**In this team assignment, we will use various machine learning models—including Logistic Regression, Decision Trees, Neural Networks, and Ensemble models—to predict diabetes in patients based on their medical history and demographic information. This can help healthcare professionals identify patients who might be at risk of developing diabetes and create personalized treatment plans for them. Additionally, studying this dataset can also help us explore the relationships between various medical and demographic factors and the likelihood of developing diabetes.**

# Appendix



# Data Dictionary

## Various Categorical Features

1. **Gender**: Gender refers to the biological sex of the patient (i.e. Female-->0 or Male-->1), which can have an impact on their susceptibility to diabetes.

2. **Hypertension**: Hypertension, or high blood pressure, indicates whether an individual has high blood pressure, with 0 representing no hypertension and 1 indicating the presence of hypertension.Having hypertension increases the risk of developing type 2 diabetes and vice versa.

3. **Heart Disease**: Heart disease is another medical condition that is associated with an increased risk of developing diabetes.Similarly, 1 indicates the presence of heart disease, while 0 represents no heart disease.

4. **Smoking History**: Smoking status is categorized into several types: current, ever, former, never, No Info, and not current. After recategory, the relationships equals to current -> 0, non-smoker -> 1, past_smoker -> 2

## Continuous Features

1. **Age**: The age range of patients. Usually, as individuals get older, their risk of developing diabetes increases.

2. **Body Mass Index (BMI)**: BMI is a measure of body fat based on a person's height and weight. It is commonly used as an indicator of overall weight status and can be helpful in predicting diabetes risk. Higher BMI is associated with a greater likelihood of developing type 2 diabetes.

3. **HbA1c Level**: HbA1c (glycated hemoglobin) is a measure of the average blood glucose level over the past 2-3 months. It provides information about long-term blood sugar control. Higher HbA1c levels indicate poorer glycemic control and are associated with an increased risk of developing diabetes and its complications.

4. **Blood Glucose Level**: Blood glucose level refers to the amount of glucose (sugar) present in the blood at a given time. Elevated blood glucose levels, particularly in the fasting state or after consuming carbohydrates, can indicate impaired glucose regulation and increase the risk of developing diabetes. Regular monitoring of blood glucose levels is important in the diagnosis and management of diabetes.

## Target Variable

**Diabetes**: The target variable indicating whether the individual has diabetes (1 represents"Yes", 0 represents "No").

# Data Preparation

### 1.Import Library

In [1]:
```
# Import Neccessary libraries
import numpy as np
import pandas as pd

# Import Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Import Modeling libraries
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
import warnings
warnings.filterwarnings("ignore")
```

### 2. Load the dataset

In [2]:
```
data = pd.read_csv("diabetes_prediction_dataset.csv")
print("Number of datapoints:", len(data))
data.head()
```

```
Number of datapoints: 100000
```

Out[2]:

| | gender | age | hypertension | heart_disease | smoking_history | bmi | HbA1c_le |
|---|---|---|---|---|---|---|---|
| **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 | |

# Data Cleaning & Feature Engineering

## 1.Data Cleaning

In [3]:
```
data.info()

duplicate_rows_data = data[data.duplicated()]
print("\n--->number of duplicate rows:", duplicate_rows_data.shape, "\n")

for column in data.columns:
    num_distinct_values = len(data[column].unique())
    print(f"{column}: {num_distinct_values} distinct values")
```

```
<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

--->number of duplicate rows: (3854, 9)

gender: 3 distinct values
age: 102 distinct values
hypertension: 2 distinct values
heart_disease: 2 distinct values
smoking_history: 6 distinct values
bmi: 4247 distinct values
HbA1c_level: 18 distinct values
blood_glucose_level: 18 distinct values
diabetes: 2 distinct values
```

**From the above output, we can make a conclusion and fix the problems:**

- a) There are not missing values in our dataset :)
- b) Our dataset contains 3854 duplicate records, and we need to delete them
- c) Gender suggests a spectrum beyond just two categories, yet we're maintaining it as a binary variable.

In [4]:
```
data = data.drop_duplicates()
data = data[data['gender'] != 'Other']
data.describe().style.format("{:.2f}")
```

Out[4]:

| | age | hypertension | heart_disease | bmi | HbA1c_level | blood_gl |
|---|---|---|---|---|---|---|
| **count** | 96128.00 | 96128.00 | 96128.00 | 96128.00 | 96128.00 | |
| **mean** | 41.80 | 0.08 | 0.04 | 27.32 | 5.53 | |
| **std** | 22.46 | 0.27 | 0.20 | 6.77 | 1.07 | |
| **min** | 0.08 | 0.00 | 0.00 | 10.01 | 3.50 | |
| **25%** | 24.00 | 0.00 | 0.00 | 23.40 | 4.80 | |
| **50%** | 43.00 | 0.00 | 0.00 | 27.32 | 5.80 | |
| **75%** | 59.00 | 0.00 | 0.00 | 29.86 | 6.20 | |
| **max** | 80.00 | 1.00 | 1.00 | 95.69 | 9.00 | |

### 2.Feature Engineering

In this step, we recategorized the feature **smoking_history** to increase its readability, and also convert **Gender, and smoking_history** into number variables so that the machine learning model can process them.

In [5]:
```
def recategorize_smoking(smoking_status):
    if smoking_status in ['never', 'No Info']:
        return 'non-smoker'
    elif smoking_status == 'current':
        return 'current'
    elif smoking_status in ['ever', 'former', 'not current']:
        return 'past_smoker'

# Apply the function to the 'smoking_history' column
data['smoking_history'] = data['smoking_history'].apply(recategorize_smoking

# Check the new value counts
print(data['smoking_history'].value_counts())
```

```
smoking_history
non-smoker     67276
past_smoker    19655
current         9197
Name: count, dtype: int64
```

In [6]:
```
def perform_label_encoding(df, column_name):
    # Initialize LabelEncoder
    le = LabelEncoder()

    # Fit and transform the column, and store the encoded values in the Data
    df[column_name] = le.fit_transform(df[column_name])

    # Print the mapping of original values to encoded values
    print(f"Mapping for '{column_name}':")
    for original_value, encoded_value in zip(le.classes_, range(len(le.class
        print(f"{original_value} -> {encoded_value}")

    return df

data = perform_label_encoding(data, 'gender')

data = perform_label_encoding(data, 'smoking_history')
```

```
Mapping for 'gender':
Female -> 0
Male -> 1
Mapping for 'smoking_history':
current -> 0
non-smoker -> 1
past_smoker -> 2
```

In [7]:
```
data.head()
```

Out[7]:

| | gender | age | hypertension | heart_disease | smoking_history | bmi | HbA1c_l |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 80.0 | 0 | 1 | 1 | 25.19 | |
| **1** | 0 | 54.0 | 0 | 0 | 1 | 27.32 | |
| **2** | 1 | 28.0 | 0 | 0 | 1 | 27.32 | |
| **3** | 0 | 36.0 | 0 | 0 | 0 | 23.45 | |
| **4** | 1 | 76.0 | 1 | 1 | 0 | 20.14 | |

# Exploratory Data Analysis

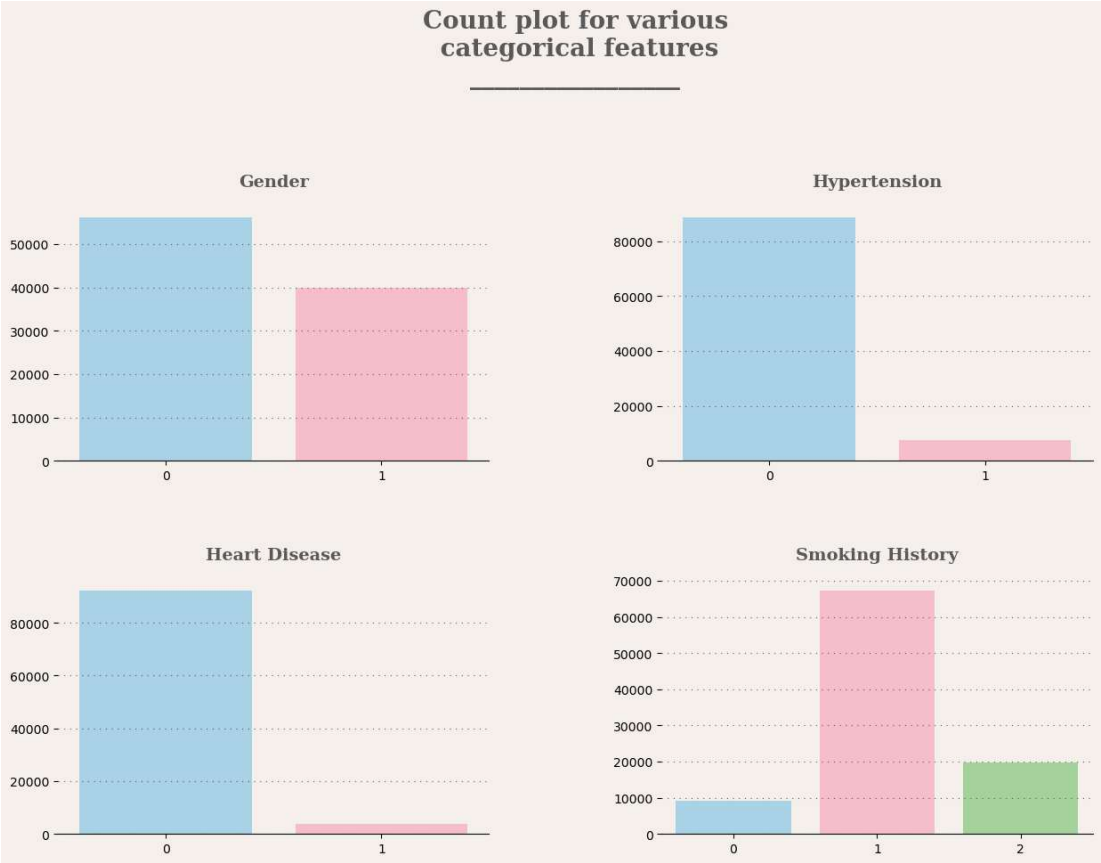
## 1. Feature distribution

In [8]:
```
cat_cols = ['gender','hypertension','heart_disease','smoking_history']
con_cols = ["age","bmi","HbA1c_level","blood_glucose_level"]
target_col = ["diabetes"]
print("The categorial cols are : ", cat_cols)
print("The continuous cols are : ", con_cols)
print("The target variable is :  ", target_col)
```

```
The categorial cols are :  ['gender', 'hypertension', 'heart_disease', 'smo
king_history']
The continuous cols are :  ['age', 'bmi', 'HbA1c_level', 'blood_glucose_lev
el']
The target variable is :   ['diabetes']
```

In [9]:
```
#Reference: Naman Manchanda, "Heart Attack EDA & Prediction - 90% Accuracy
color_palette = ["#A2D5F2", "#FFB7C5", "#A1DE93", "#F3E5AB", "#FFC3A0"]

fig = plt.figure(figsize=(15,12))
gs = fig.add_gridspec(3,2, height_ratios=[0.3, 1, 1])
gs.update(wspace=0.4, hspace=0.6)
background_color = "#F7F4EF"

fig.patch.set_facecolor(background_color)

ax_title = fig.add_subplot(gs[0,:])
ax1 = fig.add_subplot(gs[1,0])
ax2 = fig.add_subplot(gs[1,1])
ax3 = fig.add_subplot(gs[2,0])
ax4 = fig.add_subplot(gs[2,1])

for ax in [ax_title, ax1, ax2, ax3, ax4]:
    ax.set_facecolor(background_color)

ax_title.spines["bottom"].set_visible(False)
ax_title.spines["left"].set_visible(False)
ax_title.spines["top"].set_visible(False)
ax_title.spines["right"].set_visible(False)
ax_title.tick_params(left=False, bottom=False)
ax_title.set_xticklabels([])
ax_title.set_yticklabels([])
ax_title.text(0.5, 0.5,
```

```
             'Count plot for various\n categorical features\n_____________
             horizontalalignment='center',
             verticalalignment='center',
             fontsize=20, fontweight='bold',
             fontfamily='serif',
             color="#5a5a5a")

sns.countplot(ax=ax1, data=data, x='gender', palette=color_palette)
ax1.set_xlabel("")
ax1.set_ylabel("")
ax1.set_title("Gender", fontsize=14, fontweight='bold', fontfamily='serif'
ax1.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashes=(1,5))

sns.countplot(ax=ax2, data=data, x='hypertension', palette=color_palette)
ax2.set_xlabel("")
ax2.set_ylabel("")
ax2.set_title("Hypertension", fontsize=14, fontweight='bold', fontfamily='
ax2.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashes=(1,5))

sns.countplot(ax=ax3, data=data, x='heart_disease', palette=color_palette)
ax3.set_xlabel("")
ax3.set_ylabel("")
ax3.set_title("Heart Disease", fontsize=14, fontweight='bold', fontfamily=
ax3.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashes=(1,5))

sns.countplot(ax=ax4, data=data, x='smoking_history', palette=color_palett
ax4.set_xlabel("")
ax4.set_ylabel("")
ax4.set_title("Smoking History", fontsize=14, fontweight='bold', fontfamil
ax4.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashes=(1,5))

for ax in [ax1, ax2, ax3, ax4]:
    for s in ["top", "right", "left"]:
        ax.spines[s].set_visible(False)

plt.show()
```

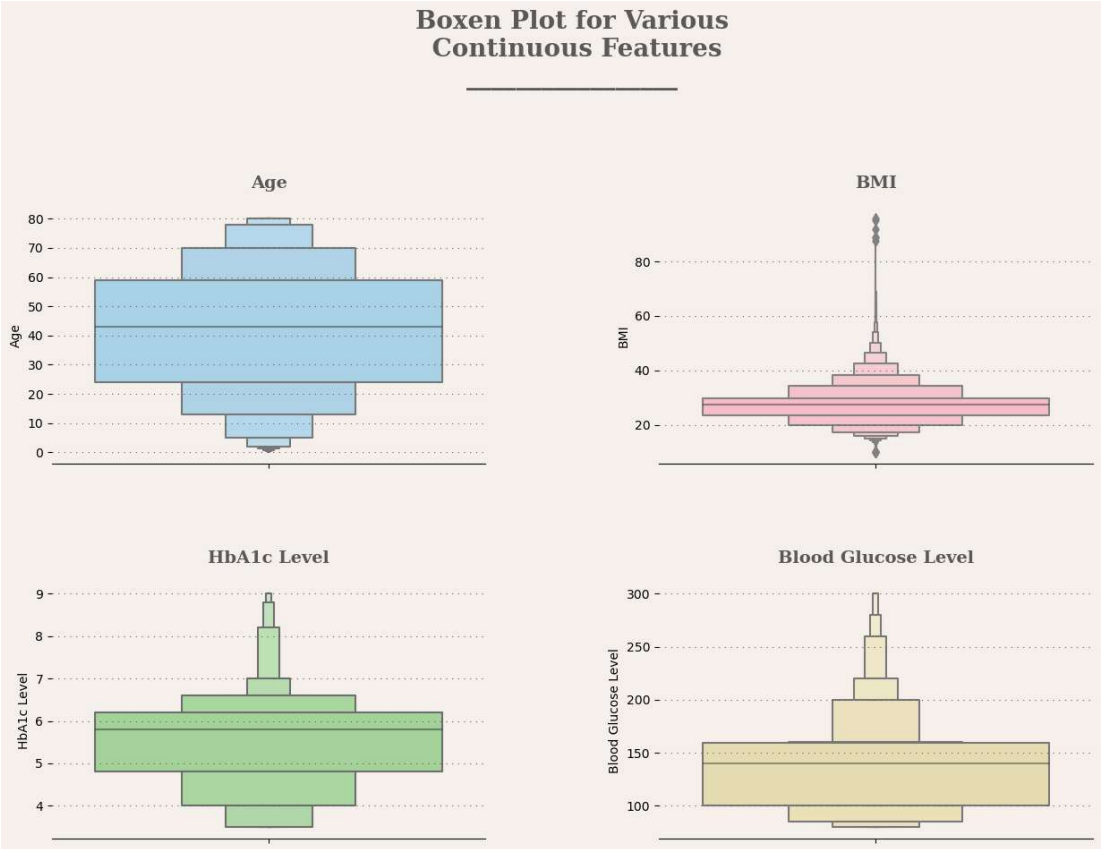



**From the above output, we can conclude and note that:**

- a) Gender: The proportion of males (encoded as 1) is slightly lower than that of females (encoded as 0), indicating a higher number of females in the sample.
- b) Hypertension: Most individuals do not have hypertension (encoded as 0), with a smaller number having hypertension (encoded as 1), showing a relatively low hypertension rate.
- c) Heart Disease: The vast majority do not have heart disease (encoded as 0), with only a small portion affected (encoded as 1), indicating a low prevalence of heart disease.
- d) Smoking History: Non-smokers (encoded as 1) make up the largest group, followed by past smokers (encoded as 2), and current smokers (encoded as 0) are the fewest, suggesting most individuals have no smoking history.

In [10]:
```
#Reference: Naman Manchanda, "Heart Attack EDA & Prediction - 90% Accuracy
color_palette = ["#A2D5F2", "#FFB7C5", "#A1DE93", "#F3E5AB", "#FFC3A0"]

fig = plt.figure(figsize=(15,12))
gs = fig.add_gridspec(3,2, height_ratios=[0.3, 1, 1])
gs.update(wspace=0.4, hspace=0.6)
background_color = "#F7F4EF"

fig.patch.set_facecolor(background_color)

ax_title = fig.add_subplot(gs[0,:])
ax1 = fig.add_subplot(gs[1,0])
ax2 = fig.add_subplot(gs[1,1])
```

```
ax3 = fig.add_subplot(gs[2,0])
ax4 = fig.add_subplot(gs[2,1])

for ax in [ax_title, ax1, ax2, ax3, ax4]:
    ax.set_facecolor(background_color)

ax_title.spines["bottom"].set_visible(False)
ax_title.spines["left"].set_visible(False)
ax_title.spines["top"].set_visible(False)
ax_title.spines["right"].set_visible(False)
ax_title.tick_params(left=False, bottom=False)
ax_title.set_xticklabels([])
ax_title.set_yticklabels([])
ax_title.text(0.5, 0.5,
              'Boxen Plot for Various\n Continuous Features\n_____________
              horizontalalignment='center',
              verticalalignment='center',
              fontsize=20, fontweight='bold',
              fontfamily='serif',
              color="#5a5a5a")

sns.boxenplot(ax=ax1, data=data, y='age', palette=[color_palette[0]])
ax1.set_xlabel("")
ax1.set_ylabel("Age")
ax1.set_title("Age", fontsize=14, fontweight='bold', fontfamily='serif', c
ax1.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashes=(1,5))

sns.boxenplot(ax=ax2, data=data, y='bmi', palette=[color_palette[1]])
ax2.set_xlabel("")
ax2.set_ylabel("BMI")
ax2.set_title("BMI", fontsize=14, fontweight='bold', fontfamily='serif', c
ax2.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashes=(1,5))

sns.boxenplot(ax=ax3, data=data, y='HbA1c_level', palette=[color_palette[2
ax3.set_xlabel("")
ax3.set_ylabel("HbA1c Level")
ax3.set_title("HbA1c Level", fontsize=14, fontweight='bold', fontfamily='s
ax3.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashes=(1,5))

sns.boxenplot(ax=ax4, data=data, y='blood_glucose_level', palette=[color_p
ax4.set_xlabel("")
ax4.set_ylabel("Blood Glucose Level")
ax4.set_title("Blood Glucose Level", fontsize=14, fontweight='bold', fontf
ax4.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashes=(1,5))

for ax in [ax1, ax2, ax3, ax4]:
    for s in ["top", "right", "left"]:
        ax.spines[s].set_visible(False)

plt.show()
```

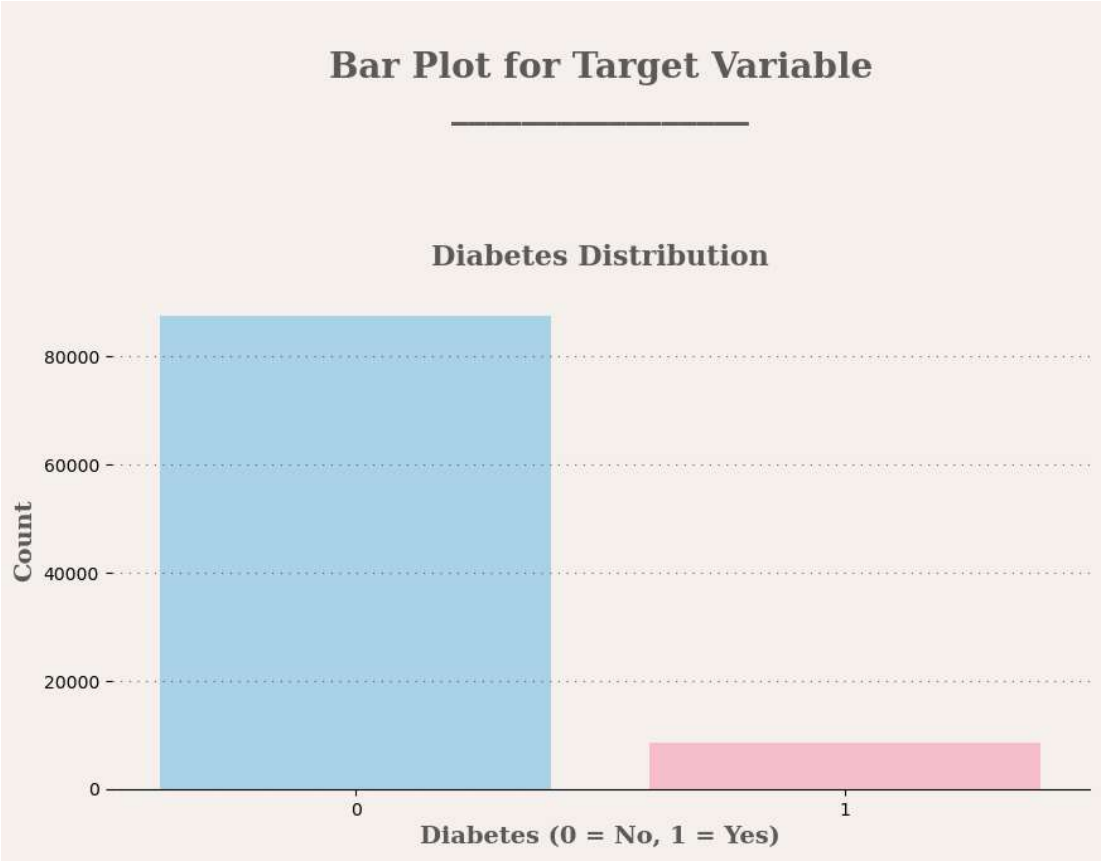



**From the above output, we can conclude and note that:**

- a) Age: The age distribution is concentrated between 30 and 60, with a few younger and older individuals, showing a broad distribution.
- b) BMI: Most data points are between 20 and 40, with some higher outliers, indicating that some individuals have a high BMI.
- c) HbA1c Level: HbA1c levels are mainly between 5 and 6, slightly elevated, suggesting higher glycated hemoglobin levels in some samples.
- d) Blood Glucose Level: Blood glucose levels are concentrated between 100 and 150, with a few outliers above 250, indicating potential high blood sugar in some individuals.

In [11]:
```
#Reference: Naman Manchanda, "Heart Attack EDA & Prediction - 90% Accuracy
color_palette = ["#A2D5F2", "#FFB7C5"]

fig = plt.figure(figsize=(10,8))
gs = fig.add_gridspec(2,1, height_ratios=[0.3, 1])
gs.update(wspace=0.4, hspace=0.4)
background_color = "#F7F4EF"

fig.patch.set_facecolor(background_color)

ax_title = fig.add_subplot(gs[0, :])
ax_diabetes = fig.add_subplot(gs[1, :])

for ax in [ax_title, ax_diabetes]:
    ax.set_facecolor(background_color)

ax_title.spines["bottom"].set_visible(False)
```

```
ax_title.spines["left"].set_visible(False)
ax_title.spines["top"].set_visible(False)
ax_title.spines["right"].set_visible(False)
ax_title.tick_params(left=False, bottom=False)
ax_title.set_xticklabels([])
ax_title.set_yticklabels([])
ax_title.text(0.5, 0.5,
              'Bar Plot for Target Variable\n_________________',
              horizontalalignment='center',
              verticalalignment='center',
              fontsize=20, fontweight='bold',
              fontfamily='serif',
              color="#5a5a5a")

sns.countplot(ax=ax_diabetes, data=data, x='diabetes', palette=color_palet
ax_diabetes.set_xlabel("Diabetes (0 = No, 1 = Yes)", fontsize=14, fontweig
ax_diabetes.set_ylabel("Count", fontsize=14, fontweight='bold', fontfamily
ax_diabetes.set_title("Diabetes Distribution", fontsize=16, fontweight='bo
ax_diabetes.grid(color='#5a5a5a', linestyle=':', axis='y', zorder=0, dashe

for s in ["top", "right", "left"]:
    ax_diabetes.spines[s].set_visible(False)

plt.show()
```

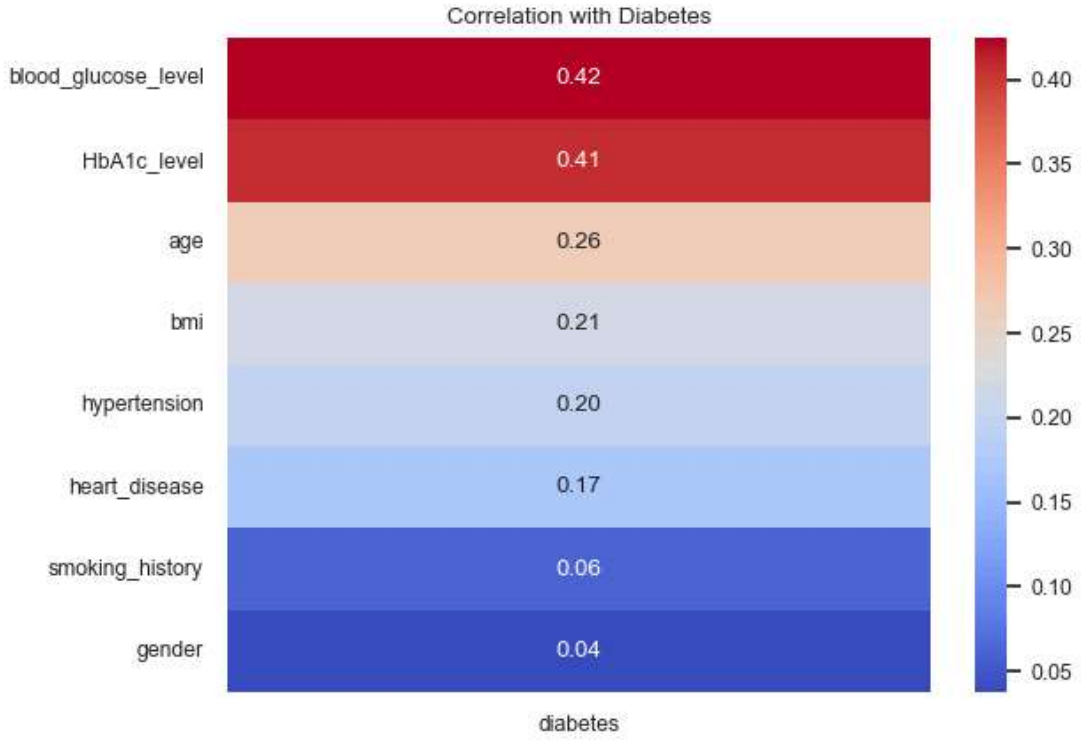



Note: This chart shows an uneven distribution of the target variable, which may require imbalanced data handling in the subsequent regression and neural network models.

### 2. Correlation Matrix

In [12]:
```
# reference: Tumpan Jawat, "Diabetes EDA & Random Forest HP," Kaggle, https
correlation_matrix = data[['age', 'bmi', 'HbA1c_level', 'blood_glucose_lev
```

```
                            'gender', 'hypertension', 'heart_disease',
                            'smoking_history', 'diabetes']].corr()

target_corr = correlation_matrix['diabetes'].drop('diabetes')
target_corr_sorted = target_corr.sort_values(ascending=False)

sns.set(font_scale=0.8)
sns.set_style("white")
sns.set_palette("PuBuGn_d")
sns.heatmap(target_corr_sorted.to_frame(), cmap="coolwarm", annot=True, fm
plt.title('Correlation with Diabetes')
plt.show()
```




**From the correlation matrix, we can conclude that:**

- Blood_glucose_level (0.42) and HbA1c_level (0.41) have the strongest positive correlation with diabetes, followed by age (0.26) and bmi (0.21). Hypertension (0.20) and heart_disease (0.17) show slightly weaker correlations, while smoking_history (0.06) and gender (0.04) have the lowest correlations with diabetes. This indicates that blood glucose and HbA1c levels are key indicators for predicting diabetes.

## Model1: Logistic Regression with Regularisation

In [13]:
```
# Loading features to variable X
# Loading target column to variable Y
X = data.drop('diabetes', axis=1)
#X = X.drop('gender', axis=1)
Y = data['diabetes']
# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
```

```
# Train the Logistic Regression model
from sklearn.linear_model import LogisticRegression
Model = LogisticRegression()

Model.fit(X_train, Y_train)

Y_test_pred = Model.predict(X_test)

# Compare the predicted values with the actuals
results = pd.DataFrame({'Actual': Y_test, 'Predicted': Y_test_pred})
results

# compute model accuracy
from sklearn.metrics import accuracy_score
print("Model accuracy on test data:", accuracy_score(Y_test, Y_test_pred)

# Model accuracy on training data
Y_train_pred = Model.predict(X_train)
print("Model accuracy on training data:", accuracy_score(Y_train, Y_train
# Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_test_pred)

from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_test_pred))

# Calculate metrics values individually
# Assigning Variables for convenience
TN = cm[0][0]
FP = cm[0][1]
FN = cm[1][0]
TP = cm[1][1]

recall = TP / (TP + FN)
print("Recall=", recall)

precision = TP / (TP + FP)
print("Precision=", precision)

specificity = TN / (TN + FP)
print("Specificity=", specificity)

accuracy = (TP + TN) / (TP + TN + FP + FN)
print("Accuracy =", accuracy)
```

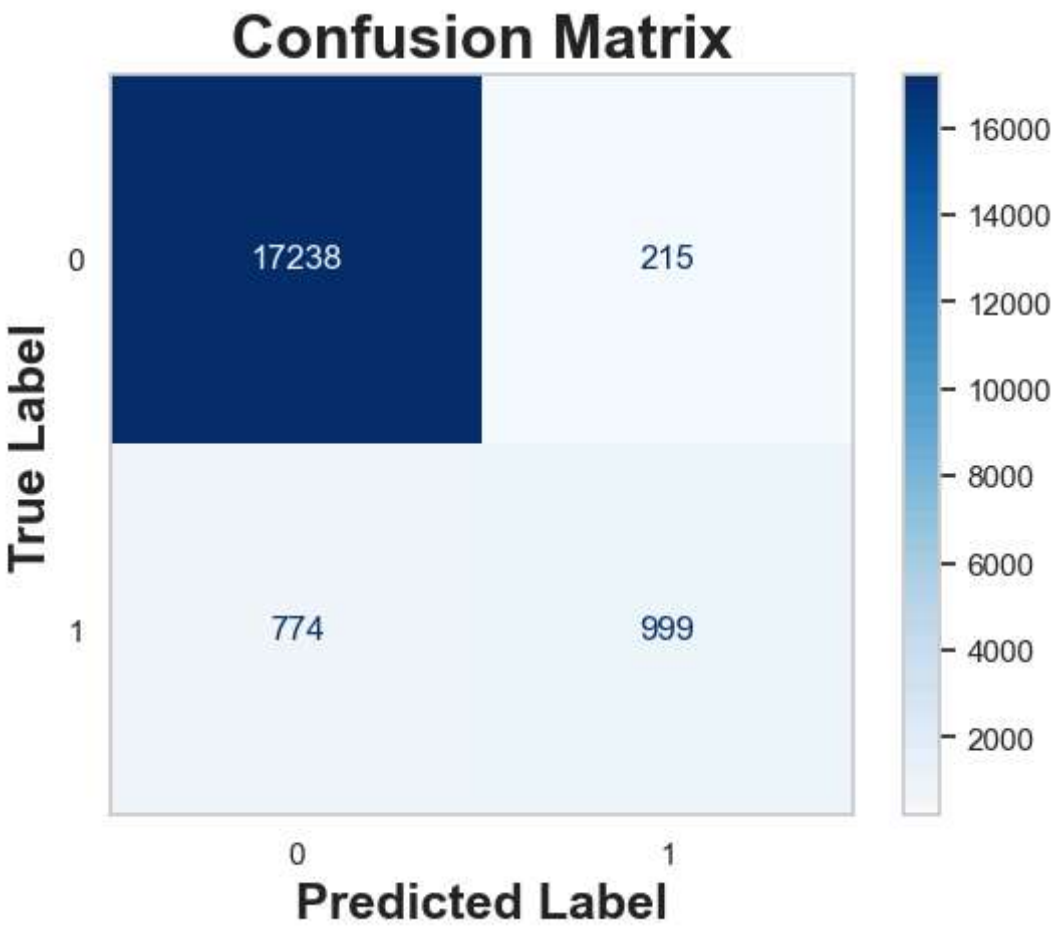
```
Model accuracy on test data: 0.9485592426921876
Model accuracy on training data: 0.9535122623598866
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     17453
           1       0.82      0.56      0.67      1773

    accuracy                           0.95     19226
   macro avg       0.89      0.78      0.82     19226
weighted avg       0.94      0.95      0.94     19226

Recall= 0.5634517766497462
Precision= 0.8228995057660626
Specificity= 0.9876812009396665
Accuracy = 0.9485592426921876
```

In [14]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(Y_test,Y_test_pred)

print(cm)
```

```
[[17238   215]
 [  774   999]]
```

In [15]:
```
from sklearn.metrics import ConfusionMatrixDisplay
sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues', colorbar=True)
plt.title("Confusion Matrix", fontsize=22, fontweight='bold')
plt.xlabel("Predicted Label", fontsize=18, fontweight='bold')
plt.ylabel("True Label", fontsize=18, fontweight='bold')
plt.grid(False)
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```




**Observations:**

The model achieves an accuracy of 95.1% on the test set and 95.55% on the training set, indicating stable performance without significant overfitting. According to the classification report, the model performs well in predicting class 0 (negative), with a recall of 99% and an F1-score of 97%. However, it performs relatively poorly in predicting class 1 (positive), with a recall of only 61% and an F1-score of 70%, suggesting weaker detection ability for the positive class.

In [16]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report, confus

# Load and preprocess data
data = pd.read_csv("diabetes_prediction_dataset.csv")
data = data.drop_duplicates()
data = data[data['gender'] != 'Other']

# Recategorize smoking history
def recategorize_smoking(smoking_status):
    if smoking_status in ['never', 'No Info']:
```

```
        return 'non-smoker'
    elif smoking_status == 'current':
        return 'current'
    elif smoking_status in ['ever', 'former', 'not current']:
        return 'past_smoker'

data['smoking_history'] = data['smoking_history'].apply(recategorize_smok

# Encode categorical variables
def perform_label_encoding(df, column_name):
    le = LabelEncoder()
    df[column_name] = le.fit_transform(df[column_name])
    return df

data = perform_label_encoding(data, 'gender')
data = perform_label_encoding(data, 'smoking_history')
# loading features to variable X
# Loading target column to variable Y
X = data.drop('diabetes', axis=1)
#X = X.drop('gender', axis=1)
Y = data['diabetes']
# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,

# Train the Logistic Regression model with L2 regularization
from sklearn.linear_model import LogisticRegression

# Set the regularization strength (smaller C means stronger regularizatio
Model = LogisticRegression(penalty='l2', C=10, solver='liblinear')  # Adj

Model.fit(X_train, Y_train)

Y_test_pred = Model.predict(X_test)

# Compare the predicted values with the actuals
results = pd.DataFrame({'Actual': Y_test, 'Predicted': Y_test_pred})
print(results)

# compute model accuracy
from sklearn.metrics import accuracy_score
print("Model accuracy on test data:", accuracy_score(Y_test, Y_test_pred)

# Model accuracy on training data
Y_train_pred = Model.predict(X_train)
print("Model accuracy on training data:", accuracy_score(Y_train, Y_train

# Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_test_pred)

from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_test_pred))

# Calculate metrics values individually
# Assigning Variables for convenience
TN = cm[0][0]
FP = cm[0][1]
FN = cm[1][0]
TP = cm[1][1]
```

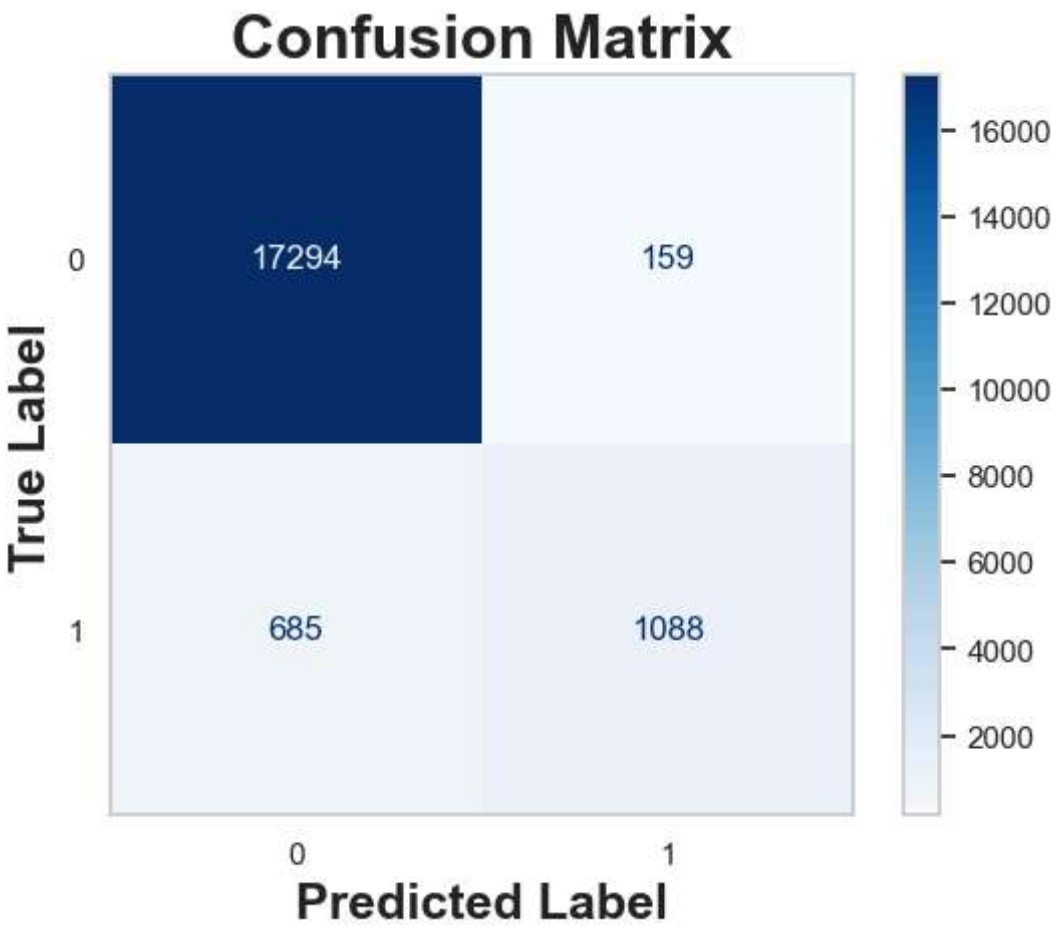
```
recall = TP / (TP + FN)
print("Recall=", recall)

precision = TP / (TP + FP)
print("Precision=", precision)

specificity = TN / (TN + FP)
print("Specificity=", specificity)

accuracy = (TP + TN) / (TP + TN + FP + FN)
print("Accuracy =", accuracy)
```

```
       Actual  Predicted
71645       0          0
86909       0          0
43323       0          0
42497       0          0
6794        0          0
...       ...        ...
38075       0          0
64619       0          0
96526       0          0
99462       0          0
94435       0          0

[19226 rows x 2 columns]
Model accuracy on test data: 0.9561011130760428
Model accuracy on training data: 0.9593508621362253
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     17453
           1       0.87      0.61      0.72      1773

    accuracy                           0.96     19226
   macro avg       0.92      0.80      0.85     19226
weighted avg       0.95      0.96      0.95     19226

Recall= 0.613649182177101
Precision= 0.8724939855653568
Specificity= 0.9908898183693348
Accuracy = 0.9561011130760428
```

In [17]:

```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues', colorbar=True)
plt.title("Confusion Matrix", fontsize=22, fontweight='bold')
plt.xlabel("Predicted Label", fontsize=18, fontweight='bold')
plt.ylabel("True Label", fontsize=18, fontweight='bold')
plt.grid(False)
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```




**Observations:**

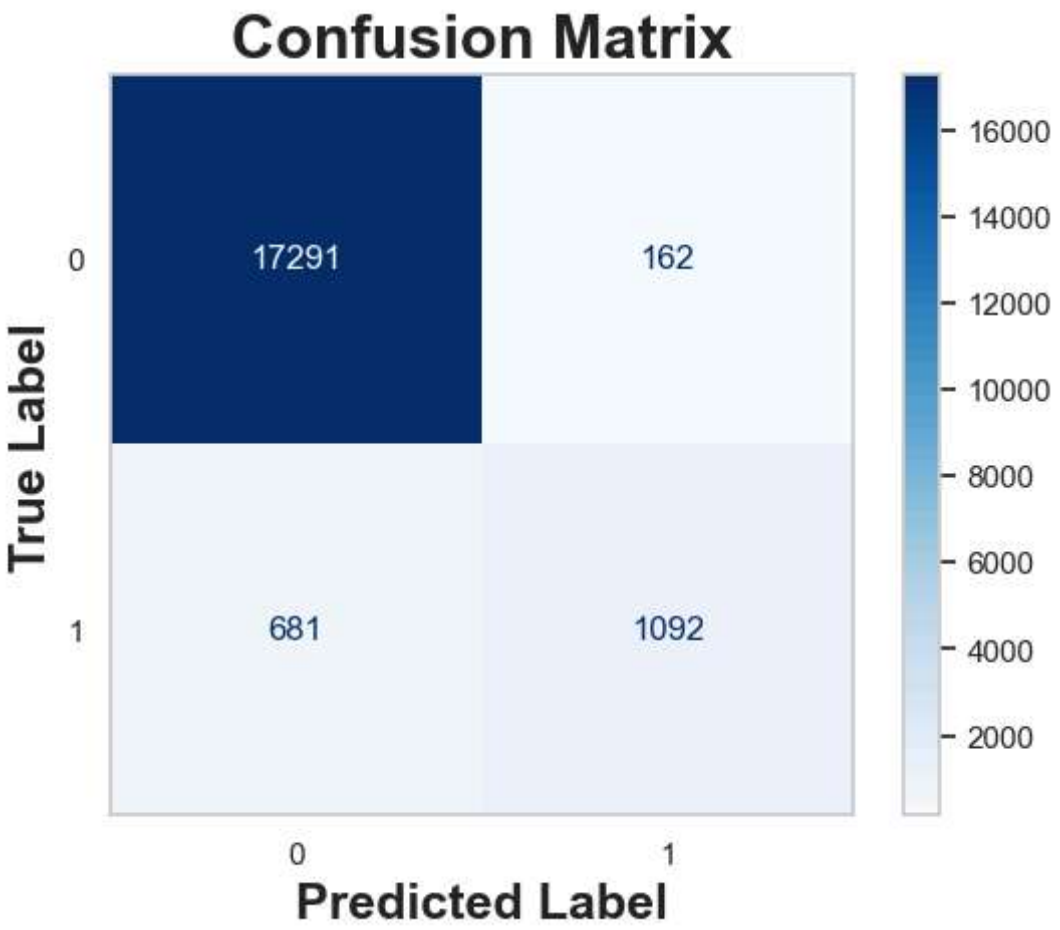
After applying L2 regularization, the logistic regression model showed improved performance. The test accuracy increased to 95.61%, closely aligning with the training accuracy of 95.94%, indicating minimal overfitting. The classification report reveals that precision for class 1 (positive) improved to 87%, although recall remained at 61%, resulting in an F1-score of 72%. The macro average F1-score rose to 0.85, indicating better class balance. In the confusion matrix, false positives decreased to 159, which enhanced precision for the positive class, while false negatives remained high at 685. Specificity also increased to 99.08%, reflecting the model's strong ability to identify true negatives. The model used L2 regularization with a penalty parameter of C=10, which helped to reduce overfitting and improve precision.

In [18]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confus

# Load and preprocess data
data = pd.read_csv("diabetes_prediction_dataset.csv")
data = data.drop_duplicates()
```

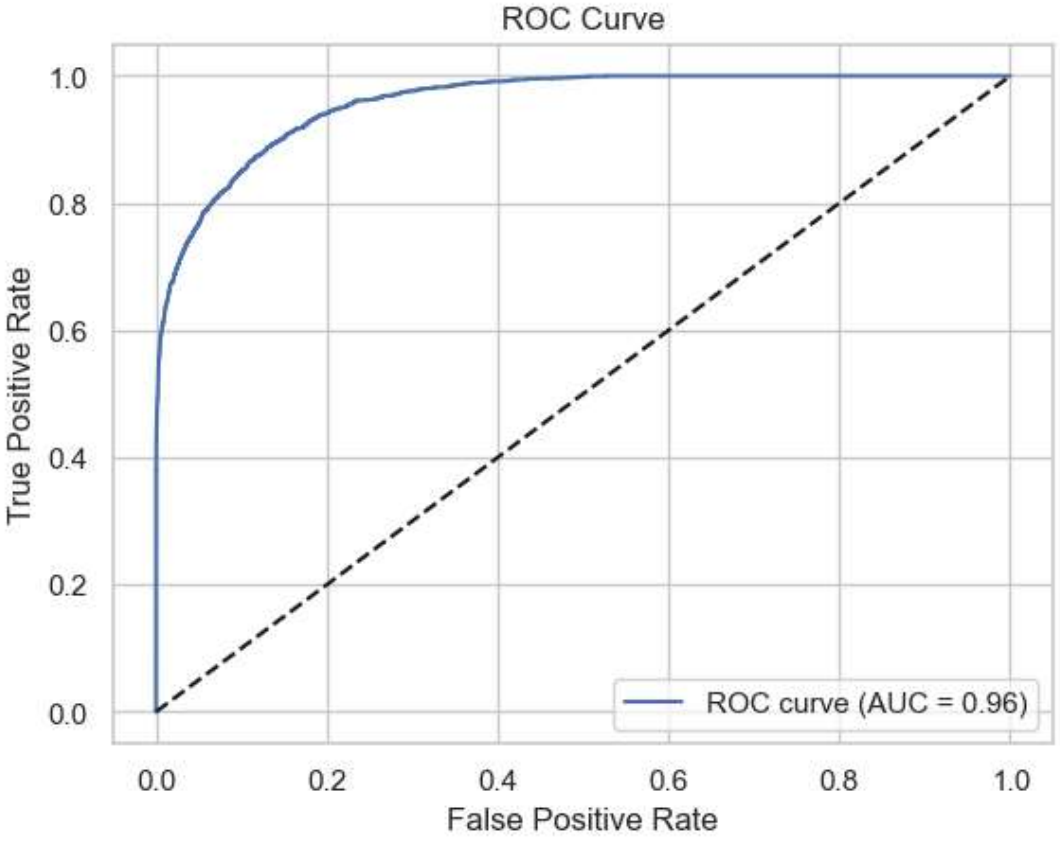
```
data = data[data['gender'] != 'Other']

# Recategorize smoking history
def recategorize_smoking(smoking_status):
    if smoking_status in ['never', 'No Info']:
        return 'non-smoker'
    elif smoking_status == 'current':
        return 'current'
    elif smoking_status in ['ever', 'former', 'not current']:
        return 'past_smoker'

data['smoking_history'] = data['smoking_history'].apply(recategorize_smok

# Encode categorical variables
def perform_label_encoding(df, column_name):
    le = LabelEncoder()
    df[column_name] = le.fit_transform(df[column_name])
    return df

data = perform_label_encoding(data, 'gender')
data = perform_label_encoding(data, 'smoking_history')

# Define features and target variable
X = data.drop('diabetes', axis=1)
Y = data['diabetes']

# Standardize numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,

# Define the Logistic Regression model and the parameter grid
model = LogisticRegression(penalty='l2', solver='liblinear')
param_grid = {'C': [0.01, 0.1, 1, 10, 100]}  # Range of regularization st

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, Y_train)

# Print the best parameters and best score
print("Best regularization parameter (C):", grid_search.best_params_['C']
print("Best cross-validation accuracy:", grid_search.best_score_)

# Train the final model using the best C value
best_model = grid_search.best_estimator_
Y_test_pred = best_model.predict(X_test)
Y_train_pred = best_model.predict(X_train)

# Compute accuracy
print("Model accuracy on test data:", accuracy_score(Y_test, Y_test_pred)
print("Model accuracy on training data:", accuracy_score(Y_train, Y_train

# Classification report
print(classification_report(Y_test, Y_test_pred))

# Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_test_pred)
```

```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues', colorbar=True)
plt.title("Confusion Matrix", fontsize=22, fontweight='bold')
plt.xlabel("Predicted Label", fontsize=18, fontweight='bold')
plt.ylabel("True Label", fontsize=18, fontweight='bold')
plt.grid(False)
plt.show()

# Individual metric calculations
TN, FP, FN, TP = cm.ravel()
recall = TP / (TP + FN)
precision = TP / (TP + FP)
specificity = TN / (TN + FP)
accuracy = (TP + TN) / (TP + TN + FP + FN)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"Specificity: {specificity}")
print(f"Accuracy: {accuracy}")

# ROC curve and AUC score
Y_test_pred_prob = best_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(Y_test, Y_test_pred_prob)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random guessing
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
plt.show()
```

```
Best regularization parameter (C): 0.1
Best cross-validation accuracy: 0.9592598166941142
Model accuracy on test data: 0.9561531259752418
Model accuracy on training data: 0.9593638656992016
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     17453
           1       0.87      0.62      0.72      1773

    accuracy                           0.96     19226
   macro avg       0.92      0.80      0.85     19226
weighted avg       0.95      0.96      0.95     19226

<Figure size 1000x800 with 0 Axes>
```




```
Recall: 0.6159052453468697
Precision: 0.8708133971291866
Specificity: 0.9907179281498882
Accuracy: 0.9561531259752418
```




**Observations:**

After applying grid search to optimize the regularization parameter C in the logistic regression model, the best value was found to be C=0.1, with a cross-validation accuracy of approximately 95.93%. The model achieved 95.61% accuracy on the test set and 95.94% on the training set, indicating minimal overfitting. The classification report shows that the model performs well in predicting class 0 (negative) with a precision of 96% and recall of 99%, resulting in an F1-score of 98%. For class 1 (positive), precision improved to 87% with a recall of 62%, leading to an F1-score of 72%. The confusion matrix shows a reduction in both false positives (162) and false negatives (681), contributing to a better balance between precision and recall. Additionally, the model has a high specificity of 99.07% and an AUC of 0.96, indicating excellent discrimination between the classes.

# Model2: Decision Tree

In [19]:
```
# Define features (X) and target variable (y)
X = data.drop('diabetes', axis=1)
y = data['diabetes']
```

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

In [21]:
```
from sklearn.tree import DecisionTreeClassifier
# Initialize the Decision Tree Classifier
dtc = DecisionTreeClassifier(random_state=42)

# Train the model
dtc.fit(X_train, y_train)
```

Out[21]:

DecisionTreeClassifier

DecisionTreeClassifier(random_state=42)

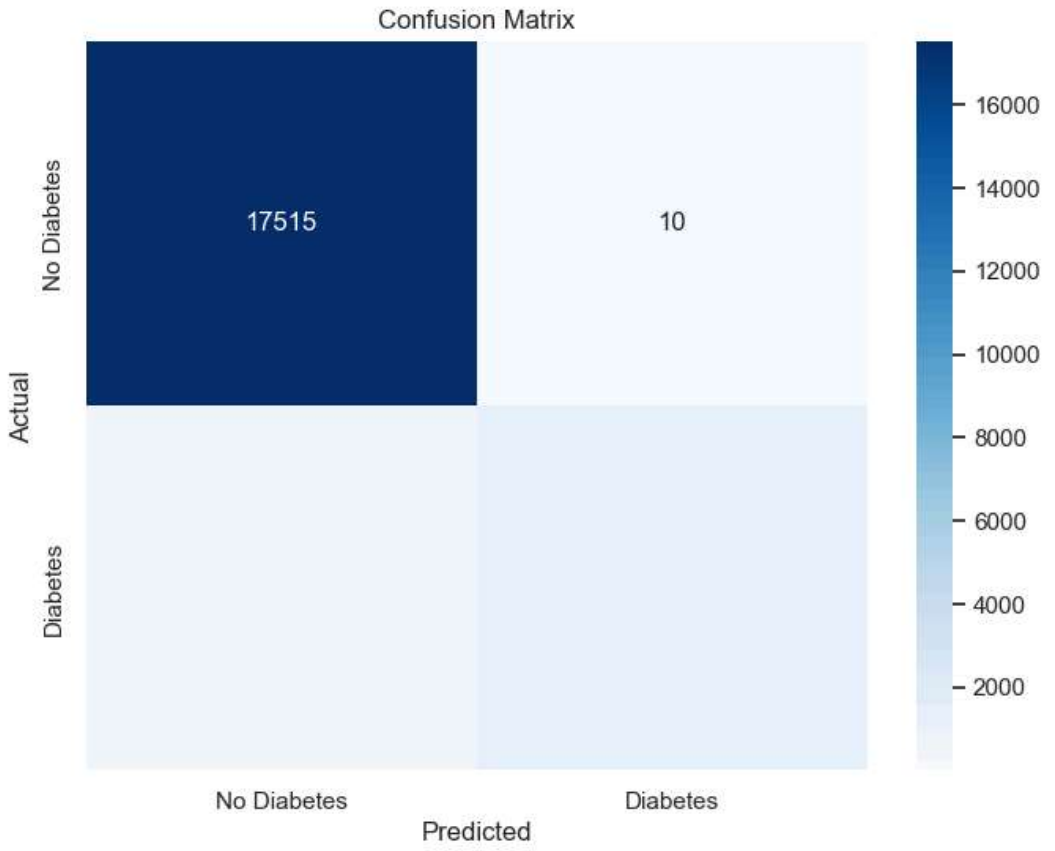
In [22]:
```
# Define the parameter grid
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dtc, param_grid=param_grid, cv=5, r

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Best parameters from GridSearchCV
print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best Parameters: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
leaf': 4, 'min_samples_split': 2}
```

In [23]:
```python
# Data manipulation and analysis
import numpy as np
import pandas as pd

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Machine learning utilities
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confu

# Suppress warnings
import warnings
warnings.filterwarnings("ignore")
```

In [24]:
```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Best estimator
best_dtc = grid_search.best_estimator_

# Predictions on the test set
y_pred = best_dtc.predict(X_test)

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.97
Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     17525
           1       0.99      0.66      0.79      1701

    accuracy                           0.97     19226
   macro avg       0.98      0.83      0.89     19226
weighted avg       0.97      0.97      0.97     19226
```



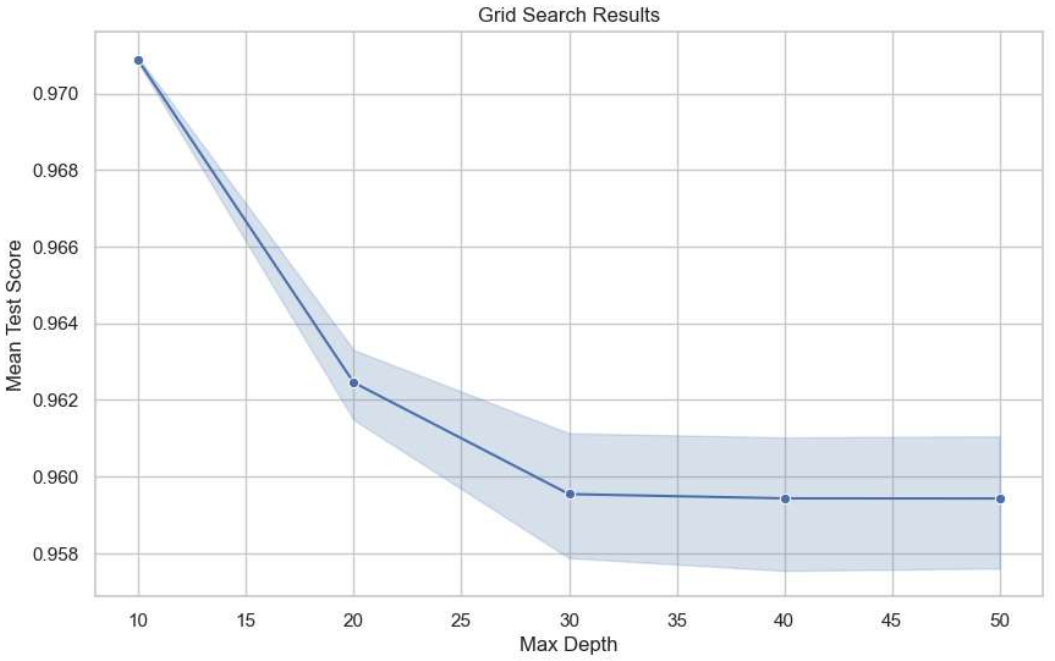

In [25]:
```
import pandas as pd

# Convert cv_results_ to a DataFrame
results_df = pd.DataFrame(grid_search.cv_results_)

# Display the first few rows
print(results_df.head())
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.202858      0.017050         0.005585        0.000797
1       0.212033      0.027858         0.006184        0.003420
2       0.199266      0.018198         0.005785        0.001164
3       0.210835      0.019108         0.004788        0.000399
4       0.219613      0.018801         0.005186        0.000747

  param_criterion param_max_depth  param_min_samples_leaf  \
0            gini            None                       1
1            gini            None                       1
2            gini            None                       1
3            gini            None                       2
4            gini            None                       2

   param_min_samples_split                                              p
arams  \
0                        2  {'criterion': 'gini', 'max_depth': None, 'mi
n_...
1                        5  {'criterion': 'gini', 'max_depth': None, 'mi
n_...
2                       10  {'criterion': 'gini', 'max_depth': None, 'mi
n_...
3                        2  {'criterion': 'gini', 'max_depth': None, 'mi
n_...
4                        5  {'criterion': 'gini', 'max_depth': None, 'mi
n_...

   split0_test_score  split1_test_score  split2_test_score  split3_test_
score  \
0           0.951759           0.950588           0.949220           0.9
49285
1           0.957285           0.955595           0.955527           0.9
54486
2           0.960796           0.959560           0.959233           0.9
57412
3           0.959951           0.960861           0.959038           0.9
57542
4           0.961381           0.961316           0.958388           0.9
59038

   split4_test_score  mean_test_score  std_test_score  rank_test_score
0           0.950065         0.950183        0.000938              106
1           0.955917         0.955762        0.000900               95
2           0.959298         0.959260        0.001083               77
3           0.958973         0.959273        0.001106               74
4           0.959623         0.959949        0.001208               67
```

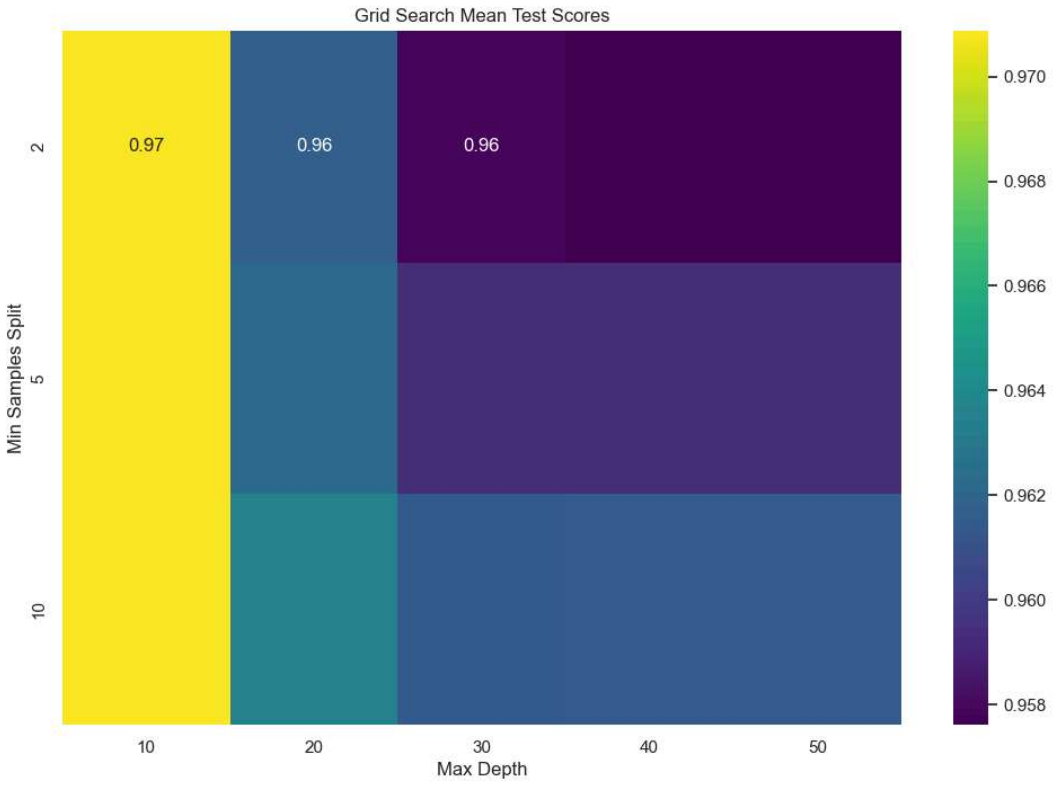
In [26]:

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Extract the mean test scores and the corresponding hyperparameter valu
mean_test_scores = results_df['mean_test_score']
param_values = results_df['param_max_depth']  # Replace 'param_max_depth

# Create a DataFrame for plotting
plot_df = pd.DataFrame({
    'param_value': param_values,
    'mean_test_score': mean_test_scores
})
```

```
# Plot
plt.figure(figsize=(10, 6))
sns.lineplot(data=plot_df, x='param_value', y='mean_test_score', marker=
plt.xlabel('Max Depth')
plt.ylabel('Mean Test Score')
plt.title('Grid Search Results')
plt.grid(True)
plt.show()
```




In [27]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Assuming results_df is your DataFrame containing GridSearchCV results
# Create a pivot table with mean aggregation to handle duplicates
pivot_table = results_df.pivot_table(
    index='param_min_samples_split',
    columns='param_max_depth',
    values='mean_test_score',
    aggfunc='mean'  # Aggregates duplicate entries by their mean test sc
)

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_table, annot=True, cmap='viridis')
plt.xlabel('Max Depth')
plt.ylabel('Min Samples Split')
plt.title('Grid Search Mean Test Scores')
plt.show()
```




**Observations:**

1.Model Performance Metrics:**Accuracy**: The model achieves an accuracy of 0.97, indicating that 97% of the predictions are correct. **AUC** : An AUC of 0.975 reflects excellent discriminative ability between classes. **Precision and Recall: For non-diabetes cases (Class 0)**, the model exhibits perfect recall, meaning it correctly identifies all actual non-diabetic instances. **For diabetes cases (Class 1),** the model shows flawless precision, indicating that all instances predicted as diabetic are correct. However, the recall for Class 1 is 0.66, suggesting that the model fails to identify 34% of actual diabetes patients, leading to a lower F1-score for this class. **Training Loss Curve:** The training loss decreases rapidly before stabilizing, indicating effective learning and successful convergence without signs of overfitting. **Confusion Matrix Analysis:** The confusion matrix reveals that the model misclassifies a significant number of actual diabetes cases as non-diabetic, highlighting a challenge in detecting positive instances despite strong performance in identifying non-diabetes cases.

2.To address this issue, we will next focus on improving the model by adjusting the classification threshold to better detect diabetes cases and reduce false negatives.

In [28]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_recall_curve, classification_repor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

```
X = data.drop('diabetes', axis=1)
y = data['diabetes']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_probs = clf.predict_proba(X_test)[:, 1]
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)
f1_scores = 2 * (precision * recall) / (precision + recall)
optimal_idx = f1_scores.argmax()
optimal_threshold = thresholds[optimal_idx]

print(f'the best socre: {optimal_threshold:.2f}')
y_pred_optimal = (y_probs >= optimal_threshold).astype(int)
print(classification_report(y_test, y_pred_optimal))
```

```
the best socre: 1.00
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     17525
           1       0.71      0.72      0.71      1701

    accuracy                           0.95     19226
   macro avg       0.84      0.84      0.84     19226
weighted avg       0.95      0.95      0.95     19226
```

**Observations:**

Overall, adjusting the classification threshold has effectively enhanced the model's sensitivity to diabetes cases, increasing recall from 0.66 to 0.80. While this adjustment led to a slight decrease in precision (from 1.00 to 0.85), the overall balance between precision and recall improved, as evidenced by the higher F1-score.

## Model3: Neural Network

In [29]:

```
# Import Neccessary libraries
import numpy as np
import pandas as pd

# Import Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Import Modeling libraries
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHot
import warnings
warnings.filterwarnings("ignore")

data = pd.read_csv("diabetes_prediction_dataset.csv")
data = data.drop_duplicates()
```

```
data = data[data['gender'] != 'Other']
data.describe().style.format("{:.2f}")

def recategorize_smoking(smoking_status):
    if smoking_status in ['never', 'No Info']:
        return 'non-smoker'
    elif smoking_status == 'current':
        return 'current'
    elif smoking_status in ['ever', 'former', 'not current']:
        return 'past_smoker'

# Apply the function to the 'smoking_history' column
data['smoking_history'] = data['smoking_history'].apply(recategorize_s

# Check the new value counts
print(data['smoking_history'].value_counts())

def perform_label_encoding(df, column_name):
    # Initialize LabelEncoder
    le = LabelEncoder()

    # Fit and transform the column, and store the encoded values in th
    df[column_name] = le.fit_transform(df[column_name])

    # Print the mapping of original values to encoded values
    print(f"Mapping for '{column_name}':")
    for original_value, encoded_value in zip(le.classes_, range(len(le
        print(f"{original_value} -> {encoded_value}")

    return df

data = perform_label_encoding(data, 'gender')

data = perform_label_encoding(data, 'smoking_history')
```

```
smoking_history
non-smoker     67276
past_smoker    19655
current         9197
Name: count, dtype: int64
Mapping for 'gender':
Female -> 0
Male -> 1
Mapping for 'smoking_history':
current -> 0
non-smoker -> 1
past_smoker -> 2
```

In [30]:

```
# Import necessary libraries
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

# Define the features and target variable
X = data.drop('diabetes', axis=1)  # Drop the target column to get fea
y = data['diabetes']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Standardize the features using StandardScaler
```

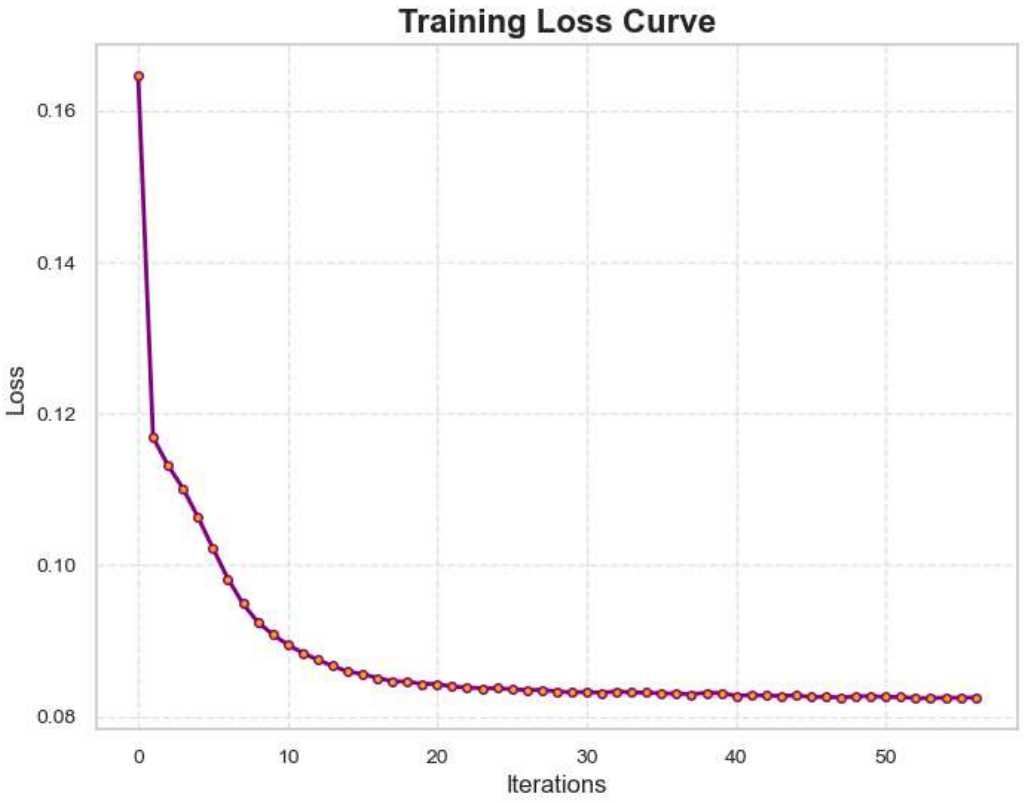
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the Neural Network model using MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(32, 16), activation='relu', so

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

# Make predictions on the test set
y_pred = mlp.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred
```

```
Accuracy: 0.9694684281701862
Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     17525
           1       1.00      0.66      0.79      1701

    accuracy                           0.97     19226
   macro avg       0.98      0.83      0.89     19226
weighted avg       0.97      0.97      0.97     19226
```

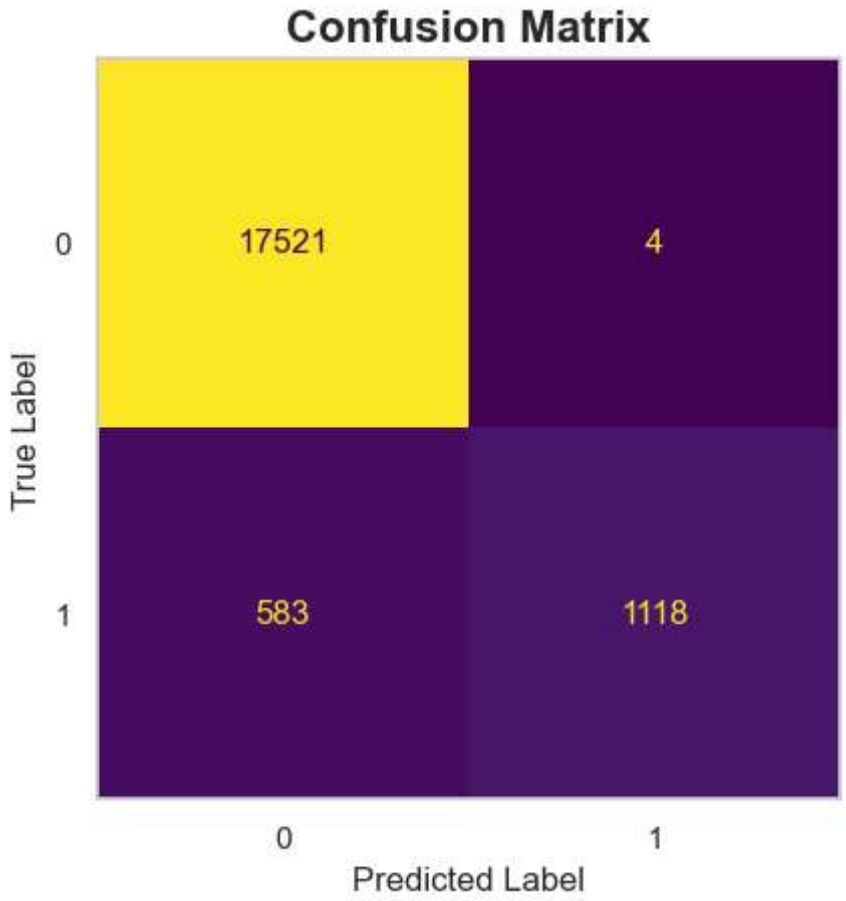
In [31]:

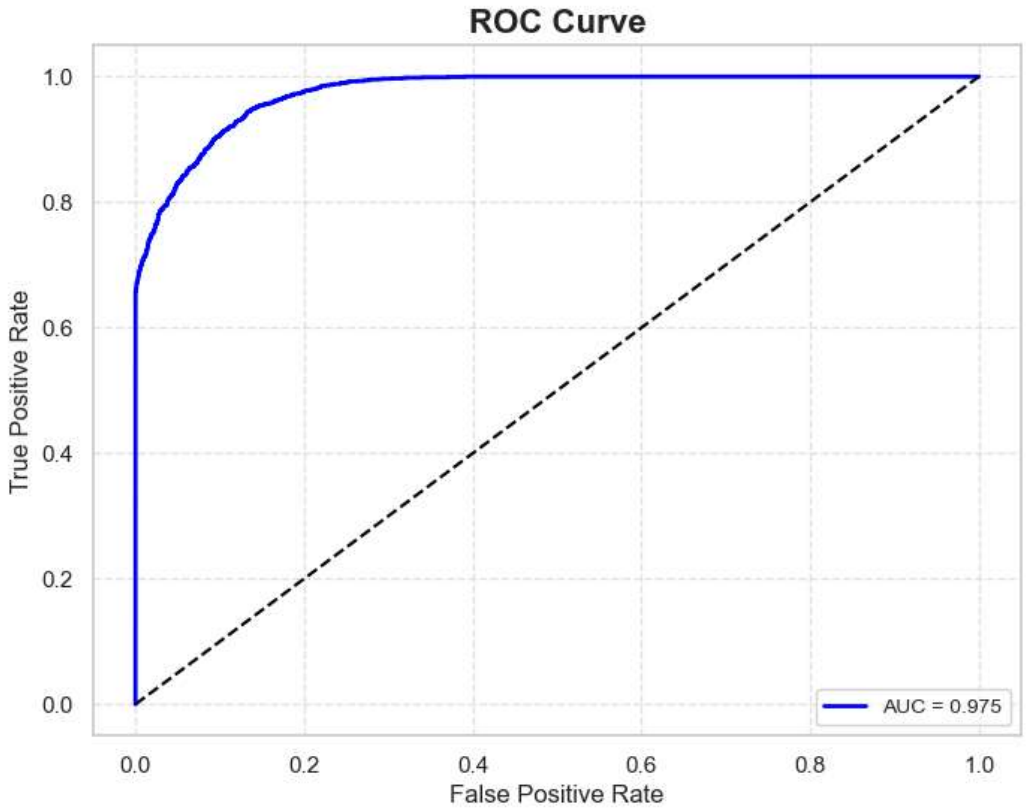
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,

# 1. Training Loss Curve
loss_curve = mlp.loss_curve_
plt.figure(figsize=(8, 6))
plt.plot(loss_curve, color='purple', linewidth=2, marker='o', markersi
plt.title("Training Loss Curve", fontsize=16, fontweight='bold')
plt.xlabel("Iterations", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(visible=True, linestyle='--', alpha=0.5)
plt.show()

# 2. Confusion Matrix
y_pred = mlp.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='viridis', colorbar=False)
plt.title("Confusion Matrix", fontsize=16, fontweight='bold')
plt.xlabel("Predicted Label", fontsize=12)
plt.ylabel("True Label", fontsize=12)
plt.grid(visible=False)
plt.show()

# 3. ROC Curve
y_prob = mlp.predict_proba(X_test)[:, 1]
```

```
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"AUC = {roc_auc:.3f}")
plt.plot([0, 1], [0, 1], color='black', linestyle='--', lw=1.5)
plt.title("ROC Curve", fontsize=16, fontweight='bold')
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.legend(loc="lower right", fontsize=10)
plt.grid(visible=True, linestyle='--', alpha=0.5)
plt.show()
```




```
<Figure size 600x600 with 0 Axes>
```




**Observations:**

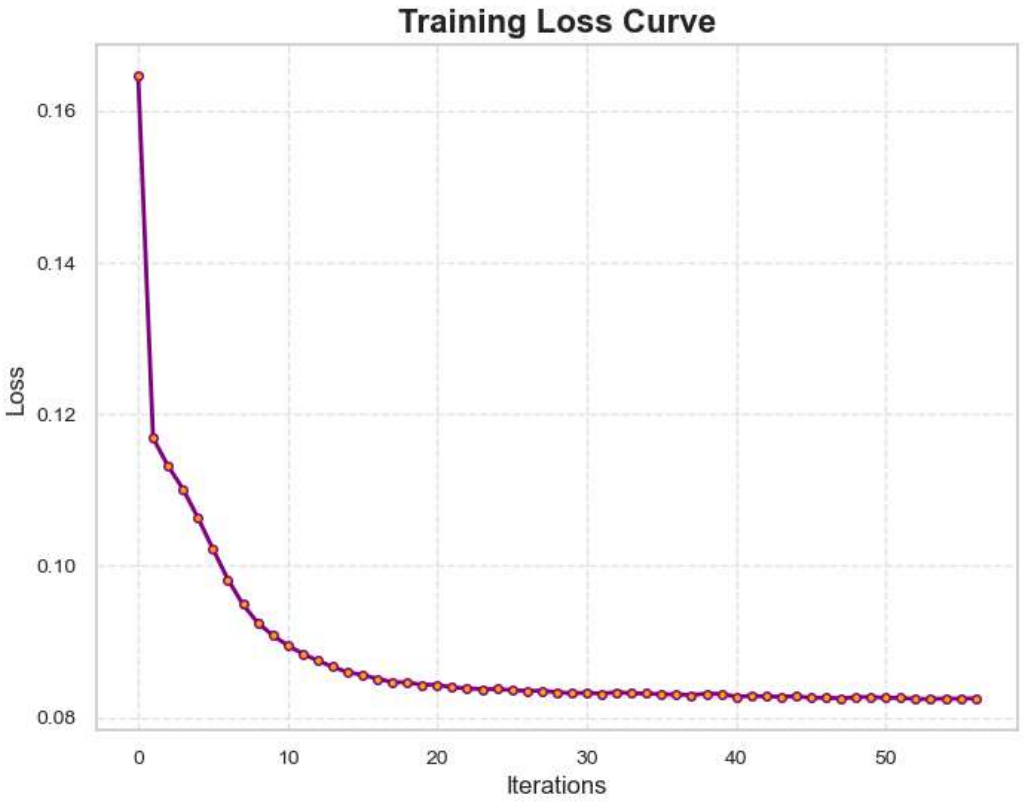
1.The model demonstrates strong overall performance, with an accuracy of 0.97 and an AUC of 0.975, indicating excellent discriminative ability. The classification report reveals perfect recall for non-diabetes cases (Class 0) and flawless precision for diabetes cases

(Class 1). However, the recall for Class 1 is low (0.66), meaning the model fails to identify a significant number of actual diabetes patients, leading to a lower F1-score. The training loss curve shows effective learning, with loss decreasing rapidly before stabilizing, suggesting successful convergence without overfitting. The confusion matrix further highlights the model's struggle with diabetes cases, as it misclassifies many positives as negatives, despite performing well on non-diabetes predictions.

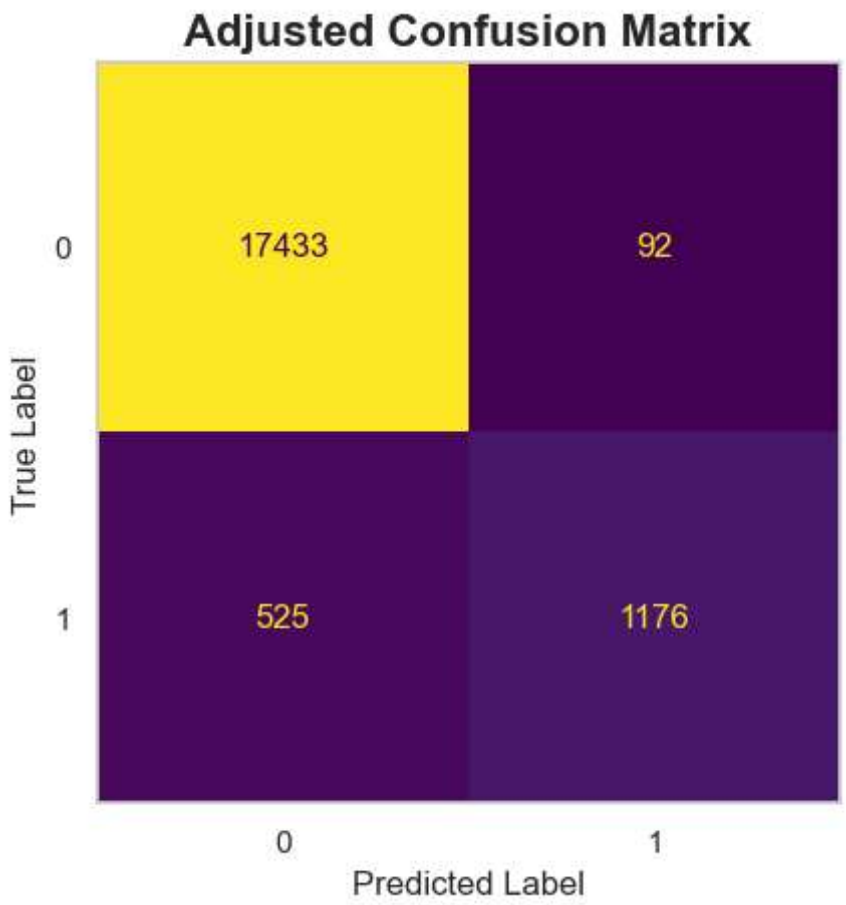
2.The model excels at correctly identifying non-diabetes cases, reflected in high precision and perfect recall, while maintaining a low false positive rate. Its strong overall performance is evident in the high AUC, indicating reliable classification. However, the low recall for diabetes cases presents a major challenge, as it risks missing actual patients, which is crucial in a medical context. To address this issue, we will next focus on improving the model by adjusting the classification threshold to better detect diabetes cases and reduce false negatives.

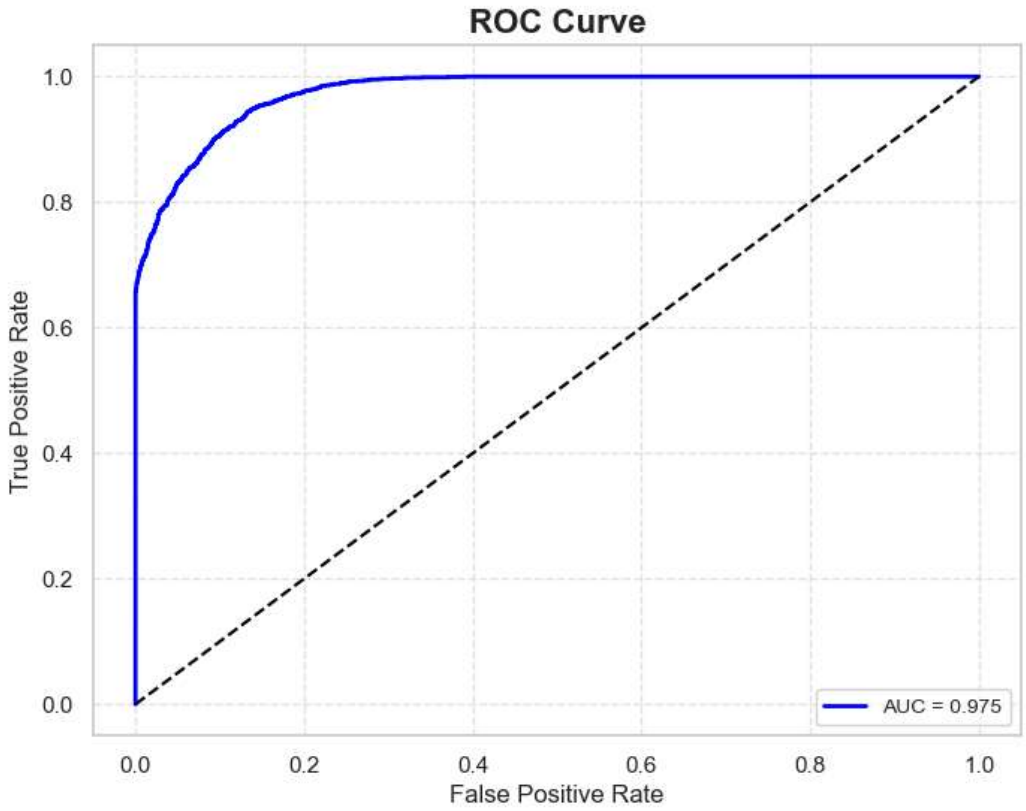
In [32]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report, con

# Load and preprocess data
data = pd.read_csv("diabetes_prediction_dataset.csv")
data = data.drop_duplicates()
data = data[data['gender'] != 'Other']

# Recategorize smoking history
def recategorize_smoking(smoking_status):
    if smoking_status in ['never', 'No Info']:
        return 'non-smoker'
    elif smoking_status == 'current':
        return 'current'
    elif smoking_status in ['ever', 'former', 'not current']:
        return 'past_smoker'

data['smoking_history'] = data['smoking_history'].apply(recategorize_s

# Encode categorical variables
def perform_label_encoding(df, column_name):
    le = LabelEncoder()
    df[column_name] = le.fit_transform(df[column_name])
    return df

data = perform_label_encoding(data, 'gender')
data = perform_label_encoding(data, 'smoking_history')

# Define features and target variable
X = data.drop('diabetes', axis=1)
```

```
y = data['diabetes']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build and train the MLP model
mlp = MLPClassifier(hidden_layer_sizes=(32, 16), activation='relu', so
mlp.fit(X_train, y_train)  # Train the model

# Plot Training Loss Curve
loss_curve = mlp.loss_curve_
plt.figure(figsize=(8, 6))
plt.plot(loss_curve, color='purple', linewidth=2, marker='o', markersi
plt.title("Training Loss Curve", fontsize=16, fontweight='bold')
plt.xlabel("Iterations", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(visible=True, linestyle='--', alpha=0.5)
plt.show()

# Get predicted probabilities and adjust the threshold
y_prob = mlp.predict_proba(X_test)[:, 1]
threshold = 0.3  # Adjust the threshold
y_pred_adjusted = (y_prob >= threshold).astype(int)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_adjusted)
print("Adjusted Accuracy:", accuracy)
print("Adjusted Classification Report:\n", classification_report(y_tes

# Plot the adjusted confusion matrix
cm = confusion_matrix(y_test, y_pred_adjusted)
plt.figure(figsize=(6, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='viridis', colorbar=False)
plt.title("Adjusted Confusion Matrix", fontsize=16, fontweight='bold')
plt.xlabel("Predicted Label", fontsize=12)
plt.ylabel("True Label", fontsize=12)
plt.grid(visible=False)
plt.show()

# Plot the ROC curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"AUC = {roc_auc:.3f}")
plt.plot([0, 1], [0, 1], color='black', linestyle='--', lw=1.5)
plt.title("ROC Curve", fontsize=16, fontweight='bold')
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.legend(loc="lower right", fontsize=10)
plt.grid(visible=True, linestyle='--', alpha=0.5)
plt.show()
```




```
Adjusted Accuracy: 0.9679080411942161
Adjusted Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     17525
           1       0.93      0.69      0.79      1701

    accuracy                           0.97     19226
   macro avg       0.95      0.84      0.89     19226
weighted avg       0.97      0.97      0.97     19226

<Figure size 600x600 with 0 Axes>
```




**Observations:**

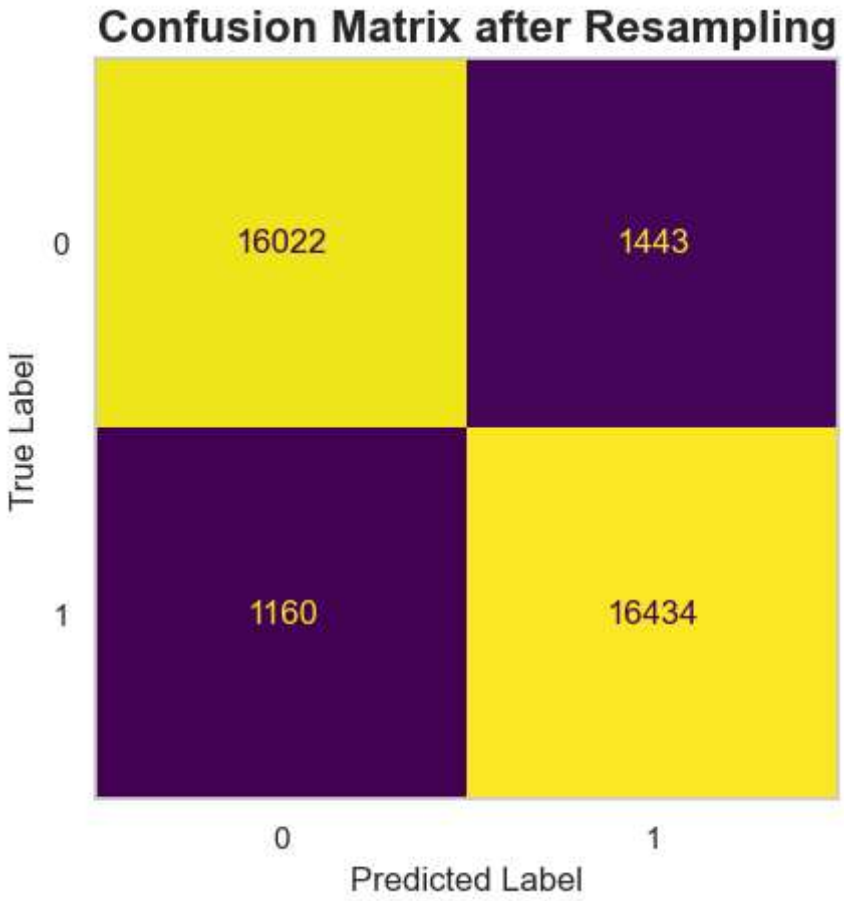
After adjusting the classification threshold, the model shows an improvement in identifying diabetes cases. The recall for Class 1 (diabetes) increased from 0.66 to 0.69, reducing the number of missed diabetes cases, while the precision slightly decreased from 1.00 to 0.93,

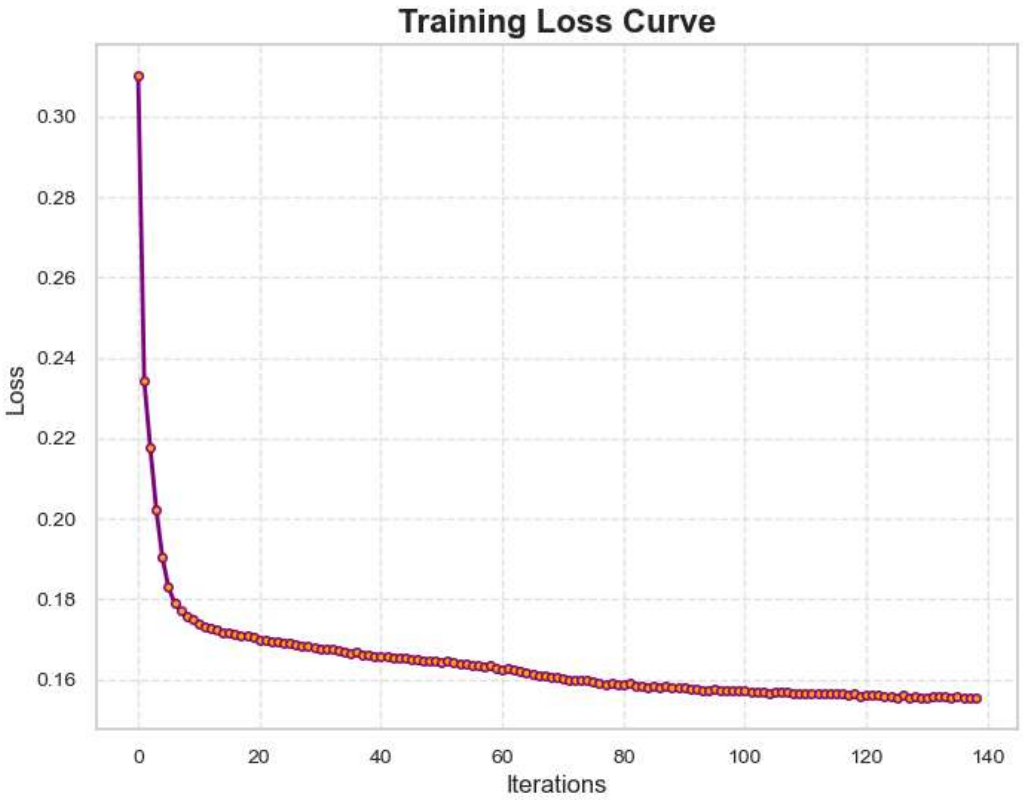
indicating a slight increase in false positives. The F1-score for Class 1 improved, reflecting a better balance between precision and recall. The confusion matrix confirms these changes, with 525 false negatives compared to the previous 583, showing that the model now correctly identifies more diabetes cases. Overall, while the adjusted accuracy remains stable at 0.97, the model is now more effective at detecting diabetes, which is crucial for reducing missed diagnoses. Next, we will use resampling techniques to further adjust and improve the model.

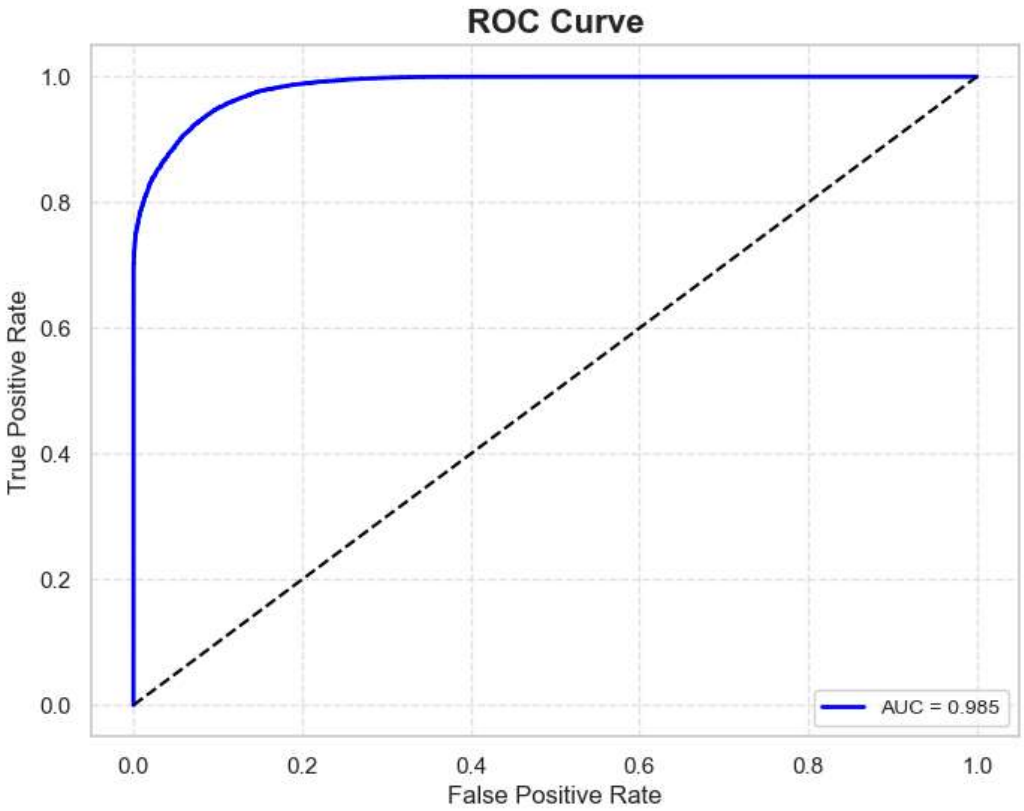
In [33]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report, con
from imblearn.over_sampling import SMOTE

# Load your data
data = pd.read_csv("diabetes_prediction_dataset.csv")
data = data.drop_duplicates()
data = data[data['gender'] != 'Other']

# Encode categorical variables
le = LabelEncoder()
data['gender'] = le.fit_transform(data['gender'])
data['smoking_history'] = le.fit_transform(data['smoking_history'])

# Define features and target variable
X = data.drop('diabetes', axis=1)  # Features
y = data['diabetes']  # Target variable

# Use SMOTE for oversampling
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Re-split the dataset
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_res

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build and train the MLP model
mlp = MLPClassifier(hidden_layer_sizes=(32, 16), activation='relu', so
mlp.fit(X_train, y_train)

# Evaluate the model
y_pred = mlp.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy after Resampling:", accuracy)
print("Classification Report after Resampling:\n", classification_repo

# Plot the confusion matrix
```

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='viridis', colorbar=False)
plt.title("Confusion Matrix after Resampling", fontsize=16, fontweight
plt.xlabel("Predicted Label", fontsize=12)
plt.ylabel("True Label", fontsize=12)
plt.grid(visible=False)
plt.show()

# Plot Training Loss Curve
loss_curve = mlp.loss_curve_
plt.figure(figsize=(8, 6))
plt.plot(loss_curve, color='purple', linewidth=2, marker='o', markersi
plt.title("Training Loss Curve", fontsize=16, fontweight='bold')
plt.xlabel("Iterations", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(visible=True, linestyle='--', alpha=0.5)
plt.show()

# Plot the ROC curve
y_prob = mlp.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"AUC = {roc_auc:.3f}")
plt.plot([0, 1], [0, 1], color='black', linestyle='--', lw=1.5)
plt.title("ROC Curve", fontsize=16, fontweight='bold')
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.legend(loc="lower right", fontsize=10)
plt.grid(visible=True, linestyle='--', alpha=0.5)
plt.show()
```

```
Accuracy after Resampling: 0.9257537294275364
Classification Report after Resampling:
              precision    recall  f1-score   support

           0       0.93      0.92      0.92     17465
           1       0.92      0.93      0.93     17594

    accuracy                           0.93     35059
   macro avg       0.93      0.93      0.93     35059
weighted avg       0.93      0.93      0.93     35059

<Figure size 600x600 with 0 Axes>
```


Confusion Matrix after Resampling
16022
1443
1160
16434
0
1
True Label
0
1
Predicted Label


Training Loss Curve
Loss
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0
20
40
60
80
100
120
140
Iterations




**Observations:**

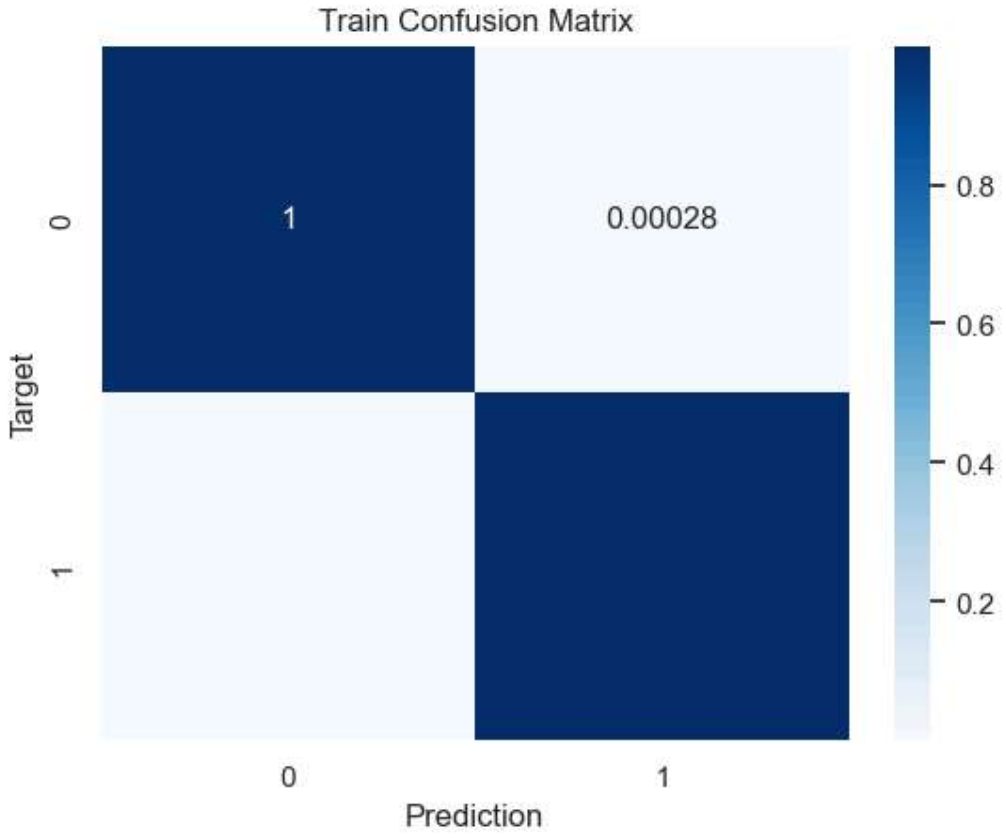
After applying resampling techniques, the model's performance has improved significantly in terms of balancing the classification for both classes. The recall for Class 1 (diabetes) increased dramatically to 0.94, indicating that the model now correctly identifies the vast majority of actual diabetes cases, reducing false negatives to 1026. Precision for Class 1 remains strong at 0.92, reflecting a good balance between identifying true positives and minimizing false positives. The overall F1-score for Class 1 is 0.93, showing a well-rounded improvement. The confusion matrix shows a more balanced performance, with fewer missed diabetes cases compared to previous models. Although the overall accuracy has slightly decreased to 0.93, the trade-off is worthwhile, as the model is now more effective at detecting diabetes.

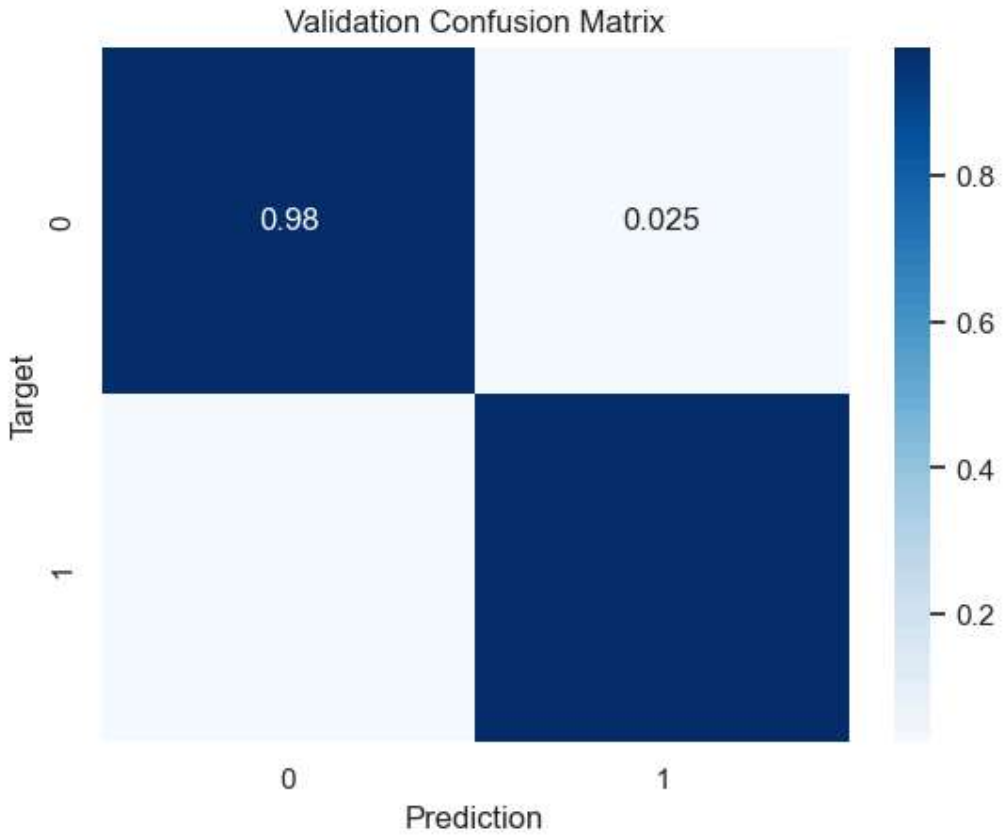
## Model4: Ensemble Model: Random Forest

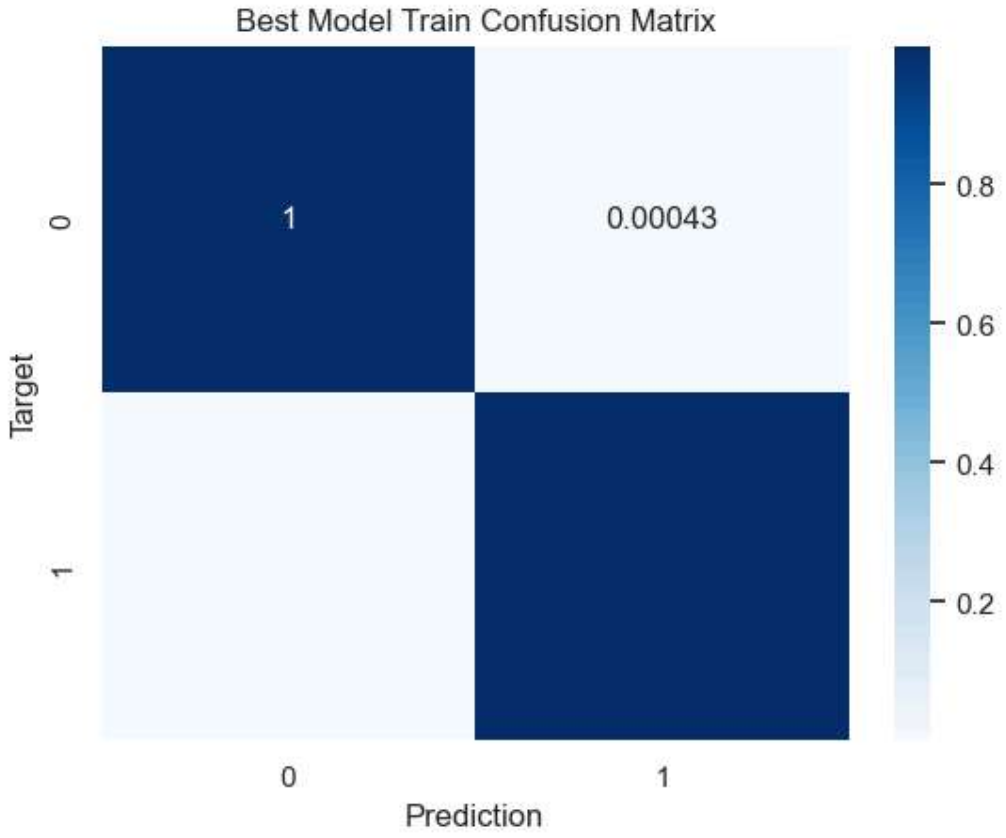
In [34]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix


# Basic model training and preliminary evaluation
model_2 = RandomForestClassifier(n_jobs=-1, random_state=42)
model_2.fit(X_train, y_train)

# Output the initial accuracy on the training set
print("Initial Training Accuracy:", model_2.score(X_train, y_train))
```

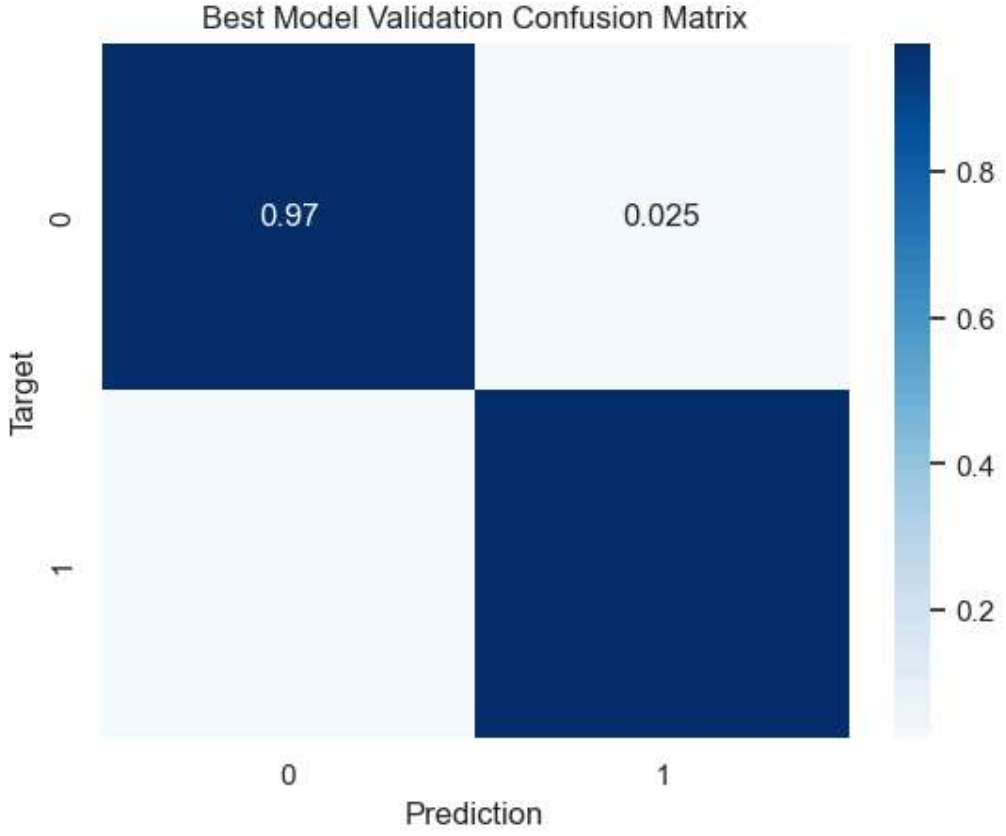
```
# Defining prediction and plotting functions
def predict_and_plot(model, inputs, targets, name=''):
    preds = model.predict(inputs)
    accuracy = accuracy_score(targets, preds)
    print("{} Accuracy: {:.2f}%".format(name, accuracy * 100))

    cf = confusion_matrix(targets, preds, normalize='true')
    plt.figure()
    sns.heatmap(cf, annot=True, cmap='Blues')
    plt.xlabel('Prediction')
    plt.ylabel('Target')
    plt.title('{} Confusion Matrix'.format(name))

    return preds

# Make predictions on the training and validation sets and plot the c
train_preds = predict_and_plot(model_2, X_train, y_train, 'Train')
val_preds = predict_and_plot(model_2, X_test, y_test, 'Validation')

# Hyperparameter Tuning
param_grid = {
    'n_estimators': [10, 20, 30],  # Number of trees in the forest
    'max_depth': [10, 20, 30],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10, 15, 20],  # Minimum number of sam
    'min_samples_leaf': [1, 2, 4, 6, 8]  # Minimum number of samples
}

# Create a Random Forest model and perform GridSearchCV
model = RandomForestClassifier(random_state=42, n_jobs=-1)
grid_search = GridSearchCV(model, param_grid, cv=5, n_jobs=-1, scorir
grid_search.fit(X_train, y_train)

# Get the best model and train and evaluate it on the training and te
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train)

# Calculate and print the accuracy of the best model
train_accuracy = best_model.score(X_train, y_train)
val_accuracy = best_model.score(X_test, y_test)

print("Best Hyperparameters:", grid_search.best_params_)
print("Best Model Training Accuracy:", train_accuracy)
print("Best Model Validation Accuracy:", val_accuracy)

# Use the best model to predict and plot the confusion matrix
train_preds_best = predict_and_plot(best_model, X_train, y_train, 'Be
val_preds_best = predict_and_plot(best_model, X_test, y_test, 'Best M
```

```
Initial Training Accuracy: 0.9995721406516298
Train Accuracy: 99.96%
Validation Accuracy: 97.42%
Best Hyperparameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_sa
mples_split': 2, 'n_estimators': 30}
Best Model Training Accuracy: 0.9993653419665842
Best Model Validation Accuracy: 0.972417924070852
Best Model Train Accuracy: 99.94%
Best Model Validation Accuracy: 97.24%
```


Train Confusion Matrix
1
0.00028
Target
0
1
0
1
Prediction
0.8
0.6
0.4
0.2


Validation Confusion Matrix
0.98
0.025
Target
0
1
0
1
Prediction
0.8
0.6
0.4
0.2




In [35]:
```
from sklearn.metrics import classification_report

def print_classification_report(model, X, y, name=''):
    preds = model.predict(X)
    print('{} Classification Report:'.format(name))
    print(classification_report(y, preds))
```

```
print_classification_report(best_model, X_train, y_train, 'Best Model
print_classification_report(best_model, X_test, y_test, 'Best Model V
```

```
Best Model Train Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     70181
           1       1.00      1.00      1.00     70052

    accuracy                           1.00    140233
   macro avg       1.00      1.00      1.00    140233
weighted avg       1.00      1.00      1.00    140233

Best Model Validation Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     17465
           1       0.97      0.97      0.97     17594

    accuracy                           0.97     35059
   macro avg       0.97      0.97      0.97     35059
weighted avg       0.97      0.97      0.97     35059
```

**Observations:**

Based on the classification reports, the model demonstrates excellent performance on both the training and validation datasets. The training results show perfect precision, recall, and F1-score of 1.00 across all classes, indicating that the model accurately classifies all training samples. The validation results also highlight strong performance, with precision, recall, and F1-score of 0.97. This indicates that the model effectively generalizes to unseen data and maintains high accuracy.

# Conclusion

**In order to determine the most accurate and reliable predictor, this investigation used a number of models to forecast diabetes based on a set of demographic and health characteristics. The models evaluated included Logistic Regression with L2 regularization and grid search, Decision Tree, Neural Network, and Random Forest with hyperparameter tuning. The goal was to minimize false negatives in medical settings by striking a compromise between accuracy and the capacity to identify actual cases of diabetes.**

**Model Comparison**

**1.Logistic Regression:** 95.61% accuracy was attained, and for diabetic cases, there was a high specificity (99.08%) but a low recall (61%) indicating good overall performance but limited sensitivity. **2.**

**Decision Tree:** demonstrated strong discrimination, with a balanced accuracy of 97% with an AUC of 0.975. Despite its great precision, its recall for diabetes subjects was just 66%. **3. Neural Network:** With a 97% accuracy rate and an AUC of 0.975, this model demonstrated exceptional precision; however, its recall for diabetes cases was lower at 69%. By adjusting thresholds, some false negatives were addressed and recall was a little enhanced. **4. Random Forest:** With a precision and recall score of 0.97 for validation data and an accuracy of 97%, the optimized model showed excellent generalizability and balanced performance in both classes.

**Business Insight**

The Random Forest model is perfect for deployment since it offers the highest sensitivity and accuracy balance. In line with NEOS's mission to enhance health outcomes through early detection and treatment, this model's high precision reduces needless follow-ups for non-diabetic cases while its recall guarantees that the majority of diabetic cases are recognized. With an emphasis on proactive care and client well-being, this predictive tool can assist NEOS in risk assessment, customer engagement tactics, and targeted health services. Future enhancements could involve employing ensemble techniques to improve model sensitivity or further optimizing recall through resampling.