Predicting Diabetes: A Comparative Study of Machine Learning Algorithms for Classification

Group 7

Hemlatha Kaur Saran, George David Asirvatharaj, Raminder Singh

Module: Introduction to Artificial Intelligence (AAI-501-IN2)

Course: Master in Applied Artificial Intelligence

Institution: University Of San Diego

Professor: Ms. Azka

Date – 11/Aug/2025

Contents

Predicting Diabetes: A Comparative Study of Machine Learning Algorithms for Classification

Overview of Diabetes

Diabetes is an ongoing disease that afflicts millions of people all over the planet who have high levels of sugar in their blood. According to the World Health Organization (WHO), over 422 million of the world's population have diabetes, and this figure keeps on increasing (World, 2024). The condition comes with considerable complications: heart diseases, kidney failures, and loss of vision, imposing a great cost to the health systems and economy of the population at large.

Machine Learning in Healthcare

Healthcare has been changed due to the inclusion of machine learning (ML) that has allowed predictive models to help diagnose and predict a disease. More specifically, ML algorithms are capable of detecting trends in complicated healthcare-related information, including medical records and patient history (Olalekan Kehinde, 2025). These models have been strong in disease prediction of such illnesses as diabetes, empowering early diagnosis, individualized treatment approaches, and patient outcomes, becoming one of the sources of change in medical practice across the world.

Purpose and Objectives

This report aims to use machine learning to predict diabetes with the help of a dataset consisting of several health characteristics. Such models as logistic regression, random forest, XGBoost, and K-nearest neighbors (KNN) will be applied to cluster patients into diabetic and non-diabetic. Its primary goals would be data preprocessing, metrics assessment of the model

performance, and the choice of the most adequate model depending on the accuracy and interpretability.

Exploratory Data Analysis (EDA)

Table 1

Basic Statistics for Numerical Features

Variable	Count	Mean	Std Dev	Min	25%	50%	75%	Max
Age	100,000	41.89	22.52	0.00	24.00	43.00	60.00	95.00
Hypertension	100,000	0.07	0.26	0.00	0.00	0.00	0.00	1.00
Heart Disease	100,000	0.04	0.19	0.00	0.00	0.00	0.00	1.00
BMI	100,000	27.32	6.64	10.01	23.63	27.32	29.59	95.60
HbA1c Level	100,000	5.53	1.08	3.50	4.00	5.00	6.20	9.00
Blood Glucose Level	100,000	138.06	40.71	80.00	100.00	140.00	159.00	300.00
Diabetes	100,000	0.09	0.28	0.00	0.00	0.00	1.00	1.00

Overview of Diabetes

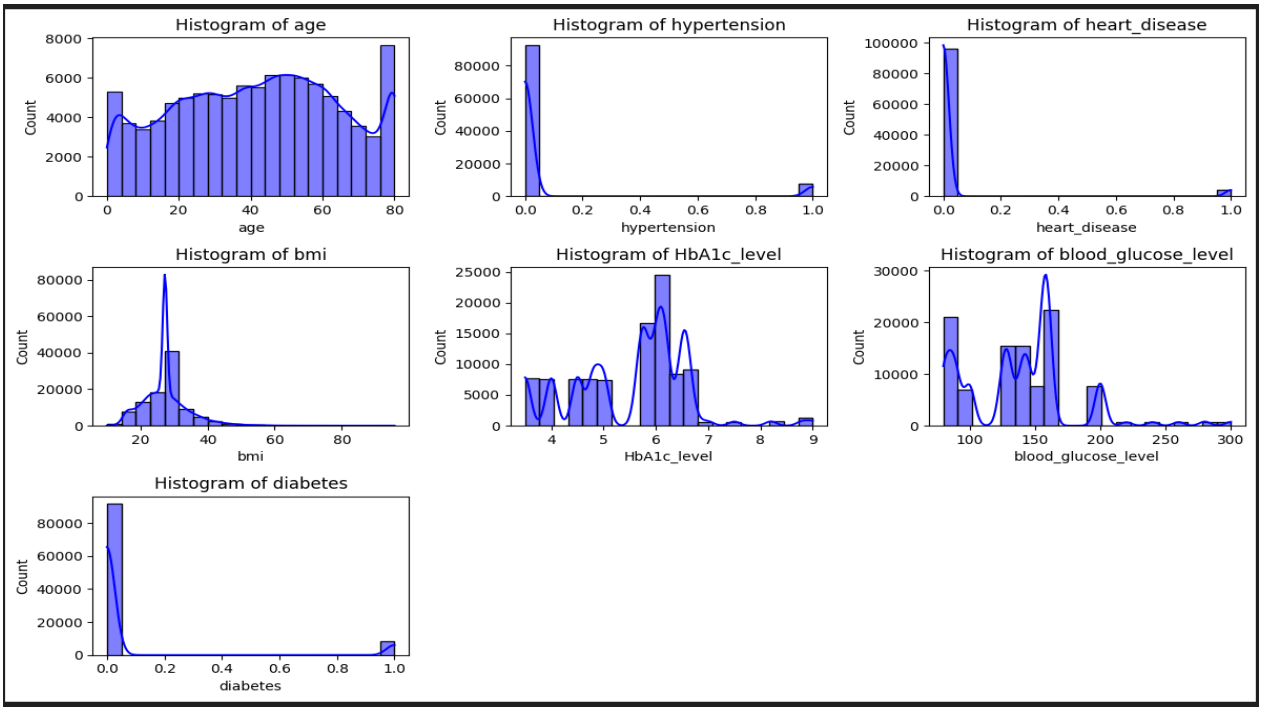
Diabetes is a serious health problem in the world with a total population of more than 420 million, and more than 50 percent of them are known to have type 2 diabetes (Basith et al.,

2019). The condition is linked to dire health risks, such as heart diseases, damage in kidneys, and blindness. The average age of people in the data is 41.89 years, and the mean BMI equals 27.32, and the mean blood glucose is 138.06, with the shown effect of lifestyle-related factors and reasons to be diagnosed and treated as early as possible.

Univariate Analysis

Figure 1

Histograms of numerical features in the diabetes dataset.



Note. The histograms illustrate the distribution of numerical variables such as age, BMI, HbA1c level, and blood glucose level. The target variable, diabetes, shows a highly imbalanced distribution with more non-diabetic cases than diabetic ones.

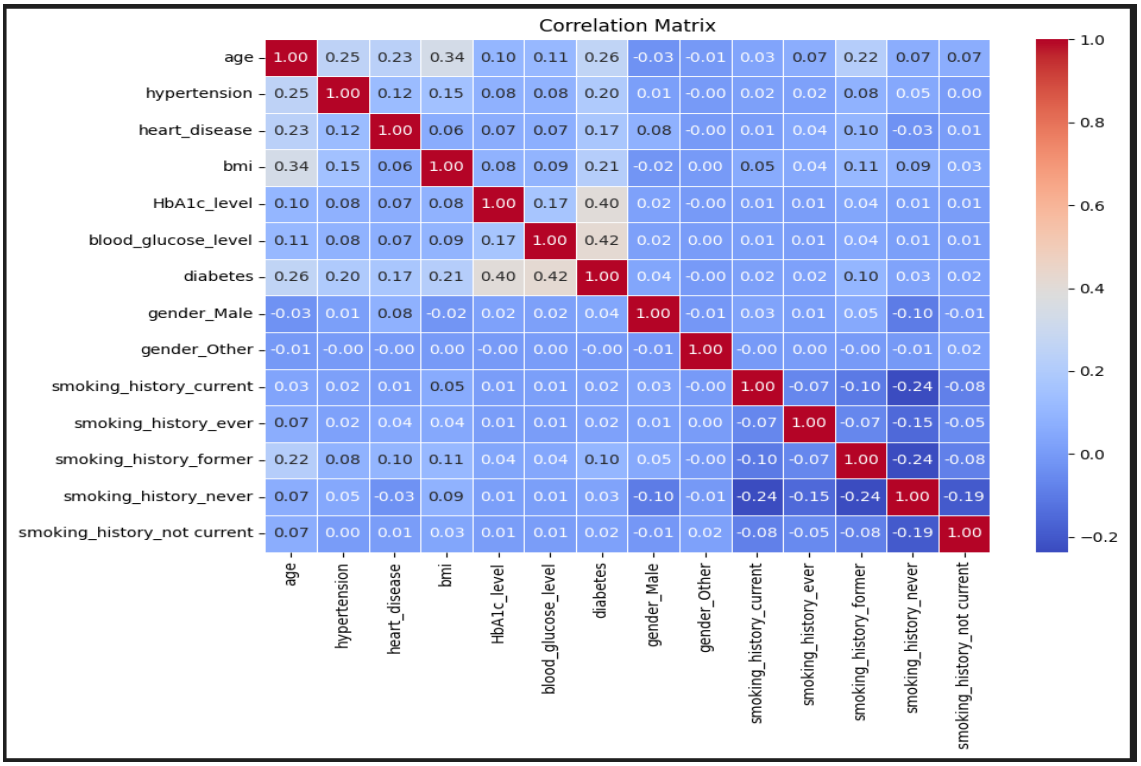
The histograms show graphically the number of occurrences of numerical data like age, BMI, HbA1c, and blood glucose levels. The age attribute has a maximum at 50 years, signifying that many of the people would be in middle age. The BMI shows a high peak of 30 that indicates

more people are overweight or obese. The concentration of HbA1c level is maximized at 6 with some elevated measurements. The concentration of blood glucose has been highly concentrated between 100 and 150 mg/dL. The distribution of diabetes is very skewed, as the cases of non-diabetes are bigger in number.

Correlation and Feature Relationships

Figure 2

Correlation Matrix of Numerical and Categorical Features in the Diabetes Dataset.



Note. The heatmap visualizes the correlation between features in the dataset. Stronger correlations are indicated by red, and weaker correlations are shown in blue. Notably, blood glucose and HbA1c levels have a significant positive correlation, and diabetes is moderately correlated with age and BMI.

The above heatmap shows the pairwise feature correlations. HbA1c level also has a strong positive relationship with blood glucose level of 0.42, which shows that an increase in blood glucose level implies an increase in HbA1c level. The correlations between age and BMI with diabetes are also positive but moderate (0.26 and 0.21, respectively), which implies that diabetes is more likely to occur in older people and in people who have higher BMI. Other features have looser ties to other features and the target variable.

Data Preprocessing

Explanation of Data Collection and Dataset Characteristics

The dataset utilized in this study was an open public one related to the prediction of diabetes. It contains 100,000 rows and 8 columns featuring age, gender, the presence of hypertension, heart disease, body mass index (BMI), level of HbA1c, level of blood glucose, and history of smoking. The target variable is the presence or absence of diabetes (1) or not (0). The features act as predictors when determining the existence or the nonexistence of diabetes.

Handling Missing Values, Outliers, and Data Normalization

To address the incompleteness, mean imputation was implemented on numerical characteristics. In the case of categorical variables, missing values were replaced by mode. The outliers were determined by the IQR method, such as capping or removing, depending on the influence of the outliers on the model. To standardize the numerical variable, the StandardScaler was used, and to adjust for the sensitivity of the models to feature scaling, such as logistic regression and KNN, we required all features to have a mean of 0 and a standard deviation of 1.

Feature Engineering Techniques

The categorical variables gender and smoking history were transformed by one-hot encoding (using pd.get_dummies) into binary form. Take the example of the gender variable,

which was transformed to two columns; one was for Male, and the other was Other (without Female as the reference group). Smoking history also underwent such a transformation and resulted in a binary feature such as current smoker, former smoker, and never smoker. These ones enable the model to work with categorical variables well in the training process.

Overview of Machine Learning Models

Logistic Regression

Table 2

Evaluation Metrics for Logistic Regression and Random Forest Models

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
Logistic Regression	0.9590	0.8646	0.6171	0.7202	0.8040
Random Forest	0.9699	0.9467	0.6868	0.7961	0.8416

Note. Logistic Regression and Random Forest evaluation metrics are based on their performance in predicting diabetes using the provided dataset.

Logistic regression is an efficient but easy model of binary classification. It avails the interpretability of estimating the probability of an outcome on the basis of input features. In this diabetic prediction model, the logistic regression had a performance accuracy of 95.90%, a precision of 86.46%, and a recall of 61.71%, showing that although the model performs sufficiently in distinguishing between the non-diabetic instances, it does not perform well when

it comes to recognizing the diabetic instances. F1 was 0.7202, and ROC AUC was 0.8040, which displays a satisfactory result.

Feature Importance (Random Forest)

Table 3

Feature Importance (Random Forest)

Feature	Importance
HbA1c_level	0.397138
blood_glucose_level	0.329611
bmi	0.121995
age	0.100404
hypertension	0.014600
heart_disease	0.010685
gender_Male	0.007010
smoking_history_never	0.005169
smoking_history_former	0.004357
smoking_history_current	0.003233
smoking_history_not current	0.003071
smoking_history_ever	0.002724
gender_Other	0.000003

Note. Feature importance values from the Random Forest model show the contribution of each feature to the diabetes prediction task. Higher values indicate greater importance in the model's decision-making process.

The Random Forest algorithm also gives good clues as to which features are the most influential following the predictions of diabetes. The level of HbA1c and the level of blood glucose have the most significance, having the relative importance of 0.3971 and 0.3296,

respectively, and thus are the most relevant ones in predicting diabetes. Other significant characteristics are the BMI (0.1220) and age (0.1004) that also significantly contribute towards the decisions made by the model. Such features as heart disease (0.0107) and gender (0.0000) are of very low importance as compared to the others.

XGBoost and KNN

Table 4

Evaluation Metrics for XGBoost and K-Nearest Neighbors (KNN)

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
XGBoost	0.9717	0.9568	0.6996	0.8083	0.8483
K-Nearest Neighbors (KNN)	0.9607	0.8960	0.6107	0.7263	0.8020

Note. Evaluation metrics for XGBoost and K-Nearest Neighbors (KNN) models, showcasing their performance on predicting diabetes. The XGBoost model outperforms KNN, particularly in accuracy and ROC AUC.

XGBoost is a powerful model that runs on gradient boosting, and it is also great with imbalanced data. XGBoost presented an accuracy of 97.17 percent, a precision of 95.68 percent, and a recall of 69.96 percent in the given diabetes prediction problem. This showed a good precision versus recall balance F1 of 0.8083. The ROC AUC value of 0.8483 also shows that it is able to, in a strong way, separate the classes compared to having simpler models such as the logistic regression.

Description of Model Evaluation Metrics

Accuracy is among the most frequently used evaluation metrics, as it assesses the percentage of correct predictions conducted by a model (Owusu-Adjei et al., 2023). It is defined as the fraction of correct predictions and the total number of predictions. Although accuracy gives a rough measure of the model performance, it is a wrong measure in the imbalanced datasets, wherein a type of class might dominate the estimates. On the other hand, precision is concerned with the accuracy of positive predictions by the model. It is calculated as the proportion of the true positive predictions to the number of the true positives plus false predictions. When the high cost comes in the form of false positives, the exactitude is essential, as in the case of diagnosing a disease like diabetes, where a false classification of someone healthy as diabetic would result in him or her receiving unwarranted care. Recall, also referred to as sensitivity, is defined as the ratio of the true positives to the actual positives. It should be considered where it is more costly to miss a positive (false negative) than a false positive, e.g., medical diagnosis where there are serious consequences due to failure to identify a diabetic person. The F1 score is the harmonic mean of precision and recall, and it provides a balanced value of both when false positives and false negatives are of interest. Lastly, the AUC-ROC curve measures the discrimination of a model in terms of classes, where a high value (2 is near-perfect) denotes higher performance, particularly when dealing with imbalanced data.

Justification for Algorithm Selection

The four algorithms (logistic regression, random forest, XGBoost, and K-nearest neighbor (KNN)) selected to undertake this test of prediction of diabetes were arrived at based on their capability of handling the profile of the dataset, as well as their effectiveness in binary classification. Logistic regression was selected because of its ease of interpretation and the

ability to understand the relationship between features and diabetes; this is of importance in the medical field. It chose Random Forest because it is robust, accurate, and purchasable in a linear and non-linear relationship, besides its ability to assign feature relevance, which is useful in the determination of the most influential factors in the prediction of diabetes (Afolabi et al., 2024). It was decided that the model should be XGBoost because it performs best when facing an imbalanced dataset, e.g., the population of non-diabetic people tends to exceed the numbers of diabetic ones. Due to its high accuracy and speed, it is good with large datasets. At last, a simple baseline model, KNN, was also added as the possibility to compare a model's performance with more complex models and see how the developed model complexity affects the prediction accuracy.

Discussion of Hyperparameter Tuning and Cross-Validation

GridSearchCV was used to tune the hyperparameter, where an exhaustive search is undertaken on the hyperparameter values provided to determine the optimal combination that can enhance model performance. The best hyperparameters of Random Forest were max_depth=10 and n_estimators=100, which means that the depth of trees needs to be limited and the number of trees should be set to 100. And the optimal hyperparameter in K-Nearest Neighbors (KNN) is n_neighbors=7 since this is the value that balances the complexity of the model and its performance that considers 7 nearest neighbors to classify. K-fold cross-validation guarantees that the performance of the model is tested against various subsets of the data, and the results would be robust and generalized, and issues of overfitting are minimized, and also the reliability of the model is improved with the results obtained.

Results and Discussion

Summary of Performance Metrics

The performance of each model based on their evaluation results is impressive, with Random Forest ranking ahead, as it achieved an accuracy of 96.99% and a precision of 94.67, and XGBoost almost followed with an accuracy of 97.17% and a precision of 95.68. Both models have high recall scores, which are 69.96 percent and 68.68 percent, respectively, and thus demonstrate that they are effective in identifying positive cases of diabetics. Logistic regression with the accuracy of 95.90 also proved to be very accurate but had a lower recall, 61.71, and thus, it was not as good at identifying diabetic patients as the ensemble models. KNN was a good baseline, where it achieved a 96.07% rate of accuracy but with a low rate of recall of 61.07 and a rate of precision of 89.60.

Analysis of Strengths and Weaknesses

Random forest and XGBoost are good models at a high level in terms of accuracy, precision, and recall, but XGBoost shone when working with imbalanced data. Logistic regression is easy to understand and interpret, and its low recall rate does not seem to fit well for the purpose of medical diseases when one wants to detect all the positive cases. KNN is simple to apply but not good on huge data and complex separation boundaries. It also has a low recall score, which means that it is not as reliable for making healthcare predictions and that it is also worse with diabetic patients.

Comparison of Models

XGBoost and Random Forest perform better than Logistic Regression and KNN, both according to accuracy and recall as they were directly compared, which makes the former more suitable as an approach to this diabetes predictive forecast. XGBoost takes the first step with a

bit better AUC-ROC and accuracy, which means better performance. The upside of Random Forest is that it gives feature importance, which comes in handy when it comes to comprehending the influence of each feature. But for logistic regression and KNN, there are simpler models that are likely to be more explainable but cannot be as predictive as the use of the ensemble methods.

Conclusion and Future Directions

The findings show that XGBoost and Random Forest models are the best predictive models of diabetes since they give high values in accuracy, precision, and recall, and XGBoost performs better in overall predictive power. Such models are best suited for the handling of imbalance data and provide good performance. Logistic regression and KNN can be applied in low complex tasks yet cannot be utilized in the detection of all cases of diabetes which is a prerequisite in the performance of healthcare.

Further improvement of the model performance can be made by considering other features such as genetic information or lifestyle parameters to model observation with greater accuracy. It may be considered to use cutting-edge algorithms that can process large and complex information, including deep learning. Additionally, the entire set of hyperparameters can be optimized further, and a more sophisticated technique should also be employed, e.g., an ensemble learning or feature selection, which could be helpful in terms of optimization of its performance. Regarding the idea of cross-validation by stratified samples, it would also offer leads on the aspects of correcting the class imbalances in being more accurate on the class predictions.

References

Afolabi, S., Nurudeen Ajadi, Jimoh, A., & Ibrahim Adenekan. (2024). Predicting Diabetes Using Supervised Machine Learning Algorithms. *Research Square (Research Square)*. https://doi.org/10.21203/rs.3.rs-4527374/v1

Basith, A., Hashim, M. J., King, J. K., Govender, R. D., Mustafa, H., & Juma Al Kaabi. (2019). Epidemiology of Type 2 Diabetes – Global Burden of Disease and Forecasted Trends. *Journal of Epidemiology and Global Health*, *10*(1), 107–107. https://doi.org/10.2991/jegh.k.191028.001

Olalekan Kehinde. (2025). Machine Learning in Predictive Modelling: Addressing Chronic Disease Management through Optimized... *International Journal of Research Publication and Reviews*, *6*(1), 1525–1539. https://www.researchgate.net/publication/388123356_Machine_Learning_in_Predictive_Modelling_Addressing_Chronic_Disease_Management_through_Optimized_Healthcare_Processes

Owusu-Adjei, M., Ben Hayfron-Acquah, J., Frimpong, T., & Abdul-Salaam, G. (2023). Imbalanced class distribution and performance evaluation metrics: A systematic review of prediction accuracy for determining model performance in healthcare systems. *PLOS Digital Health*, *2*(11), e0000290. https://doi.org/10.1371/journal.pdig.0000290

World. (2024, November 14). *Diabetes*. Who.int; World Health Organization: WHO. https://www.who.int/news-room/fact-sheets/detail/diabetes

Appendix

Python based Jupyter Notebook Code Details

GitHub Repo url: https://github.com/ramindersinghusd/aai-501-in2-project-group7

USD-MS AA1-501-IN2 - Diabetes Prediction Project

- Group: 7
- Names: Hemlatha Kaur Saran, George David Asirvatharaj, Raminder Singh
- Module: Introduction to AI
- Date: 10.Aug.2025

Local Development Env. Setup

1. Install python: 3.11.3
2. Installed VSCode
3. Add Python and jupyter extension
4. Set kernel
5. conda install -n base ipykernel jupyter
6. conda -V >> conda 23.5.2
7. pip install jupyter notebook pandas numpy matplotlib scipy scikit-learn pandoc nbconvert[webpdf] nbconvert notebook-as-pdf seaborn xgboost shap openpyxl
8. run >> jupyter notebook
9. Github url for code repo: https://github.com/usd-ms-aai/aai-501-in2-project-group7

In [49]:
```
# Import necessary libraries for EDA and Model processing
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier

import xgboost as xgb
import matplotlib.pyplot as plt # Importing the necessary module for plotting
import seaborn as sns
```

In [50]:
```
#1 Data Loading from dataset - the xlsx file
print("Loading Data")
data = pd.read_excel('diabetes_prediction_dataset.xlsx')
print("Dataset loaded successfully.....")

# Display basic information about the dataset
print("Dataset Info:")
data.info()
```

```
print(">>>> First 5 rows of the dataset:")
print(data.head())
```

```
Loading Data
Dataset loaded successfully.....
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   gender               100000 non-null  object 
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64  
 3   heart_disease        100000 non-null  int64  
 4   smoking_history      100000 non-null  object 
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64  
 8   diabetes             100000 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB
>>>> First 5 rows of the dataset:
   gender   age  hypertension  heart_disease smoking_history    bmi  \
0  Female  80.0             0              1           never  25.19   
1  Female  54.0             0              0         No Info  27.32   
2    Male  28.0             0              0           never  27.32   
3  Female  36.0             0              0         current  23.45   
4    Male  76.0             1              1         current  20.14   

   HbA1c_level  blood_glucose_level  diabetes  
0          6.6                  140         0  
1          6.6                   80         0  
2          5.7                  158         0  
3          5.0                  155         0  
4          4.8                  155         0  
```

In [51]:
```
#2 Encode categorical variables using pd.get_dummies() (this will convert 'gender'
data = pd.get_dummies(data, drop_first=True)
```

In [52]:
```
#3 Define the features (X) and target variable (y)
print("Separate features / target variables...")
X = data.drop('diabetes', axis=1) # Features
y = data['diabetes']              # Target variable
print("Features (X) shape:", X.shape)
print("Target (y) shape:", y.shape)
```

```
Separate features / target variables...
Features (X) shape: (100000, 13)
Target (y) shape: (100000,)
```

In [53]:
```
#4 Normalize the data (Standardization) - this is moved to 5.1
#scaler = StandardScaler()
#X_scaled = scaler.fit_transform(X)
```

In [54]:
```
#5 Split data into training and testing sets (80% train, 20% test)
# stratify=y ensures that the proportion of target classes is the same in both trai
# which is crucial for imbalanced datasets.
print("Train/test split....")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
print(f"Original X shape: {X.shape}")
print(f"X_train shape before preprocessing: {X_train.shape}")
print(f"X_test shape before preprocessing: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
Train/test split....
Original X shape: (100000, 13)
X_train shape before preprocessing: (80000, 13)
X_test shape before preprocessing: (20000, 13)
y_train shape: (80000,)
y_test shape: (20000,)
```

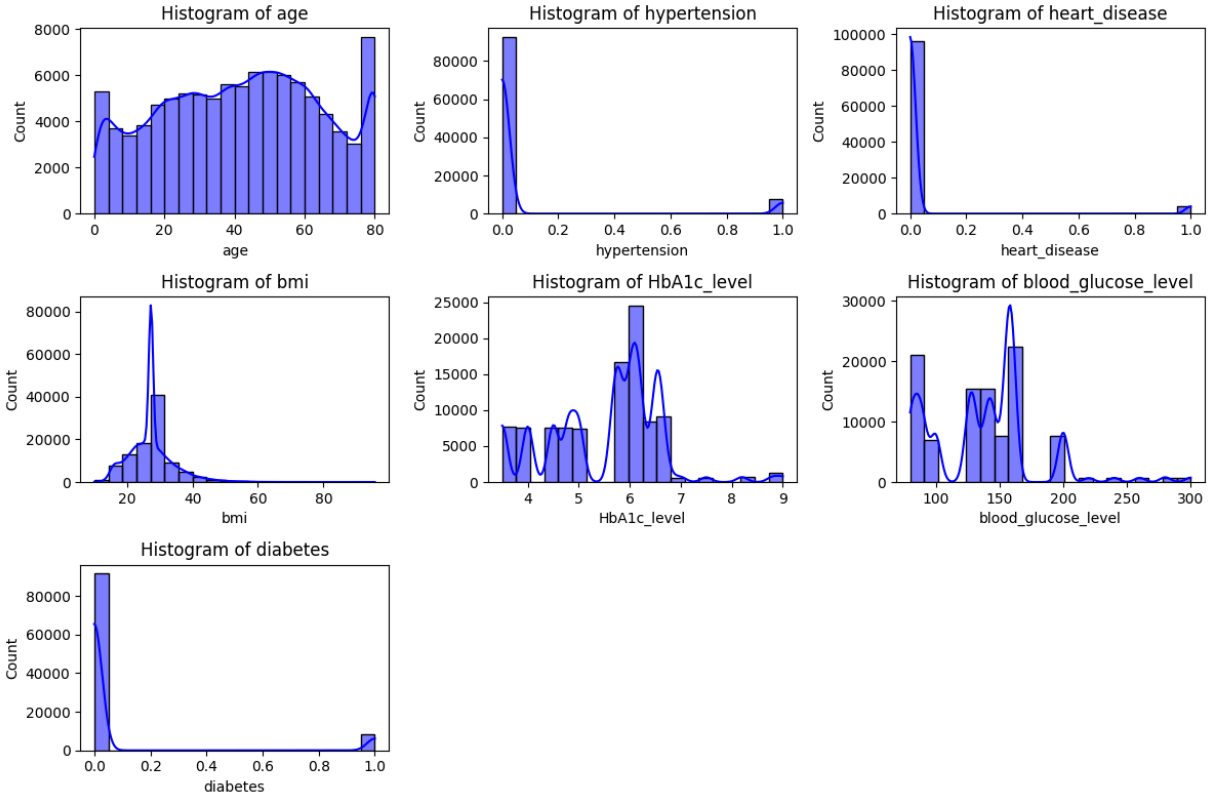
In [55]:
```
#5.1 Apply preprocessing to training and testing data

# Get feature names after one-hot encoding for categorical features.
# This is important for feature importance visualization later, so we know which co
print("Encode categorical features & Standardize features")
# Identify categorical and numerical features
categorical_features = X.select_dtypes(include=['object']).columns
numerical_features = X.select_dtypes(include=np.number).columns

print(f"Numerical features: {list(numerical_features)}")
print(f"Categorical features: {list(categorical_features)}")

# Numerical features will be standardized
# Categorical features will be one-hot encoded
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

# fit_transform on X_train to learn scaling parameters and encoding categories
X_train_processed = preprocessor.fit_transform(X_train)
# transform on X_test using parameters learned from X_train
X_test_processed = preprocessor.transform(X_test)

print(f"X_train shape after preprocessing: {X_train_processed.shape}")
print(f"X_test shape after preprocessing: {X_test_processed.shape}")

if len(categorical_features) > 0:
    ohe_feature_names = preprocessor.named_transformers_['cat'].get_feature_names_o
    all_feature_names = list(numerical_features) + list(ohe_feature_names)
else:
    all_feature_names = list(numerical_features)
```

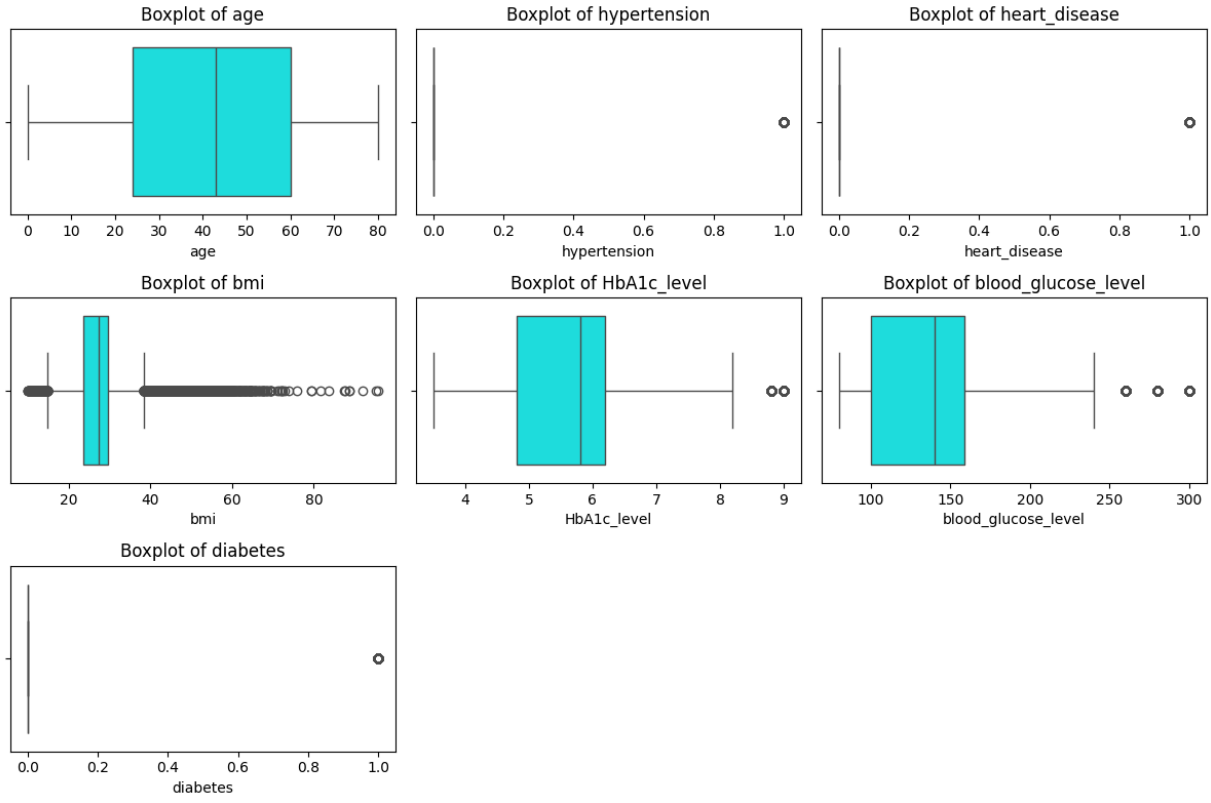
```
Encode categorical features & Standardize features
Numerical features: ['age', 'hypertension', 'heart_disease', 'bmi', 'HbA1c_level',
'blood_glucose_level']
Categorical features: []
X_train shape after preprocessing: (80000, 6)
X_test shape after preprocessing: (20000, 6)
```

In [56]:
```python
#6 Basic statistics for numerical features
print("\nBasic statistics for numerical features:")
print(data.describe())
```

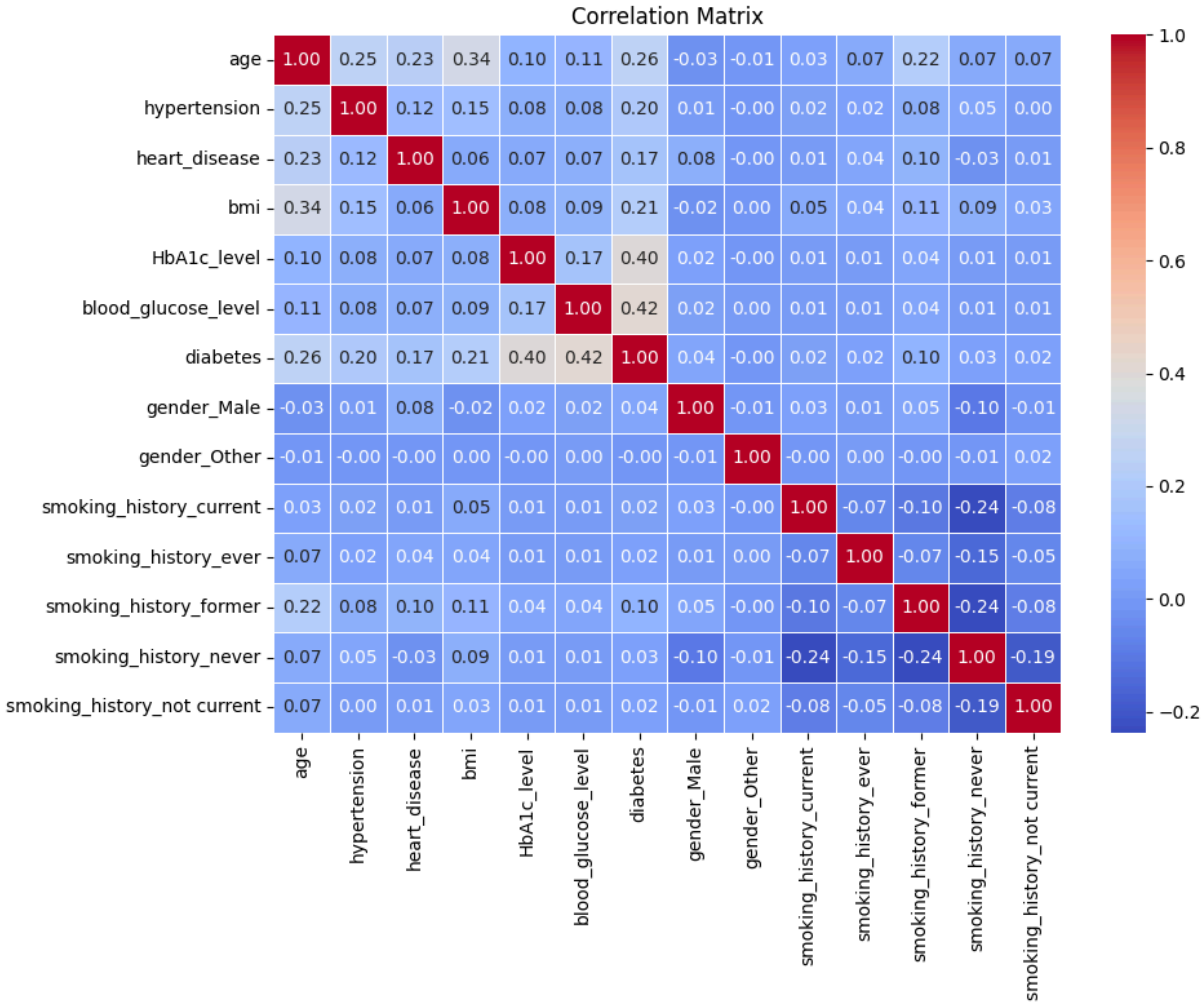
```
Basic statistics for numerical features:
                 age  hypertension  heart_disease            bmi  \
count  100000.000000  100000.00000  100000.000000  100000.000000
mean       41.885856       0.07485       0.039420      27.320767
std        22.516840       0.26315       0.194593       6.636783
min         0.080000       0.00000       0.000000      10.010000
25%        24.000000       0.00000       0.000000      23.630000
50%        43.000000       0.00000       0.000000      27.320000
75%        60.000000       0.00000       0.000000      29.580000
max        80.000000       1.00000       1.000000      95.690000

         HbA1c_level  blood_glucose_level       diabetes
count  100000.000000        100000.000000  100000.000000
mean        5.527507           138.058060       0.085000
std         1.070672            40.708136       0.278883
min         3.500000            80.000000       0.000000
25%         4.800000           100.000000       0.000000
50%         5.800000           140.000000       0.000000
75%         6.200000           159.000000       0.000000
max         9.000000           300.000000       1.000000
```

In [57]:
```python
#7 Visualizing the distribution of numerical features (histograms)
plt.figure(figsize=(12, 8))  # Set figure size
numeric_cols = data.select_dtypes(include=['float64', 'int64']).columns.tolist()
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(data[col], kde=True, bins=20, color='blue')
    plt.title(f'Histogram of {col}')
    plt.tight_layout()
plt.show()

# Visualizing the distribution of numerical features using boxplots
plt.figure(figsize=(12, 8))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(x=data[col], color='cyan')
    plt.title(f'Boxplot of {col}')
    plt.tight_layout()
plt.show()
```





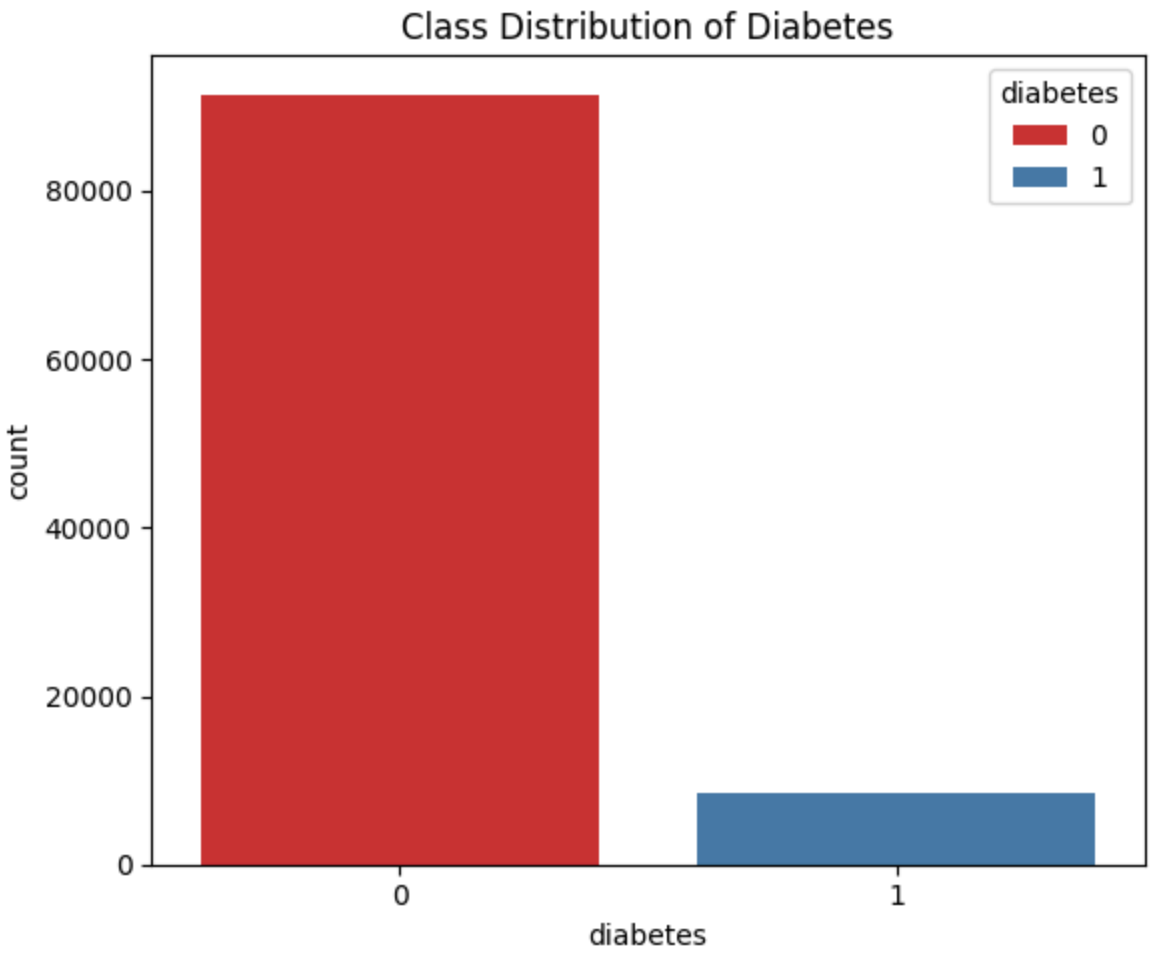
In [58]:
```
#8 Exploring the correlation matrix
correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=
plt.title('Correlation Matrix')
plt.tight_layout()
plt.show()
```



In [59]:
```
#9. Exploring categorical variables (e.g., 'gender', 'smoking_history')
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()
print(f"categorical_cols: {categorical_cols}")
for col in categorical_cols:
    plt.figure(figsize=(8, 5))
    sns.countplot(x=col, data=data, palette='Set2', hue=None)
    plt.title(f'Countplot of {col}')
    plt.tight_layout()
    plt.show()
```

```
categorical_cols: []
```

In [60]:
```
#10 Checking class distribution of the target variable 'diabetes'
plt.figure(figsize=(6, 5))
sns.countplot(x='diabetes', data=data, palette='Set1', hue='diabetes')
plt.title('Class Distribution of Diabetes')
plt.tight_layout()
plt.show()
```



In [61]:
```
print("Starting with 4 Models Training and Evaluation.....")
#models = {
#    "Logistic Regression": logistic_regression_model,
#    "Random Forest": random_forest_model,
#    "XGBoost": xgb_model,
#    "K-Nearest Neighbors (KNN)": knn_model
#}
```

```
Starting with 4 Models Training and Evaluation.....
```

In [62]:
```
#11 > Model1: LogisticRegression
# Initialize Logistic Regression model
# solver='liblinear' is good for small datasets and binary classification
logistic_regression_model = LogisticRegression(random_state=42, solver='liblinear')

# Train the model
print("Training Logistic Regression Model....")
logistic_regression_model.fit(X_train_processed, y_train)

# Make predictions on the test set
print("Predicting Logistic Regression Model....")
y_pred_logreg = logistic_regression_model.predict(X_test_processed)

# Display the evaluation metrics
print(f"Evaluation > Logistic Regression Model....")
# Evaluate the model using accuracy, precision, recall, F1 score, and AUC-ROC
```

```
accuracy = accuracy_score(y_test, y_pred_logreg)
precision = precision_score(y_test, y_pred_logreg)
recall = recall_score(y_test, y_pred_logreg)
f1 = f1_score(y_test, y_pred_logreg)
roc_auc = roc_auc_score(y_test, y_pred_logreg)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
```

```
Training Logistic Regression Model....
Predicting Logistic Regression Model....
Evaluation > Logistic Regression Model....
Accuracy: 0.9607
Precision: 0.8621
Recall: 0.6400
F1 Score: 0.7346
ROC AUC: 0.8152
```

In [63]:
```
#12 > Model2: Random Forest
# Initialize Random Forest model
random_forest_model = RandomForestClassifier(random_state=42)

print("Training Random Forest Classifier")
# Train the model
random_forest_model.fit(X_train_processed, y_train)

# Make predictions on the test set
y_pred_rf = random_forest_model.predict(X_test_processed)

# Display the evaluation metrics
print(f"Evaluation > Random Forest Model....")
# Evaluate the model using accuracy, precision, recall, F1 score, and AUC-ROC
accuracy = accuracy_score(y_test, y_pred_rf)
precision = precision_score(y_test, y_pred_rf)
recall = recall_score(y_test, y_pred_rf)
f1 = f1_score(y_test, y_pred_rf)
roc_auc = roc_auc_score(y_test, y_pred_rf)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
```

```
Training Random Forest Classifier
Evaluation > Random Forest Model....
Accuracy: 0.9686
Precision: 0.9117
Recall: 0.6982
F1 Score: 0.7908
ROC AUC: 0.8460
```

In [64]:
```
#13 > Model3: XGBoost
print("Training XGBoost Classifier...")
```

```
# objective='binary:logistic' for binary classification
# eval_metric='logloss' is a common evaluation metric for classification
# use_label_encoder=False suppresses a future warning
#xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss',
xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss', r

# Train the model
xgb_model.fit(X_train_processed, y_train)

# Make predictions on the test set
y_pred_xgb = xgb_model.predict(X_test_processed)

# Display the evaluation metrics
print(f"Evaluation > XGBoost Classifier Model....")
# Evaluate the model using accuracy, precision, recall, F1 score, and AUC-ROC
accuracy = accuracy_score(y_test, y_pred_xgb)
precision = precision_score(y_test, y_pred_xgb)
recall = recall_score(y_test, y_pred_xgb)
f1 = f1_score(y_test, y_pred_xgb)
roc_auc = roc_auc_score(y_test, y_pred_xgb)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
```

```
Training XGBoost Classifier...
Evaluation > XGBoost Classifier Model....
Accuracy: 0.9709
Precision: 0.9501
Recall: 0.6941
F1 Score: 0.8022
ROC AUC: 0.8454
```

In [65]:
```
#14 > Model4: K-Nearest Neighbors (KNN)
# Initialize K-Nearest Neighbors model
knn_model = KNeighborsClassifier()

# Train the model
knn_model.fit(X_train_processed, y_train)

# Make predictions on the test set
y_pred_knn = knn_model.predict(X_test_processed)

# Display the evaluation metrics
print(f"Evaluation > KNN Model....")
# Evaluate the model using accuracy, precision, recall, F1 score, and AUC-ROC
accuracy = accuracy_score(y_test, y_pred_knn)
precision = precision_score(y_test, y_pred_knn)
recall = recall_score(y_test, y_pred_knn)
f1 = f1_score(y_test, y_pred_knn)
roc_auc = roc_auc_score(y_test, y_pred_knn)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
```

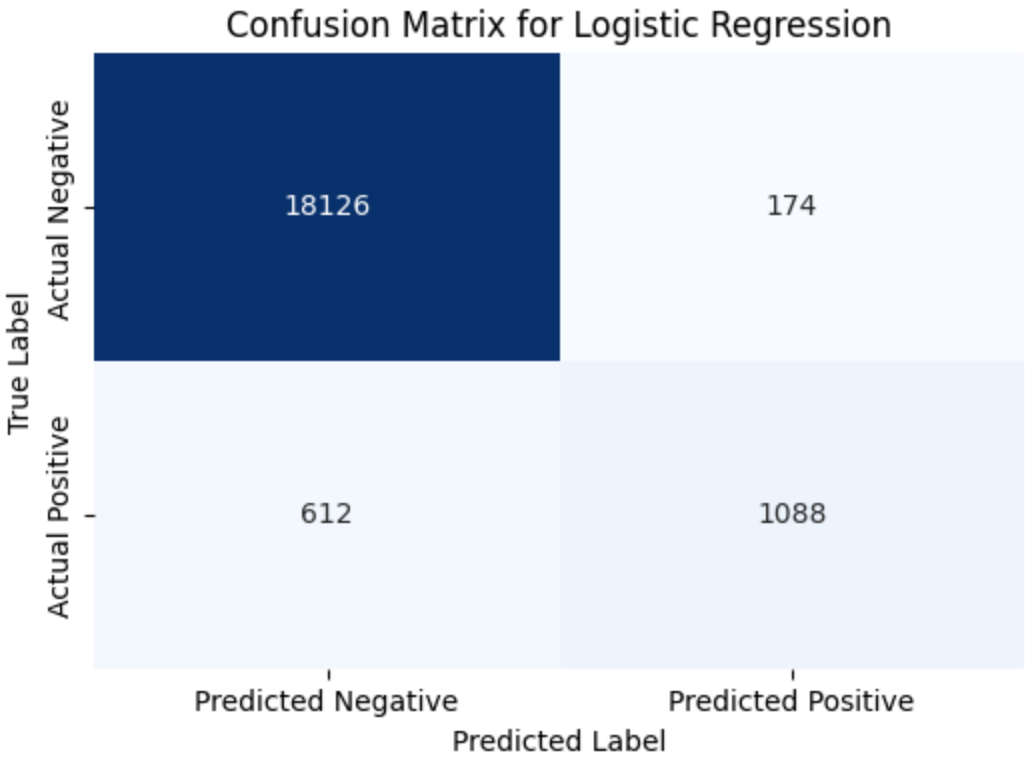
```
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
```

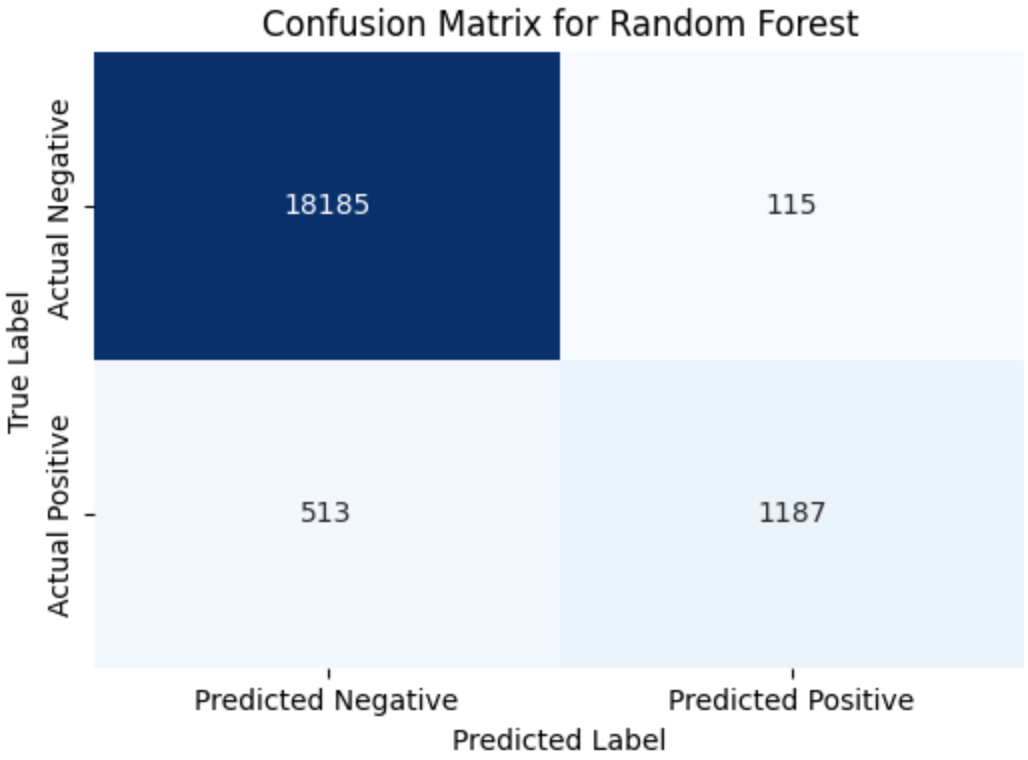
```
Evaluation > KNN Model....
Accuracy: 0.9654
Precision: 0.9078
Recall: 0.6600
F1 Score: 0.7643
ROC AUC: 0.8269
```

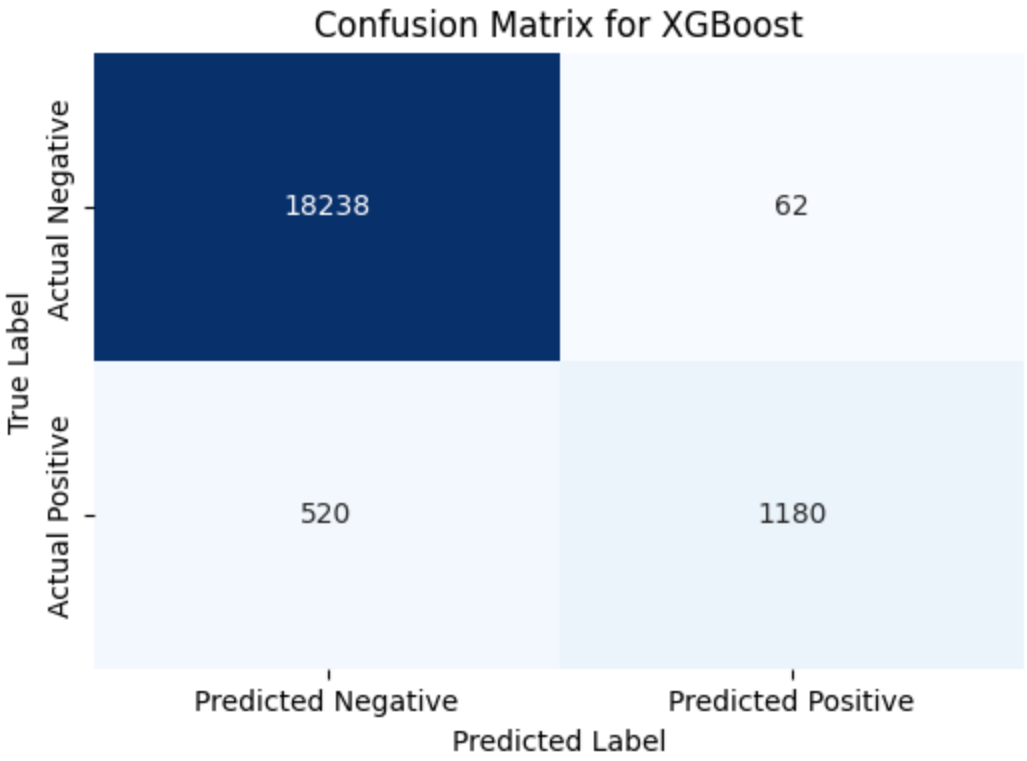
In [66]:

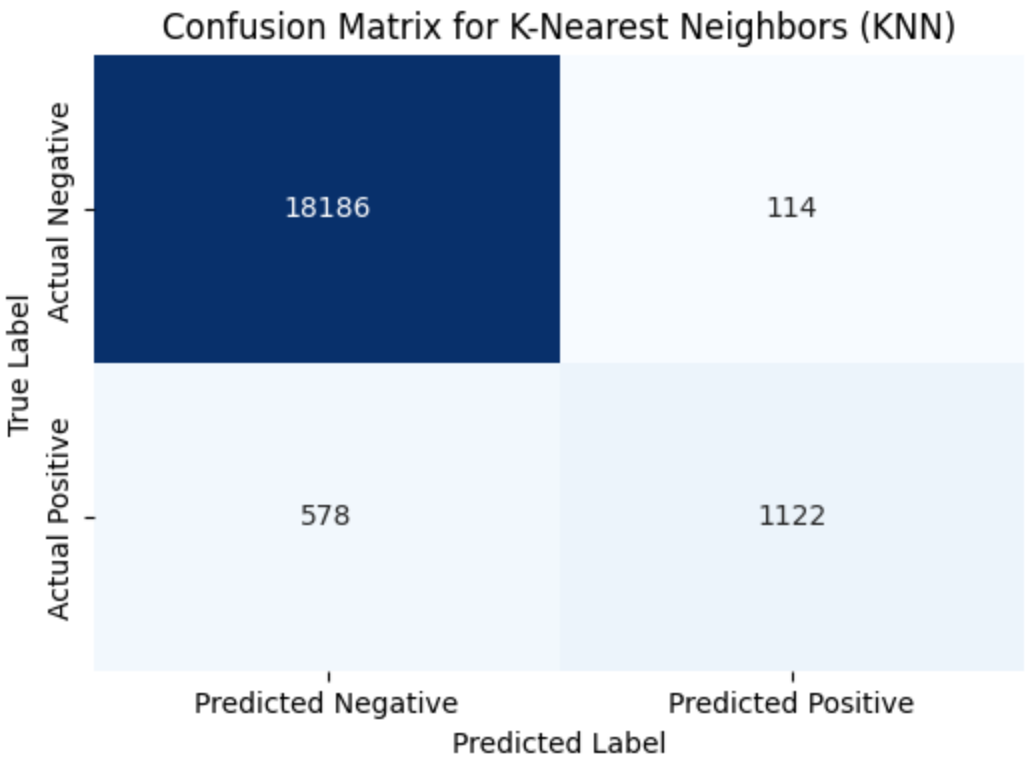
```
##15 Confusion Matrix for Models

#Model1: LogisticRegression
name ="Logistic Regression"
confu_matrix = confusion_matrix(y_test, y_pred_logreg)
plt.figure(figsize=(6, 4))
sns.heatmap(confu_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title(f'Confusion Matrix for {name}')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

#Model2: Random Forest
name ="Random Forest"
confu_matrix = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(6, 4))
sns.heatmap(confu_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title(f'Confusion Matrix for {name}')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

#Model3: XGBoost
name ="XGBoost"
confu_matrix = confusion_matrix(y_test, y_pred_xgb)
plt.figure(figsize=(6, 4))
sns.heatmap(confu_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title(f'Confusion Matrix for {name}')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()


#Model4: KNN
name ="K-Nearest Neighbors (KNN)"
confu_matrix = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(6, 4))
sns.heatmap(confu_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
```

```
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title(f'Confusion Matrix for {name}')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```





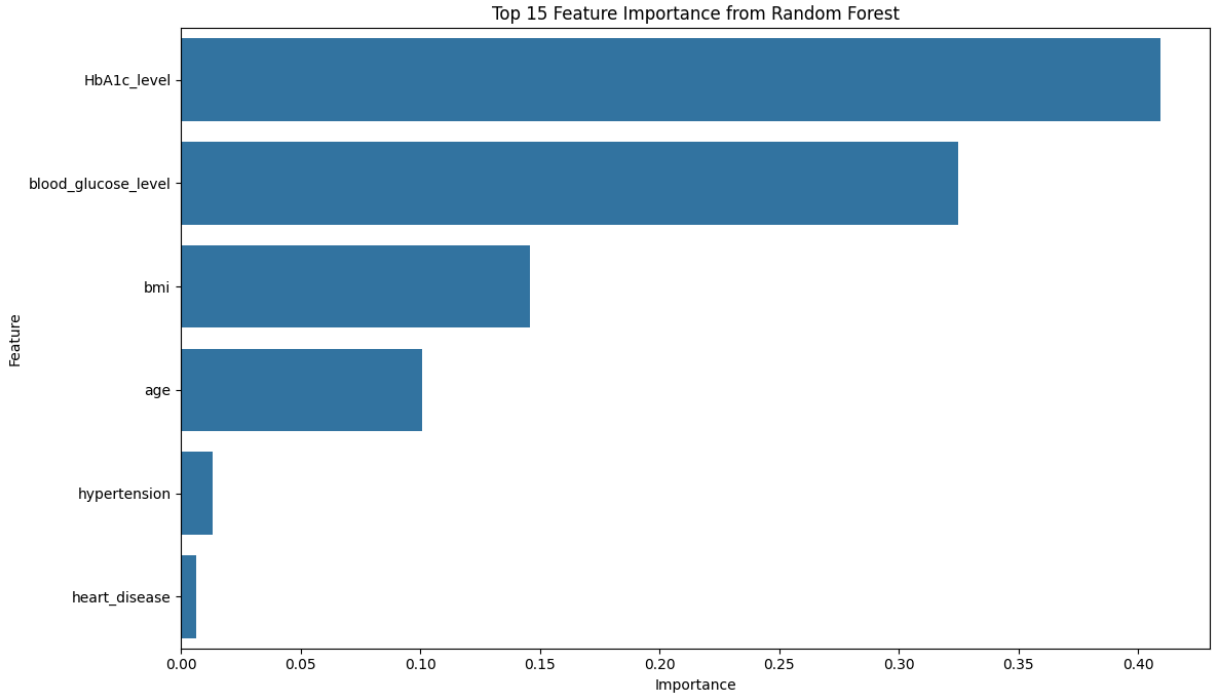




In [67]:
```
#16    Feature Importance (Random Forest)
print(f"Feature Importance (Random Forest)")

# Define all_feature_names for feature importance visualization
#all_feature_names = X.columns.tolist()

# Ensure the random_forest_model is trained and has feature_importances_ attribute
if hasattr(random_forest_model, 'feature_importances_'):
```

```
    importances = random_forest_model.feature_importances_

    # Create a DataFrame for better visualization
    feature_importance_df = pd.DataFrame({'feature': all_feature_names, 'importance
    feature_importance_df = feature_importance_df.sort_values(by='importance', asce

    print("Top 15 Features by Importance (Random Forest):")
    print(feature_importance_df.head(15))

    # Visualize feature importance
    plt.figure(figsize=(12, 7))
    sns.barplot(x='importance', y='feature', data=feature_importance_df.head(15))
    plt.title('Top 15 Feature Importance from Random Forest')
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.tight_layout()
    plt.show()
else:
    print("Random Forest model does not have 'feature_importances_' attribute. Ensu
```

```
Feature Importance (Random Forest)
Top 15 Features by Importance (Random Forest):
              feature  importance
4          HbA1c_level    0.409467
5  blood_glucose_level    0.324617
3                  bmi    0.145643
0                  age    0.100718
1         hypertension    0.013193
2        heart_disease    0.006361
```



In [68]:
```
#17 Hyperparameter Tuning (Random Forest)
print("Hyperparameter Tuning for Random Forest")
# Define the parameter grid for GridSearchCV
# GridSearchCV is a powerful tool that automates the process of creating different
# combinations of the hyperparameters listed below for the Random Forest model
```

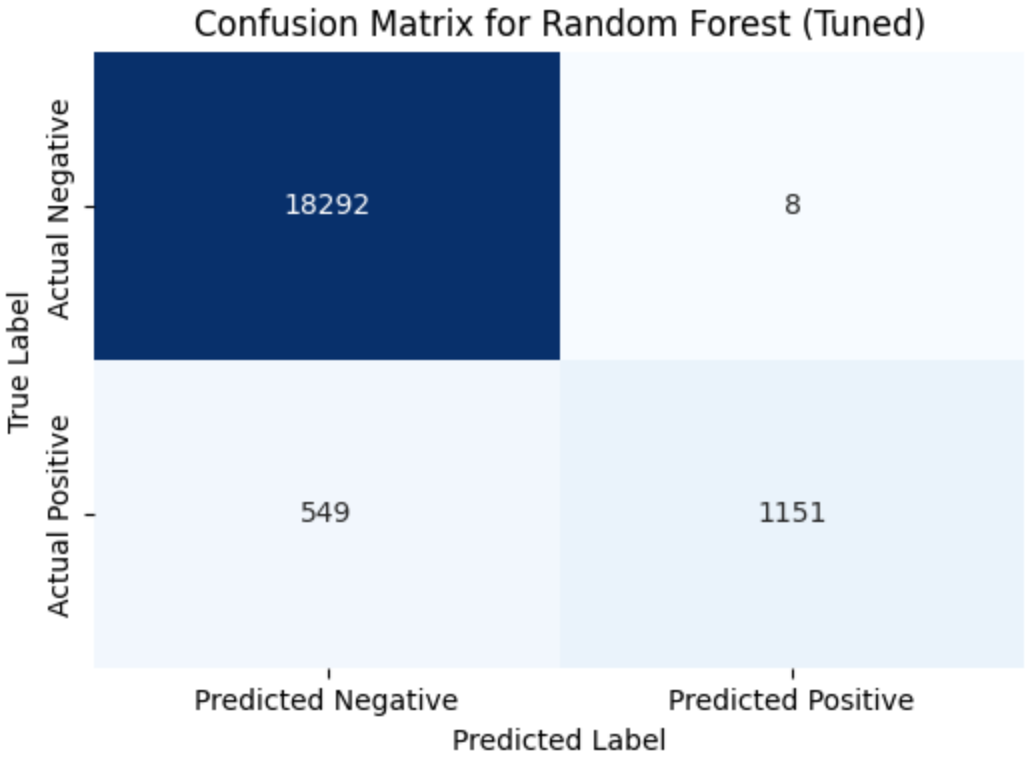
```
# It measures the performance using roc_auc_score and reports the best combinations
# hyperparameter settings
# In short, hyperparameter tuning is how we find the optimal "settings" for our mod
# That is automatically done by GridSearchCV
param_grid_rf = {
    'n_estimators': [100, 200, 300], # Number of trees in the forest [try with 100,
    'max_depth': [10, 20, None],     # Maximum depth of the tree (None means unlimi
    'min_samples_split': [2, 5],     # This hyperparameter sets the minimum number
    # before it is allowed to split into two new nodes.
    'min_samples_leaf': [1, 2]       # This hyperparameter sets the minimum number
}

# Initialize GridSearchCV
# We use the preprocessed data directly here since GridSearchCV will handle the fit
grid_search_rf = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                              param_grid=param_grid_rf,
                              cv=3, # 3-fold cross-validation
                              n_jobs=-1, # Use all available cores
                              scoring='roc_auc', # Optimize for ROC AUC
                              verbose=1)

grid_search_rf.fit(X_train_processed, y_train)
print(f"Best parameters for Random Forest: {grid_search_rf.best_params_}")
print(f"Best ROC AUC score for Random Forest (from cross-validation): {grid_search_

# Evaluate the best Random Forest model on the test set
tuned_rf_model = grid_search_rf.best_estimator_
y_pred = tuned_rf_model.predict(X_test_processed)

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)
print(f"Random Forest (Tuned) - Accuracy: {accuracy:.3f}, Precision: {precision:.3f
```

```
Hyperparameter Tuning for Random Forest
Fitting 3 folds for each of 36 candidates, totalling 108 fits
Best parameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 2, 'min_sam
ples_split': 2, 'n_estimators': 300}
Best ROC AUC score for Random Forest (from cross-validation): 0.9749
Random Forest (Tuned) - Accuracy: 0.972, Precision: 0.993, Recall: 0.677, F1-score:
0.805
```

In [69]:

```
#17.1
# The roc_auc_score is a metric used to evaluate the performance of a classificatio
# ROC -> Receiver Operating Characteristic, AUC -> Area Under the Curve
# AUC ranges from 0 to 1
# 1 -> represents a perfect model that can perfectly distinguish between the positi
# 0.5 -> indicates a model that performs no better than random guessing.
# below 0.5 -> suggests the model is performing worse than random guessing and like

confu_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(confu_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
```

```
plt.title(f'Confusion Matrix for Random Forest (Tuned)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



In [70]:
```
#18 Hyperparameter Tuning KNN
print("Hyperparameter Tuning for KNN....")

param_grid_knn = {'n_neighbors': [3, 5, 7, 9]}
grid_search_knn = GridSearchCV(knn_model, param_grid_knn, cv=3,  n_jobs=-1, scoring

grid_search_knn.fit(X_train_processed, y_train)
print(f"Best Hyperparameters for KNN: {grid_search_knn.best_params_}")
print(f"Best ROC AUC score for KNN (from cross-validation): {grid_search_knn.best_s

# Evaluate the best Random Forest model on the test set
tuned_knn_model = grid_search_knn.best_estimator_

y_pred = tuned_knn_model.predict(X_test_processed)
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)
print(f" KNN (Tuned) - Accuracy: {accuracy:.3f}, Precision: {precision:.3f}, Recall
```

```
Hyperparameter Tuning for KNN....
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best Hyperparameters for KNN: {'n_neighbors': 9}
Best ROC AUC score for KNN (from cross-validation): 0.9287
 KNN (Tuned) - Accuracy: 0.966, Precision: 0.936, Recall: 0.646, F1-score: 0.764
```

In [ ]: