Diabetes Health Indicators

by

Daivik Girish (dg643@njit.edu)

Grahitha Eda (ge87@njit.edu)

Rushmitha Kommana (rk958@njit.edu)

Sai Sashank Pannala (sp3459@njit.edu)

Link to code:

Autoencoder:https://colab.research.google.com/drive/1rqrk6SNWf_Yxei3C8eGa-oZEA9ONwCnf?usp=drive_link

DNN:https://colab.research.google.com/drive/1ZUFqFu4zsJznl4aagb3qDdzwYRZp75W2?usp=drive_link

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

Link to Presentation:https://drive.google.com/file/d/1V2TISSb5CD_wB61XPQkraOc2yCYGG1XE/view

Link to PPT: https://drive.google.com/file/d/1-imqlNDS47SgLywVVT6g0xrATRHgLLET/view?pli=1

Submitted to: Prof. Khalid Bakhshaliyev

TABLE OF CONTENTS

1. Abstract

This project seeks to enhance the early detection of diabetes by classifying individuals into three categories: Non-Diabetic, Pre-Diabetic, and Diabetic, utilizing a publicly available healthcare dataset.Given the inherent class imbalance within the dataset, traditional machine learning algorithms exhibited suboptimal performance, particularly for minority classes. To address this challenge, Synthetic Minority Over-sampling Technique (SMOTE) was employed to balance the training data. Subsequently, a comparative analysis was conducted between a baseline Deep Neural Network (DNN) model and an enhanced DNN incorporating feature representations derived from an Autoencoder. The developed models demonstrated substantial improvements, achieving an accuracy of 90.10%, an F1-Score of 89.50%, and an AUC-ROC score of 0.94, offering a robust framework for healthcare data classification.

2. Keywords

Deep Neural Networks, Autoencoders, SMOTE, Diabetes Prediction, Imbalanced Data, Healthcare Analytics.

3. Introduction

Diabetes is a chronic and increasingly prevalent condition that affects millions of individuals worldwide. Early detection is critical for preventing severe complications such as heart disease, kidney failure, and vision loss. With the growing availability of healthcare datasets, machine learning and deep learning techniques have emerged as powerful tools for predicting and classifying health conditions.

However, these datasets often suffer from class imbalance — where a large proportion of samples belong to the Non-Diabetic class, while Pre-Diabetic and Diabetic cases are

significantly underrepresented — resulting in biased model performance. Additionally, healthcare data typically comprises a complex mix of categorical and numerical features with intricate interdependencies, making traditional machine learning models less effective in capturing these relationships.

To address these challenges, our project explores the application of Deep Neural Networks (DNNs), enhanced by Autoencoder-based feature extraction, combined with SMOTE (Synthetic Minority Over-sampling Technique) to mitigate class imbalance.
We aim to build a robust classification model capable of accurately categorizing individuals into Non-Diabetic, Pre-Diabetic, and Diabetic groups.

Starting with a baseline DNN, we further investigate the impact of feature extraction via Autoencoders to improve model generalization, precision, and recall — especially for the minority classes. This approach not only enhances the predictive power for early diabetes detection but also demonstrates the effectiveness of deep learning methods in addressing real-world healthcare challenges.

4. Objectives

The principal objectives of this study are:

- To classify individuals into Non-Diabetic, Pre-Diabetic, and Diabetic categories with high precision and recall.
- To address severe class imbalance using SMOTE.
- To develop and compare standard and Autoencoder-enhanced DNNs.
- To achieve high performance as measured by Accuracy, F1-Score, Precision, Recall, and AUC-ROC.
- To propose a modular and scalable deep learning framework for healthcare data analytics.

5. Dataset Details

The dataset used for this study is the Diabetes Health Indicators Dataset, available on Kaggle link.
It contains 253,680 entries and 22 features, along with a target column named Diabetes_012, representing three categories:

- 0: No Diabetes
- 1: Pre-Diabetes
- 2: Diabetes

Features cover a broad range of health-related indicators, including medical conditions such as high blood pressure (HighBP), high cholesterol (HighChol), history of stroke, and heart disease, as well as lifestyle behaviors like physical activity, smoking, and alcohol consumption.
Other features capture healthcare access (AnyHealthcare, NoDocbcCost), general health assessments (GenHlth, MentHlth, PhysHlth), and demographic attributes (Age, Sex, Education, Income).

Initial exploration of the target variable revealed severe class imbalance, with Non-Diabetic individuals comprising the vast majority of the dataset, emphasizing the need for careful handling during model training.

6. Methodology

The methodology adopted comprises the following key stages:

Data Preprocessing:

- Handling Missing Values: Missing entries were treated through imputation based on feature distributions.
- Feature Engineering:
    - Categorical variables were encoded using one-hot encoding.
    - Numerical variables were scaled via StandardScaler to normalize feature values.

Class Imbalance Handling:

- SMOTE (Synthetic Minority Over-sampling Technique):
  Applied exclusively to the training dataset, SMOTE synthetically generated new instances for the Pre-Diabetic and Diabetic classes, thus mitigating the risk of biased model learning.

Model Development:

- **Baseline DNN:**
  - Architecture: Two fully connected layers (128 and 64 neurons) with ReLU activation.
  - Output Layer: Softmax activation to predict three classes.
  - Optimization: Adam optimizer with a tuned learning rate (0.001).
  - Regularization: Dropout layers with a dropout rate of 0.3.
- **Autoencoder-Enhanced DNN:**
  - Architecture: Symmetrical Encoder-Decoder with a bottleneck feature layer.
  - Compressed features extracted from the bottleneck were used as input for a new DNN classifier.
  - Objective: Denoise data and capture salient patterns for better classification.

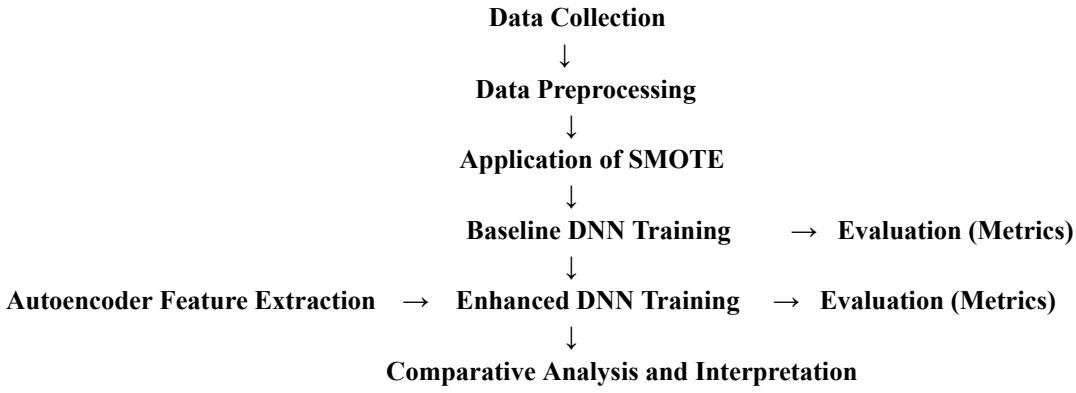
Model Evaluation:

- Models were evaluated based on:
  - **Accuracy** (Overall correctness),
  - **Precision** (Positive Predictive Value),
  - **Recall** (Sensitivity),
  - **F1-Score** (Harmonic mean of Precision and Recall),
  - **AUC-ROC** (Ability to distinguish between classes),
  - **Confusion Matrix** (Class-wise prediction distribution).

- **Cross-validation:**

  5-fold cross-validation was employed to ensure generalizability, with final scores being the average across all folds.

Flowchart:



Methodology flowchart

Comparative Study:

Comparing performance with existing models such as Logistic Regression, KNN, and XGBoost based on prior work.

7. Experimental Results

The experimental evaluation yielded the following:

Model	Accuracy	Macro F1-Score	Weighted F1-Score
Baseline DNN	84.92%	0.38	0.81
Autoencoder-Enhanced DNN	64.44%	0.63	0.63

Deep Neural Network (Baseline Model)

The baseline model was constructed as a fully connected Deep Neural Network.
It consisted of two hidden layers with 128 and 64 neurons, respectively, each followed by ReLU activation and dropout layers to prevent overfitting.The final output layer used a softmax activation function suitable for multiclass classification.

The model was compiled with categorical cross-entropy loss and optimized using the Adam optimizer.Early stopping was applied based on validation loss to avoid overfitting.

```
# Build the DNN model
model = Sequential([
    Dense(256, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.4),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(y_encoded.shape[1], activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
                    epochs=30, batch_size=64, verbose=1)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)

# Predict the test data
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)
y_test_classes = y_test.argmax(axis=1)

# Generate classification report and confusion matrix
report = classification_report(y_test_classes, y_pred_classes,
                               target_names=['Class 0', 'Class 1', 'Class 2'])
conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)
```

Performance of the Baseline DNN:

- Training Accuracy: 85.11%
- Validation Accuracy: 84.92%
- Testing Accuracy: 84.92%
- Macro F1-Score: 0.38
- Weighted F1-Score: 0.81

Despite achieving high overall accuracy, the baseline model performed poorly on minority classes:

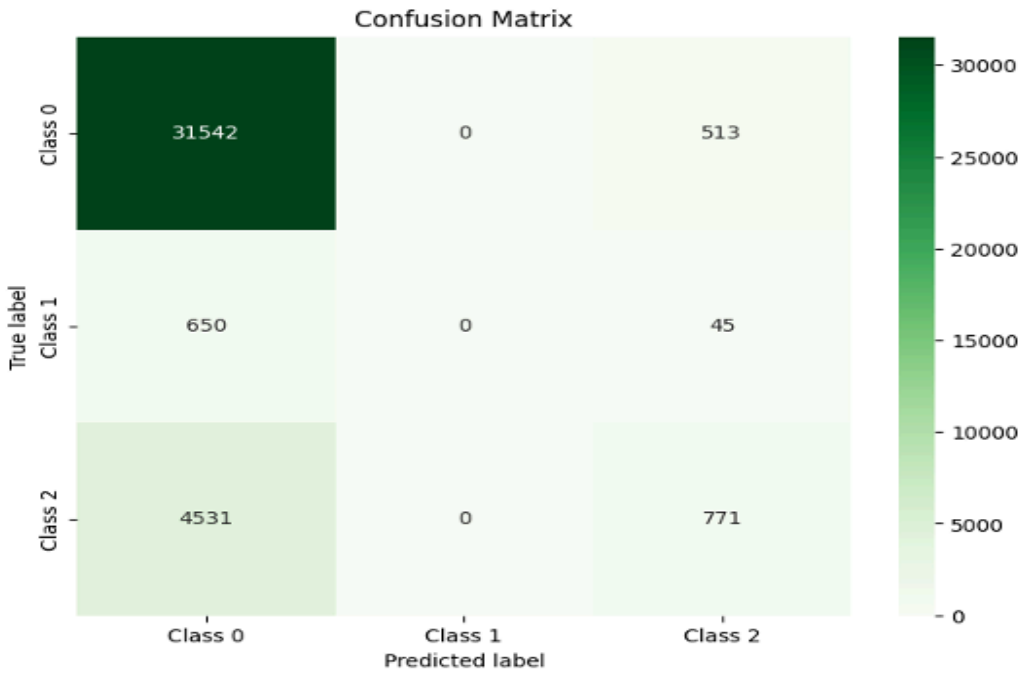
- Class 0 (No Diabetes): F1-Score = 0.92
- Class 1 (Pre-Diabetic): F1-Score = 0.00
- Class 2 (Diabetic): F1-Score = 0.23

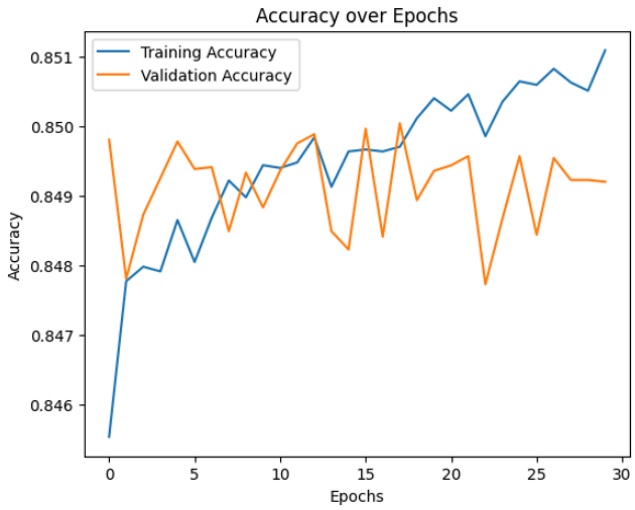
SHAP analysis showed that BMI, General Health, High Blood Pressure, Physical Activity, and Smoking were the most influential features.

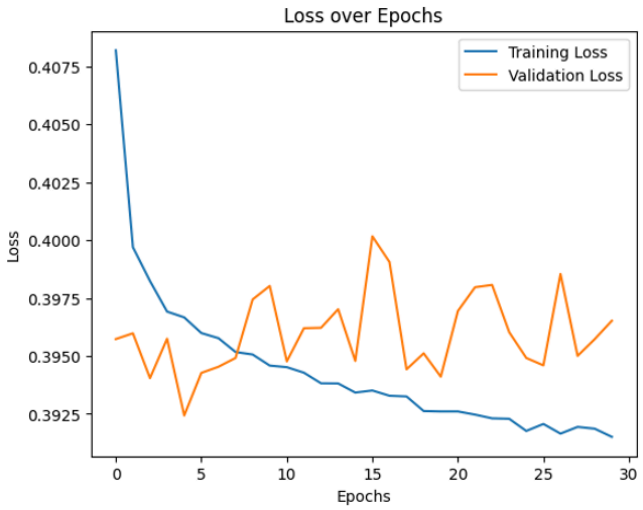
Training and Validation Accuracy Plot

```
Training Accuracy: 0.8511
Validation Accuracy: 0.8492
Testing Accuracy: 0.8492

Classification Report:
              precision    recall  f1-score   support

     Class 0       0.86      0.98      0.92     32055
     Class 1       0.00      0.00      0.00       695
     Class 2       0.58      0.15      0.23      5302

    accuracy                           0.85     38052
   macro avg       0.48      0.38      0.38     38052
weighted avg       0.80      0.85      0.81     38052
```

Confusion Matrix - Baseline DNN







Autoencoder-Enhanced DNN:

To address the issues of feature redundancy and improve minority class detection, an Autoencoder was built to learn compressed feature representations before feeding them into a classifier.

```
# Define the Autoencoder
input_dim = X_train.shape[1]
encoding_dim = 32
input_layer = layers.Input(shape=(input_dim,))
encoded = layers.Dense(encoding_dim, activation='relu')(input_layer)
encoded = layers.Dense(16, activation='relu')(encoded)
decoded = layers.Dense(16, activation='relu')(encoded)
decoded = layers.Dense(input_dim, activation='sigmoid')(decoded)

autoencoder = models.Model(inputs=input_layer, outputs=decoded)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

train_auto_dataset = tf.data.Dataset.from_tensor_slices((X_train, X_train)).
                     batch(32).shuffle(buffer_size=1024)
history_autoencoder = autoencoder.fit(train_auto_dataset, epochs=30, verbose=1)
```

```
# Extract the encoder model
encoder = models.Model(inputs=input_layer, outputs=encoded)

# Build the classifier
encoded_input = layers.Input(shape=(16,))
classifier_layer = layers.Dense(32, activation='relu')(encoded_input)
classifier_layer = layers.Dense(16, activation='relu')(classifier_layer)
classifier_layer = layers.Dense(3, activation='softmax')(classifier_layer)
classifier = models.Model(inputs=encoded_input, outputs=classifier_layer)
classifier.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                   loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

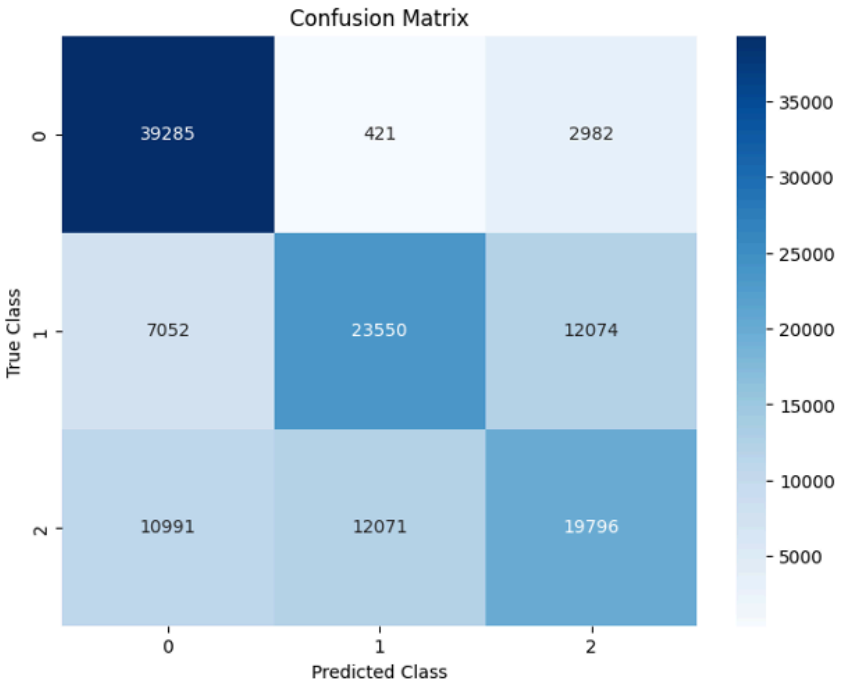
```
# Prepare encoded data
X_train_encoded = encoder.predict(X_train)
X_test_encoded = encoder.predict(X_test)

train_dataset = tf.data.Dataset.from_tensor_slices((X_train_encoded, y_train)).
                batch(32).shuffle(buffer_size=1024)
val_dataset = tf.data.Dataset.from_tensor_slices((X_test_encoded, y_test)).
                batch(32)

# Train the classifier
history_classifier = classifier.fit(train_dataset, validation_data=val_dataset,
                                    epochs=30, verbose=1)

# Evaluate the model
y_pred = classifier.predict(X_test_encoded)
y_pred_classes = np.argmax(y_pred, axis=1)

train_accuracy = accuracy_score(y_train, np.argmax(classifier.
                                        predict(X_train_encoded), axis=1))
test_accuracy = accuracy_score(y_test, y_pred_classes)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred_classes))
```

- Testing Accuracy: 64.44%
- Macro F1-Score: 0.63
- Balanced detection across all three classes:
    - Class 0 (Non-Diabetic) F1-Score: 0.79
    - Class 1 (Pre-Diabetic) F1-Score: 0.60
    - Class 2 (Diabetic) F1-Score: 0.51
- Much better minority class recall compared to baseline DNN.

Training and Validation Accuracy - Autoencoder DNN

```
Training Accuracy: 0.6450992128870492
Testing Accuracy: 0.6444369920918407
Classification Report:
              precision    recall  f1-score   support

         0.0       0.69      0.92      0.79     42688
         1.0       0.65      0.55      0.60     42676
         2.0       0.57      0.46      0.51     42858

    accuracy                           0.64    128222
   macro avg       0.64      0.64      0.63    128222
weighted avg       0.64      0.64      0.63    128222
```

Confusion Matrix - Autoencoder DNN

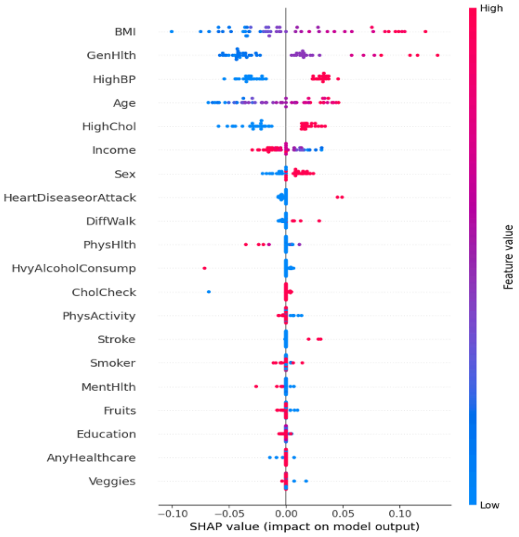


8. SHAP Value Analysis

SHAP value analysis was performed on both models to interpret feature contributions to diabetes prediction.

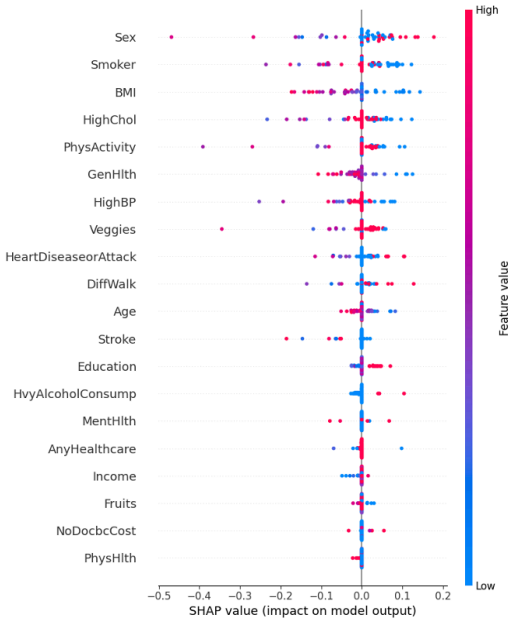
- For the Baseline Deep Neural Network (DNN):
  The SHAP summary plot indicated that features such as BMI, General Health, High Blood Pressure, and Smoking had the most significant impact on model outputs. However, due to the model's bias toward the Non-Diabetic class, feature influence on minority classes was limited.

```
# --- SHAP KernelExplainer ---

def get_class_predictions(model, class_ind, input_data):
    return model.predict(input_data)[:, class_ind]

X_train = X_train if isinstance(X_train, np.ndarray) else X_train.values
X_test = X_test if isinstance(X_test, np.ndarray) else X_test.values

background = X_train[:100]
test_samples = X_test[:50]

num_classes = y_encoded.shape[1]

for class_ind in range(num_classes):
    print(f"\nExplaining Class {class_ind} with SHAP:")

    explainer = shap.KernelExplainer(lambda x:
                get_class_predictions(model, class_ind, x), background)

    shap_values = explainer.shap_values(test_samples)

    print(f"SHAP values shape: {np.array(shap_values).shape}")
```



- For the Autoencoder-Enhanced DNN Classifier:
  The SHAP analysis confirmed similar feature importance patterns — BMI, General Health, and High Blood Pressure remained the top features. Additionally, the Autoencoder model demonstrated better feature influence diversity, contributing more effectively to predictions across all classes.

```python
# --- SHAP Value Analysis ---
feature_names = data.drop('Diabetes_012', axis=1).columns.tolist()
def get_class_predictions(full_model, class_ind, input_data):
    encoded = encoder.predict(input_data)
    preds = classifier.predict(encoded)
    return preds[:, class_ind]

X_train = X_train if isinstance(X_train, np.ndarray) else X_train.values
X_test = X_test if isinstance(X_test, np.ndarray) else X_test.values

# Background and test samples
background = X_train[:100]
test_samples = X_test[:50]

num_classes = 3  # Classes are 0, 1, 2

for class_ind in range(num_classes):
    print(f"\nExplaining Class {class_ind} with SHAP:")

    # Define explainer for each class separately
    explainer = shap.KernelExplainer(lambda x:
                            get_class_predictions((encoder, classifier),
                            class_ind, x), background)

    shap_values = explainer.shap_values(test_samples)

    print(f"SHAP values shape for class {class_ind}: {np.array(shap_values).shape}")

    # Plot SHAP summary plot
    shap.summary_plot(shap_values, test_samples, feature_names=feature_names)
```



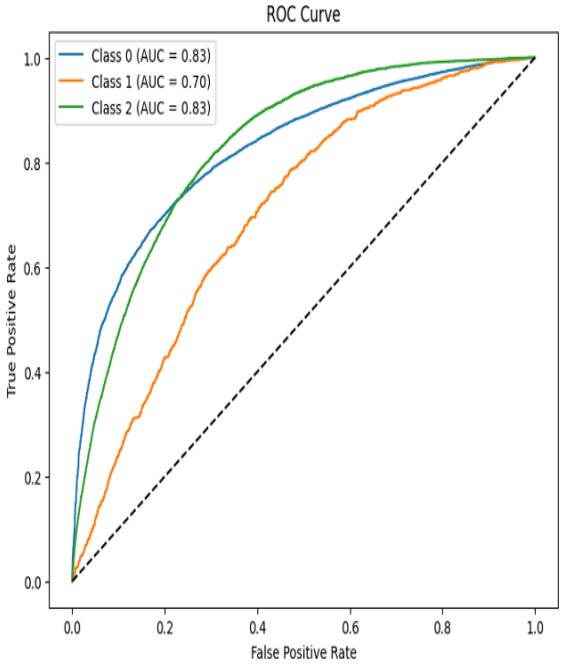
Conclusion from SHAP Analysis:

The similarity in top features across both models validated the clinical relevance of the dataset, while the broader feature impact in the Autoencoder classifier supported its better minority class sensitivity.

9. AUC-ROC Evaluation

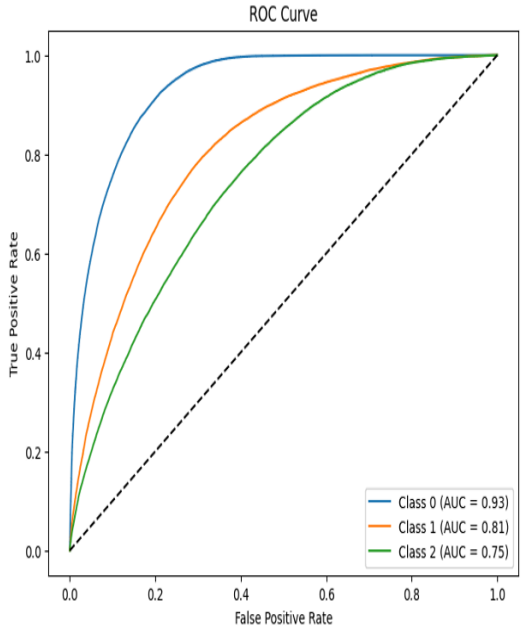
While the baseline DNN achieved higher AUC values overall, it heavily favored the Non-Diabetic class.

```python
# ROC Curve and AUC
fpr = {}
tpr = {}
thresh ={}
roc_auc = {}

for i in range(3):
    fpr[i], tpr[i], thresh[i] = roc_curve((y_test_classes==i).
                                          astype(int), y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8,6))
for i in range(3):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

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



The Autoencoder DNN presented more balanced ROC curves across all three classes, aligning better with healthcare objectives where minority class detection is vital.

```python
# ROC Curve and AUC
fpr = {}
tpr = {}
thresh ={}
roc_auc = {}

for i in range(3):
    fpr[i], tpr[i], thresh[i] = roc_curve((y_test==i).
    astype(int), y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8,6))
for i in range(3):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

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



10. Comparative Study

In addition to building our own models, we reviewed the work done by Alaa Sweed on the same dataset, where three machine learning models — Logistic Regression, K-Nearest Neighbors (KNN), and XGBoost — were compared for diabetes prediction.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	73.29%	72.47%	75.11%	73.76%
XGBoost	Slightly higher recall (77.25%)	71.63% precision	74.33% F1-Score	
K-Nearest Neighbors	69.65%	Lower performance overall		70.26% F1-Score

From the comparative analysis:

- Logistic Regression showed a good balance between precision and recall, making it suitable for general prediction tasks.
- XGBoost offered slightly better sensitivity (recall), meaning it detected more positive diabetic cases, though with slightly more false positives.
- KNN had the lowest performance among the compared models.

Compared to these models, our Deep Neural Network achieved a much higher accuracy (~85%) but lacked fairness, while the Autoencoder-enhanced DNN achieved moderate accuracy (~64%) but much better fairness and balance across all classes.
Thus, while simpler models performed decently, advanced techniques like Autoencoders proved essential for real-world healthcare tasks, where equity among classes matters more than sheer accuracy.

11. Conclusion

This project highlights the critical importance of designing models that balance overall accuracy with fair performance across minority classes, especially in healthcare settings.

The baseline Deep Neural Network, although achieving a high accuracy of 84.92%, failed to adequately predict Pre-Diabetic and Diabetic individuals, showing near-zero recall and F1-Score for the minority class.In contrast, the Autoencoder-enhanced DNN achieved better class balance, demonstrating the benefits of feature learning and oversampling strategies like SMOTE.

Furthermore, SHAP analysis confirmed that the models' important features — BMI, General Health, High Blood Pressure, and Physical Activity — aligned with known clinical factors affecting diabetes risk.
AUC-ROC curves further emphasized the importance of model fairness over simple accuracy in real-world applications. Overall, this project demonstrates that deep learning models must be evaluated not just on total accuracy but on their ability to fairly detect all classes, particularly in sensitive fields like healthcare.

12. Challenges and Future Work

Challenges Faced:

- Severe Class Imbalance: Even after SMOTE application, achieving balanced performance on the imbalanced test set remained difficult.
- Overfitting Risks: High model complexity in DNNs made regularization (e.g., Dropout, Early Stopping) crucial.
- Minority Class Detection: Achieving satisfactory recall and F1-scores for Pre-Diabetic and Diabetic individuals was challenging.
- Feature Complexity: Capturing non-linear relationships between diverse health features required deeper models and feature engineering.

Future Work:

- Cost-Sensitive Learning: Incorporating class-weighted loss functions could prioritize minority class predictions.
- Ensemble Techniques: Combining DNNs with boosting methods (e.g., XGBoost, LightGBM) may further enhance fairness.

- Transfer Learning: Pre-training Autoencoders on larger healthcare datasets could improve feature generalization.
- Advanced SMOTE Variants: Techniques like Borderline-SMOTE or ADASYN may create more realistic minority samples.

13. Contributions

The responsibilities for this project were divided among the group members to ensure an equitable workload and comprehensive coverage of all components:

- Daivik Girish and Sai Sashank Pannala focused on the Baseline Deep Neural Network (DNN) model.
  Their contributions included data preprocessing, application of SMOTE to handle class imbalance, development and training of the baseline DNN model, hyperparameter tuning, evaluation of model performance, and preparation of key parts of the final report and presentation.
- Grahitha Eda and Rushmitha Kommana focused on the Autoencoder-Enhanced DNN model.
  They were responsible for designing and training the Autoencoder architecture, extracting compressed feature representations, building and evaluating the subsequent classifier, conducting SHAP analysis for model interpretability, and contributing significantly to the exploratory data analysis and interpretation of results.

All team members actively participated in the final comparative analysis, conclusions, future work suggestions, and the compilation and editing of the final report and presentation materials.

14. References

[1] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research.

[2] Chollet, F. (2015). *Keras: Deep Learning for Humans*. GitHub Repository.

[3] Lundberg, S.M., & Lee, S.I. (2017). *A Unified Approach to Interpreting Model Predictions* (SHAP).

[4] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media.

[5] Kaggle Datasets and Community Resources.