267	0.978261	0.967742	0.972973	0.994124

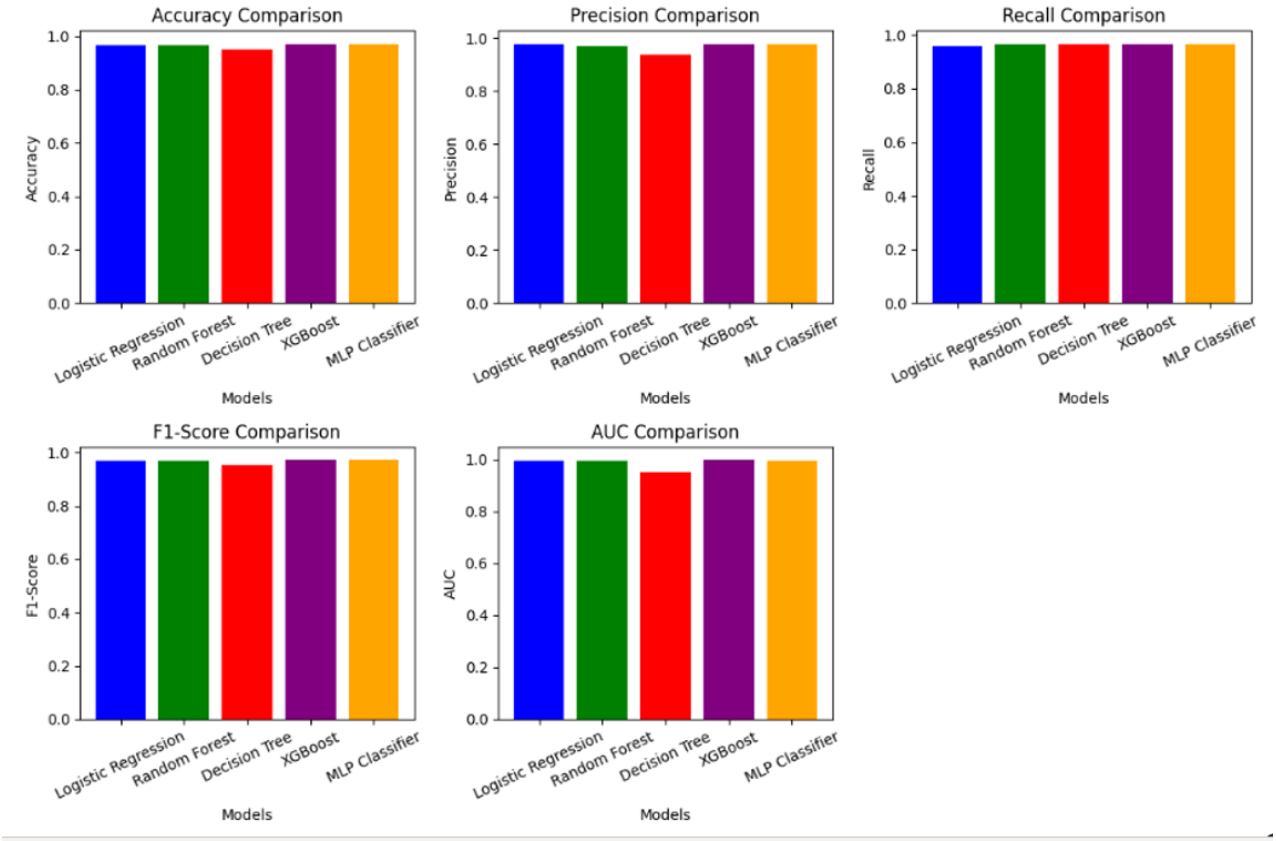

Fig. 4.5.4 Performance after SMOTE

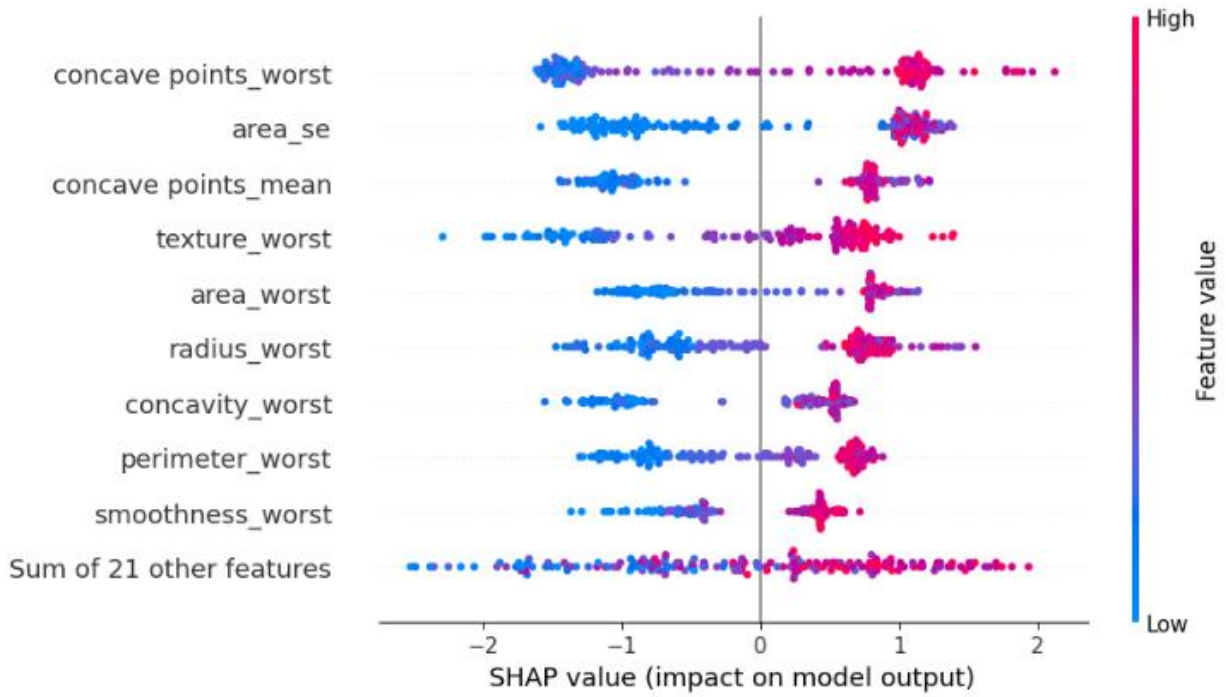

Fig. 4.5.5 SHAP Beeswarm – Feature Contributions

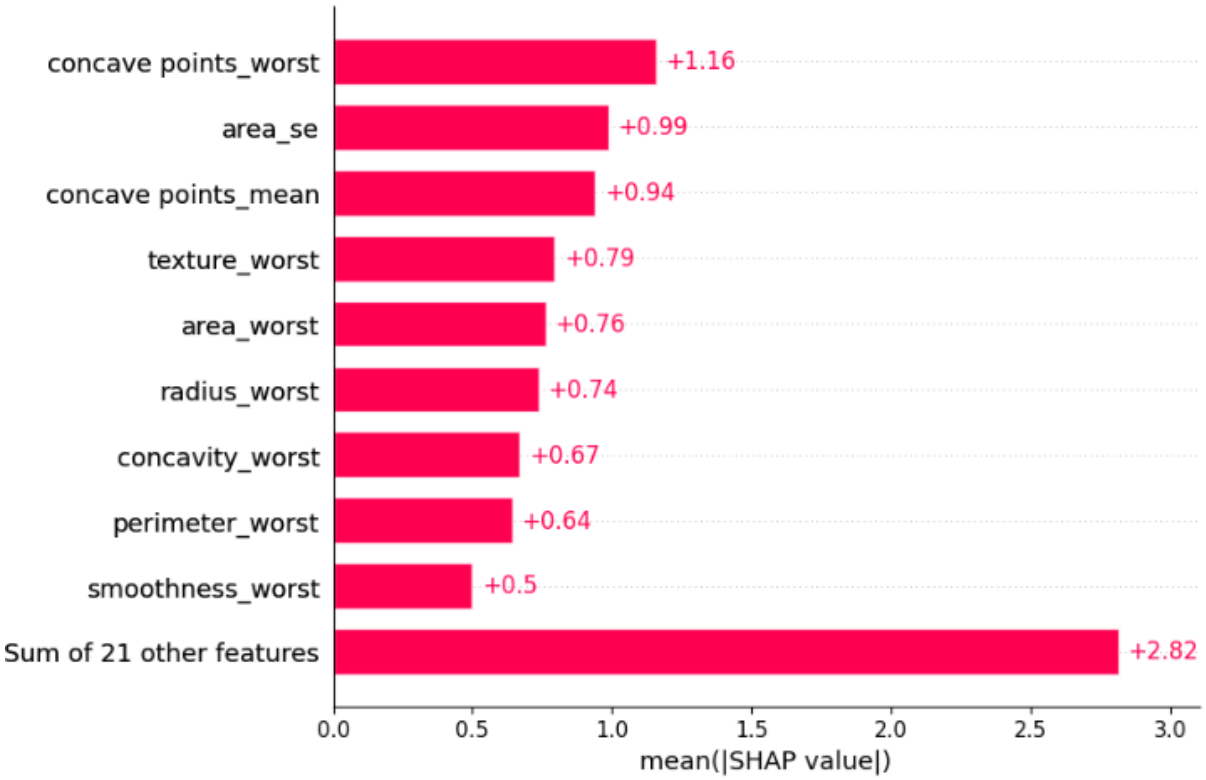

Fig. 4.5.6 SHAP Bar – Mean Feature Importance

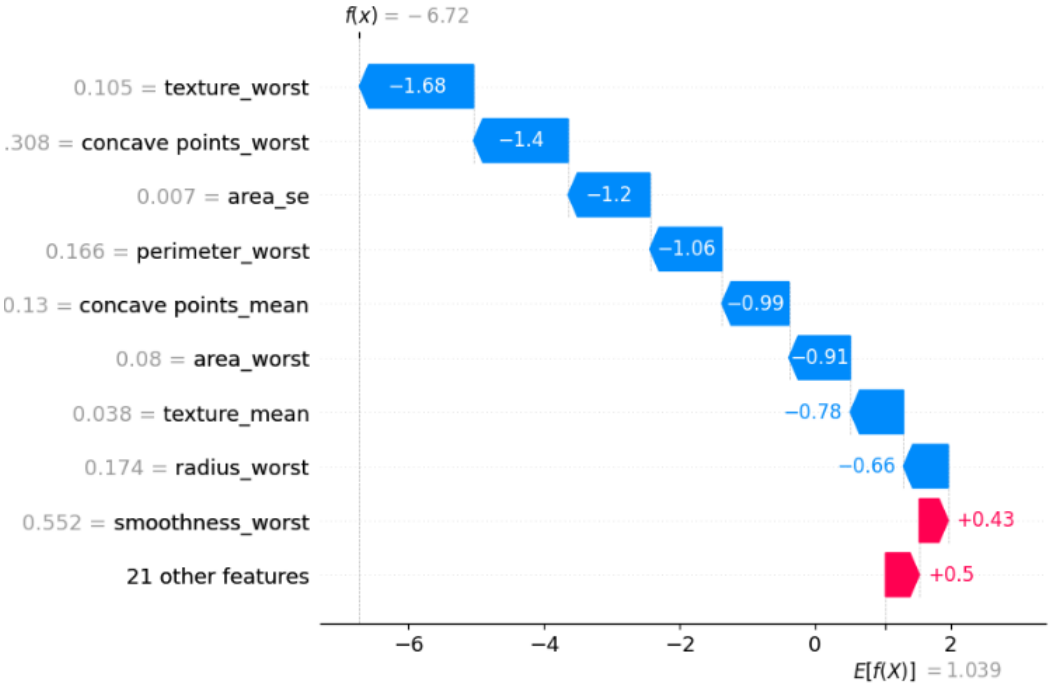

Fig. 4.5.7 SHAP Summary – Feature Impact

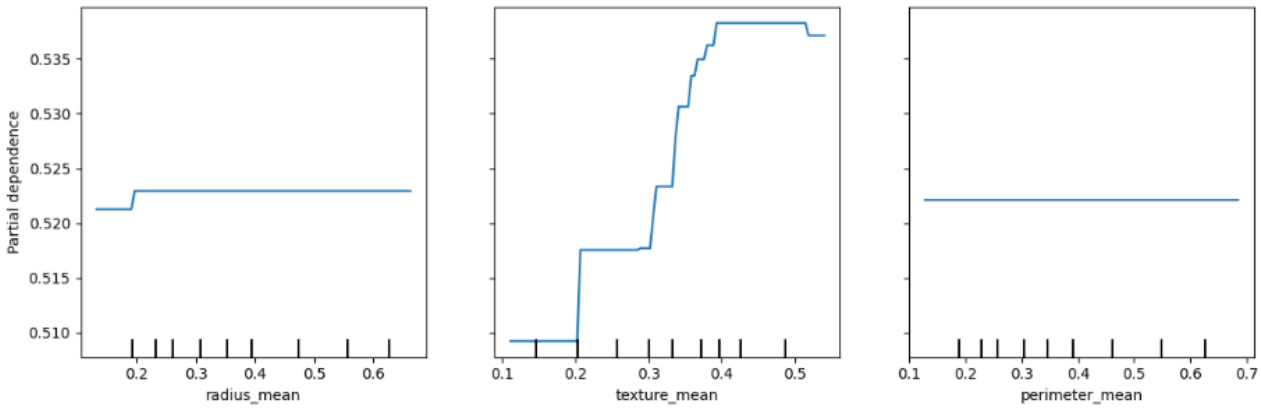

Fig. 4.5.8 PDP

CHAPTER 5

Conclusion and Future Plans

Conclusion:

This elucidates the potential of integrating machine learning and explainable AI (XAI) methodologies to augment the diagnosis of heart-related diseases, encompassing diabetes, obesity, breast cancer, and cardiovascular conditions. Through the application of synthetic minority over-sampling technique (SMOTE) and under-sampling techniques, the proposed approach ensures the robustness of model training. Furthermore, the utilization of ensemble methods such as Stacking and Bagging enhances predictive performance. Moreover, the incorporation of explainability techniques, including SHAP, LIME, and Partial Dependence Plot (PDP), provides pivotal insights into model decision-making, fostering transparency and trust among clinicians.

Future Plans:

Hyper parameters tuning: Improve model accuracy by applying different methods like GridSearch, Randomized Search, Bayesian optimization and select parameters that suits best to the model

Feature Engineering: Instead of using all the features present in dataset, apply some feature engineering techniques like pca to extract relevant or top features and try to improve the model accuracy

CHAPTER 6

REFERENCES

1. S. Pattanayak and T. Singh, ‘‘Cardiovascular disease classification based on machine learning algorithms using GridSearchCV, cross-validation and stacked ensemble methods,’’ in *Advances in Computing and Data Sciences*, M. Singh, V. Tyagi, P. K. Gupta, J. Flusser, T. Ören, Eds. Cham, Switzerland: Springer, 2022, pp. 219–230.
2. Singh, A., Mahapatra, H., Biswal, A.K., Mahapatra, M., Singh, D. and Samantaray, M., 2024. *Heart Disease Detection Using Machine Learning Models*. *Procedia Computer Science*, 235, pp.
3. B. J. Kenner et al., ‘‘Artificial intelligence and early detection of pancreatic cancer: 2020summativereview,’’Pancreas,vol.50,p. 251–279, Mar. 2021.
4. F. Javier Basterra-Gortari, M. Bes-Rastrollo, M. Seguí-Gómez, L. Forga, J. Alfredo Martínez, and M. Ángel Martínez-González, ‘‘Tendencias de la obesidad, diabetes mellitus, hipertensión e hipercolesterolemia en España (1997–2003),’’ Medicina Clínica, vol. 129, no. 11, pp. 405–408, Sep. 2007.
5. N. Haug, J. Sorger, T. Gisinger, M. Gyimesi, A. Kautzky-Willer, S. Thurner, and P. Klimek, ‘‘Decompression of multimorbidity along the disease trajectories of diabetes mellitus patients,’’ Frontiers Physiol., vol. 11, Jan. 2021, Art. no. 612604.
6. E. Martin-Rodriguez, F. Guillen-Grima, A. Martí, and A. Brugos Larumbe, ‘‘Comorbidity associated with obesity in a large population: The APNA study,’’ Obesity Res. Clin. Pract., vol. 9, no. 5, pp. 435–447, Sep. 2015.
7. K. Zatońska, P. Psikus, A. Basiak-Rasała, Z. Stepnicka, D. Gaweł-Däbrowska, M. Wołyniec, J. Gibka, A. Szuba, and K. Połtyn-Zaradna, ‘‘Obesity and chosen non-communicable diseases in PURE Poland cohort study,’’ Int. J. Environ. Res. Public Health, vol. 18, no. 5, p. 2701, Mar. 2021.
8. D. P. Guh, W. Zhang, N. Bansback, Z. Amarsi, C. L. Birmingham, and A. H. Anis, ‘‘The incidence of co-morbidities related to obesity and overweight: A systematic review and meta-analysis,’’ BMC Public Health, vol. 9, no. 1, p. 88, Dec. 2009.

CHAPTER 7

Appendix and Base Paper

Base Paper: EXplainable AI for Decision Support to Obesity Comorbidities Diagnosis

URL: https://ieeexplore.ieee.org/document/10265033