

Phase 3 Project



Introduction

Diabetes is a disease that occurs when your blood glucose, also called blood sugar, is too high. Other factors such as gender, age, and bmi can also be contributors to the disease. This project will be focused in building models that can help predict whether a patient has the disease or not. This can be useful for healthcare professionals in identifying patients who may be at risk of developing diabetes and in developing personalized treatment plans. Additionally, the dataset can be used by researchers to explore the relationships between various medical and demographic factors and the likelihood of developing diabetes.

Problem Statement

The head of medical Board has tasked me to create a predictive model that can predict whether a patient has diabetes or not.

To answer the question, I will build a classifier to predict whether patients have cancer with the given dataset. This will be based on several diagnostic measurements, including glucose level, blood pressure, smoking history, gender, BMI, and age. I will train models such as logistic regression, decision trees and ensemble methods such as random forests and evaluate which model performs better.

Objectives

My top objectives for modelling will be:

1. Determine important factors that affect diabetes.
2. Determine the best model to predict diabetes.
3. Create a model that predicts diabetes with an accuracy of 90% or above.

Data sources

The data for this modelling was obtained from kaggle.

The dataset can be found by clicking the following link: [Diabetes Dataset](#)

Data Understanding

In this section I will try to understand the dataset before getting into any analysis.

This will involve understanding the shape of the dataset, the columns present and what they represent, the datatypes of the columns and any missing values.

Explanation of columns used:

1. Age: This is the age of all patients diagnosed.
2. Gender: The gender column displays whether one is male or female.
3. Bmi: This is the body mass index of all patients which is a measure of body fat based on a person's height and weight.
4. Hypertension: This column displays whether the patient has high blood pressure or not.
5. Heart disease: This column displays whether the patient has heart disease or not.
6. Smoking history: This column shows the history of the patient regarding smoking.
7. HbA1c Level: HbA1c (glycated hemoglobin) is a measure of the average blood glucose level over the past 2-3 months.
8. Blood Glucose Level: Blood glucose level refers to the amount of glucose (sugar) present in the blood at a given time.

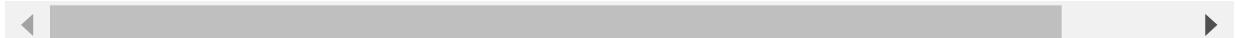
In [1]:

```
# importing all the packages I will use
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
%matplotlib inline
# ignore warnings
import warnings
warnings.filterwarnings('ignore')
# modelling
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split, GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
```

I load in the data first and check the first 5 rows

```
In [2]: # reading the dataset
df = pd.read_csv("./Data/diabetes.csv")
df.head()
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level
0	Female	80.0	0	1	never	25.19	6.6	148
1	Female	54.0	0	0	No Info	27.32	6.6	85
2	Male	28.0	0	0	never	27.32	5.7	113
3	Female	36.0	0	0	current	23.45	5.0	130
4	Male	76.0	1	1	current	20.14	4.8	120



From the head we can see that most of the features are numeric.

Gender and smoking history can be seen as categorical.

```
In [3]: # check for the shape of the dataset
df.shape
```

```
Out[3]: (100000, 9)
```

The dataset has 9 columns of features and 100,000 rows

```
In [4]: # check information about the dataset
df.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
```

From the information we have **9 columns** and **100,000 rows**.

- Gender and smoking history columns are of object datatype.
- Age, bmi and HbA1c_level columns are of float datatype.
- Hypertension, heart disease, blood glucose level and diabetes columns are of int datatype.
- The **diabetes column** will be used as the **target** while the **other columns** will be the **predictors**.

```
In [5]: # check for duplicates
df.duplicated().sum()
```

Out[5]: 3854

We have 3854 duplicates in our dataset. These will be dropped during data cleaning.

In [6]: `# description about the dataset
df[['age', 'bmi', 'HbA1c_level', 'blood_glucose_level']].describe()`

	age	bmi	HbA1c_level	blood_glucose_level
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	41.885856	27.320767	5.527507	138.058060
std	22.516840	6.636783	1.070672	40.708136
min	0.080000	10.010000	3.500000	80.000000
25%	24.000000	23.630000	4.800000	100.000000
50%	43.000000	27.320000	5.800000	140.000000
75%	60.000000	29.580000	6.200000	159.000000
max	80.000000	95.690000	9.000000	300.000000

This creates a brief statistical description of our dataset.

We can see that the mean age in our dataset is 41 years and our mean for bmi(body mass index) is 27.

In [7]: `# check for any missing values
df.isna().sum()`

Out[7]: gender 0
age 0
hypertension 0
heart_disease 0
smoking_history 0
bmi 0
HbA1c_level 0
blood_glucose_level 0
diabetes 0
dtype: int64

We have no missing values in our dataset.

In [8]: `# create a copy of the dataset for further analysis keeping the original intact
diabetes_df = df.copy()
diabetes_df.head()`

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level
0	Female	80.0	0	1	never	25.19	6.6	125
1	Female	54.0	0	0	No Info	27.32	6.6	113
2	Male	28.0	0	0	never	27.32	5.7	113
3	Female	36.0	0	0	current	23.45	5.0	113
4	Male	76.0	1	1	current	20.14	4.8	113



Data Cleaning

This is where I work on the dataset to ensure it is goof for use in modelling.

```
In [9]: # drops the duplicates in the data
diabetes_df = diabetes_df.drop_duplicates()
diabetes_df.shape
```

```
Out[9]: (96146, 9)
```

```
In [10]: # check for distinct values

for col in diabetes_df.columns:
    distinct_vals = len(diabetes_df[col].unique())
    print(f"{col}: {distinct_vals} unique values")
```

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

```
In [11]: # check value counts for gender
diabetes_df['gender'].value_counts()
```

```
Out[11]: Female      56161
Male        39967
Other         18
Name: gender, dtype: int64
```

```
In [12]: # normalize the counts
diabetes_df['gender'].value_counts(normalize=True)
```

```
Out[12]: Female      0.584122
Male        0.415691
Other       0.000187
Name: gender, dtype: float64
```

From that we can see that female and male have more values while other only has 18 values which is 0.000187%. We will remove that 'other' category to only have female and male.

```
In [13]: # Drop the 'Other' category
diabetes_df = diabetes_df[diabetes_df['gender'].isin(['Female', 'Male'])]

# Verify the value counts to confirm
print(diabetes_df['gender'].value_counts())
```

```
Female      56161
Male        39967
Name: gender, dtype: int64
```

```
In [14]: diabetes_df['smoking_history'].value_counts()
```

```
Out[14]: never      34395
No Info     32881
former      9299
current     9197
not current 6359
ever        3997
Name: smoking_history, dtype: int64
```

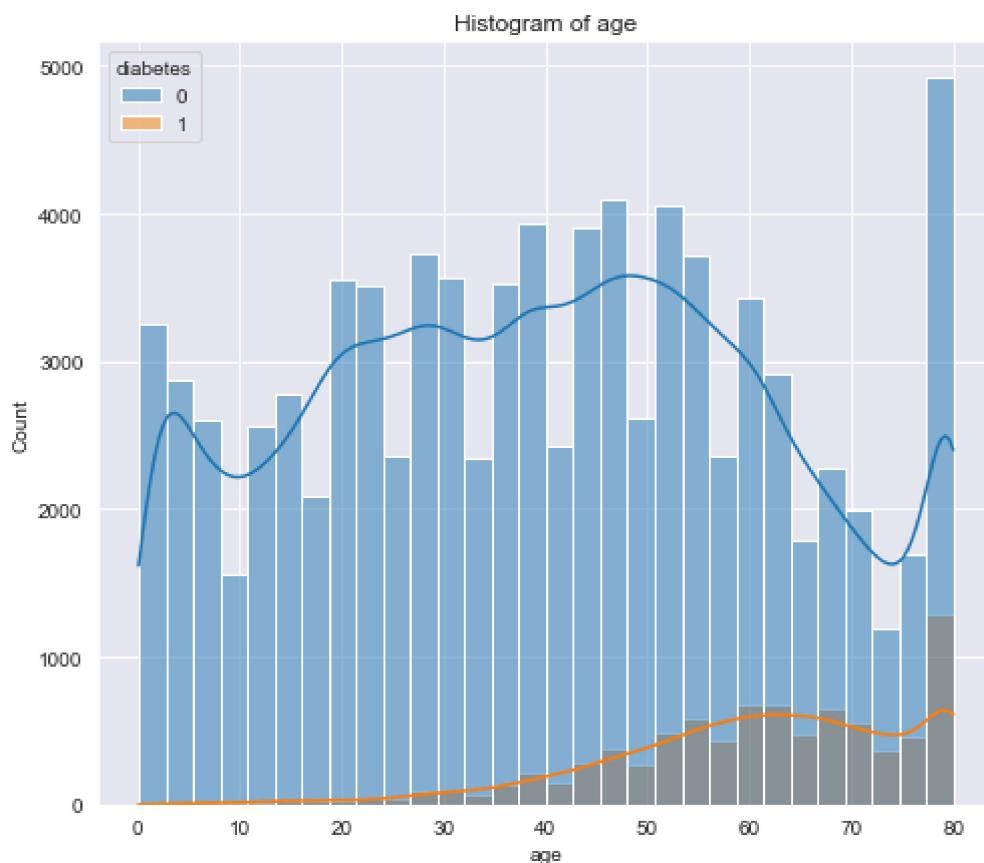
```
In [15]: # check the columns
diabetes_df.columns
```

```
Out[15]: Index(['gender', 'age', 'hypertension', 'heart_disease', 'smoking_history',
       'bmi', 'HbA1c_level', 'blood_glucose_level', 'diabetes'],
      dtype='object')
```

Exploratory Data Analysis

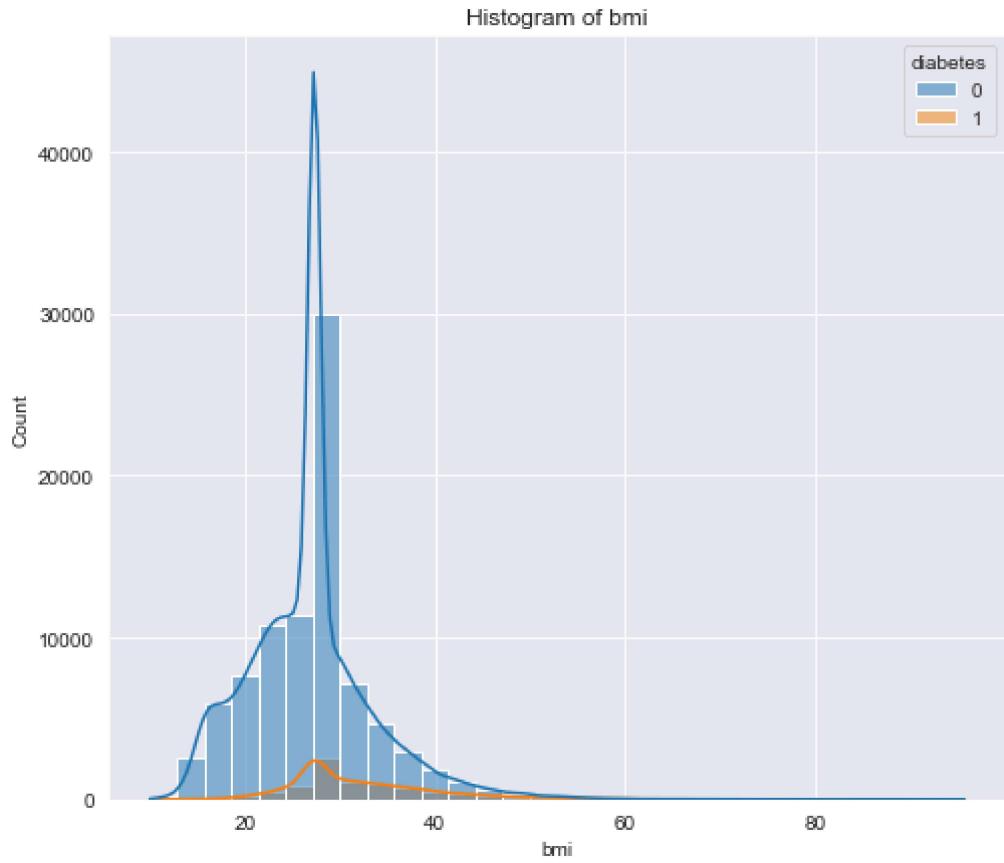
```
In [16]: # create a function for plotting a histogram
def plot_hist(df, col):
    fig, ax = plt.subplots(figsize=(8,7))
    sns.histplot(data=df, x=col, bins=30, hue= 'diabetes', kde=True)
    plt.title(f"Histogram of {col}")
    plt.xlabel(f"{col}")
    plt.ylabel("Count");
```

```
In [17]: # check distribution of age
plot_hist(diabetes_df, "age")
```



I can see that the highest age is 80 and more ages are distributed between 20 to around 65 years. Above the ages of 40, I can see an increase in patients with diabetes.

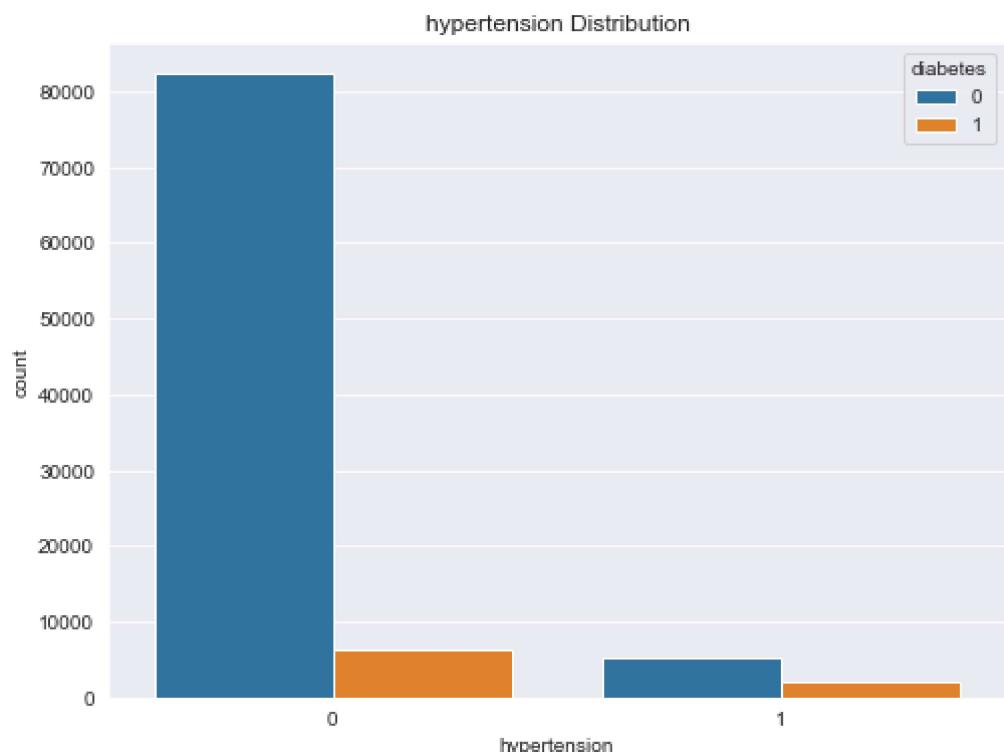
```
In [18]: plot_hist(diabetes_df, "bmi")
```



I can see a spike of bmi around 30. From the kde curve the distribution is not normally distributed.

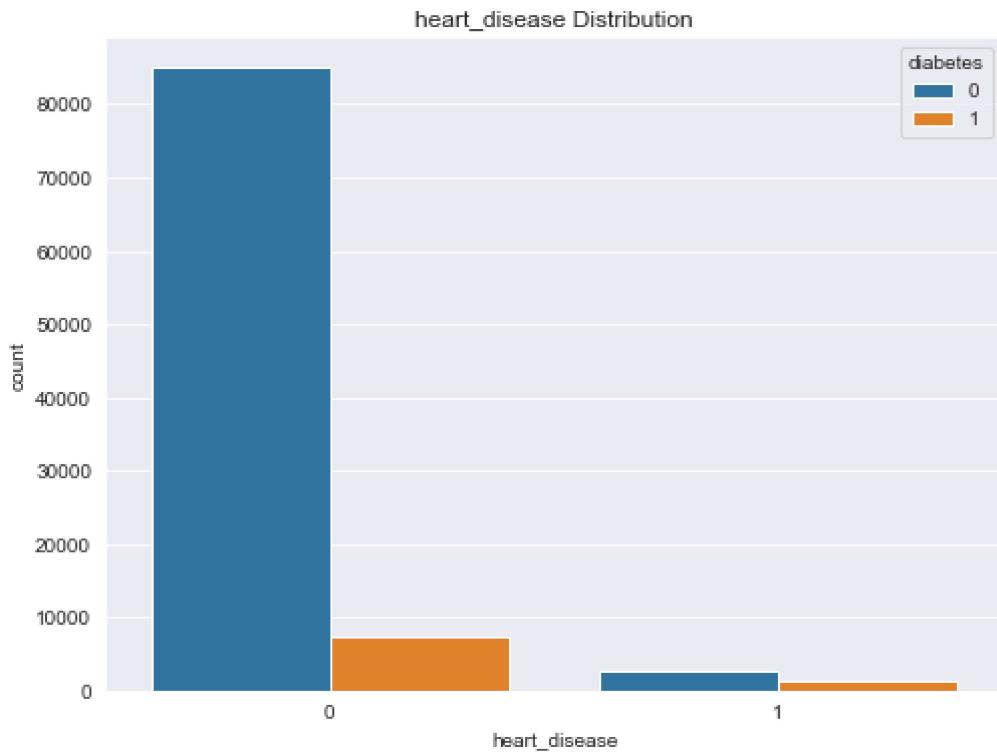
```
In [19]: # create a function to plot countplots
def count_plot(df, col):
    fig, ax = plt.subplots(figsize=(8,6))
    sns.countplot(x=col, data=df, hue= 'diabetes')
    plt.title(f'{col} Distribution');
```

```
In [20]: count_plot(diabetes_df, "hypertension")
```



- I can see that people with no hypertension are more than those with hypertension.
- For people with **no hypertension**, there is **less risk** in having diabetes as compared to those with hypertension.

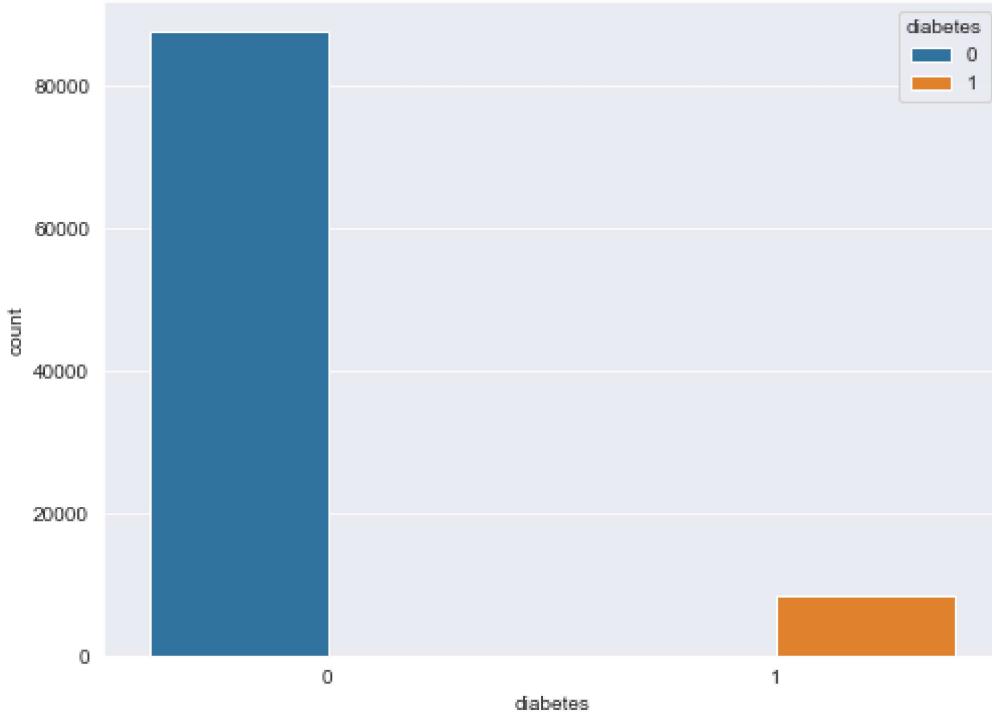
In [21]: `count_plot(diabetes_df, "heart_disease")`



- Patients with **no heart** disease are more than those with heart disease.
- Patients with **heart disease** have a higher chance of having diabetes.

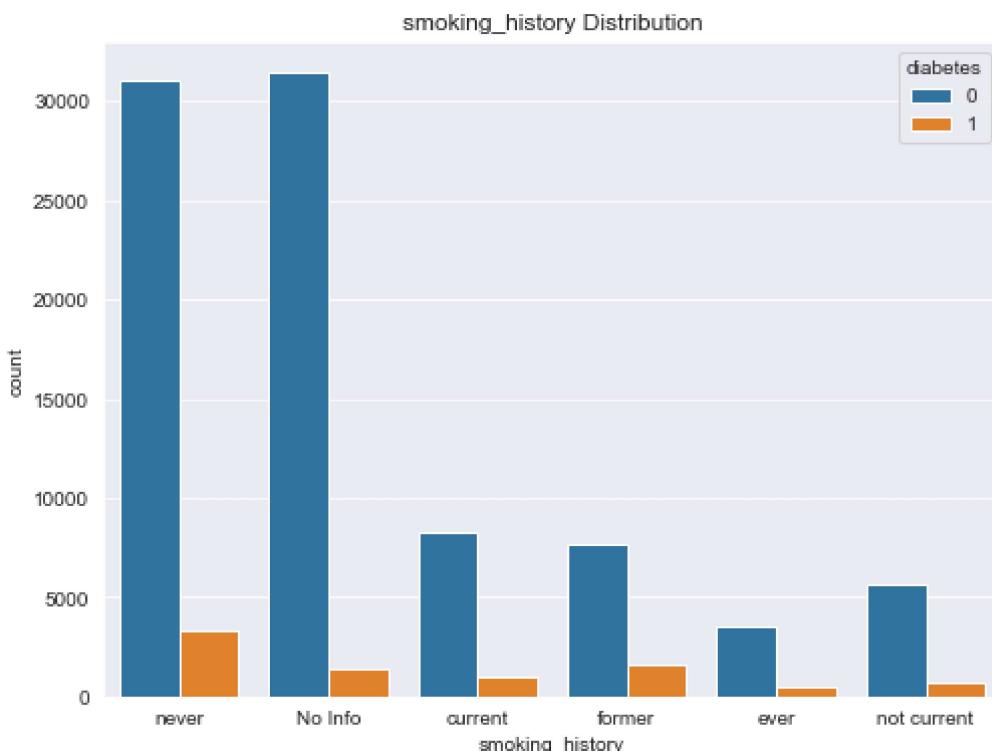
In [22]: `count_plot(diabetes_df, "diabetes")`

diabetes Distribution



- People with no diabetes are more than those with diabetes.
- This is the target and we can see that the distribution of patients with diabetes are less. This can be attributed to people not having hypertension or heart disease as they are some causes of diabetes from domain knowledge, or lack of enough data collection.

In [23]: `count_plot(diabetes_df, "smoking_history")`

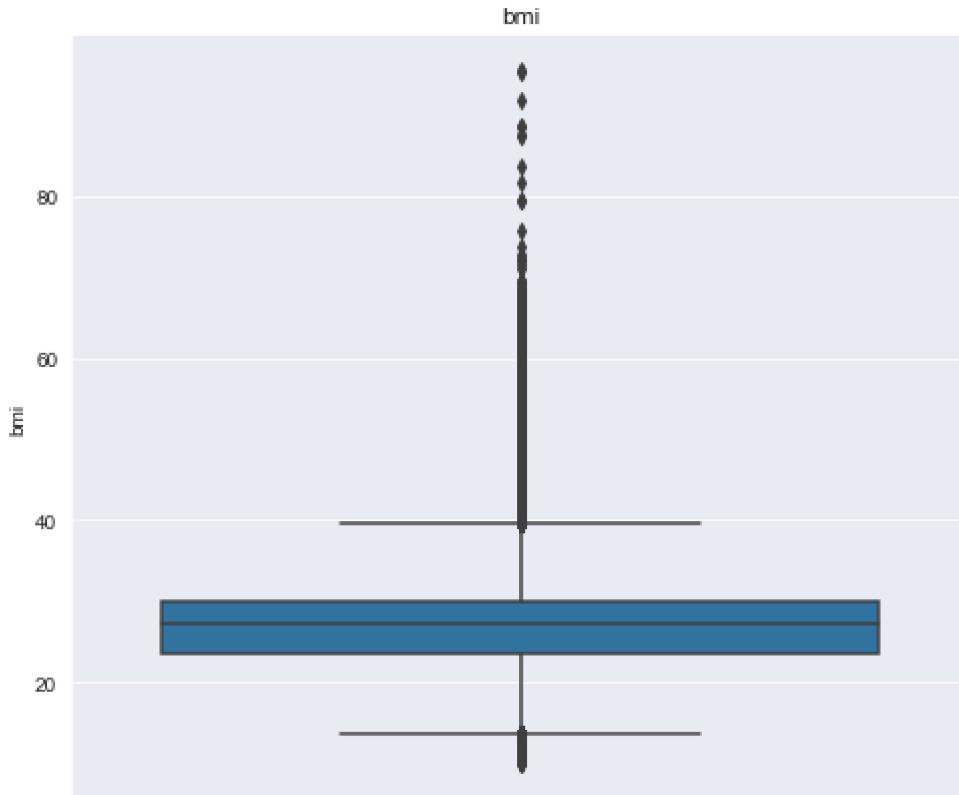


- People who never smoked are more than any current smokers

In [24]: `# create a function for box plots vs diabetes`
`def box_plot(df, col):`

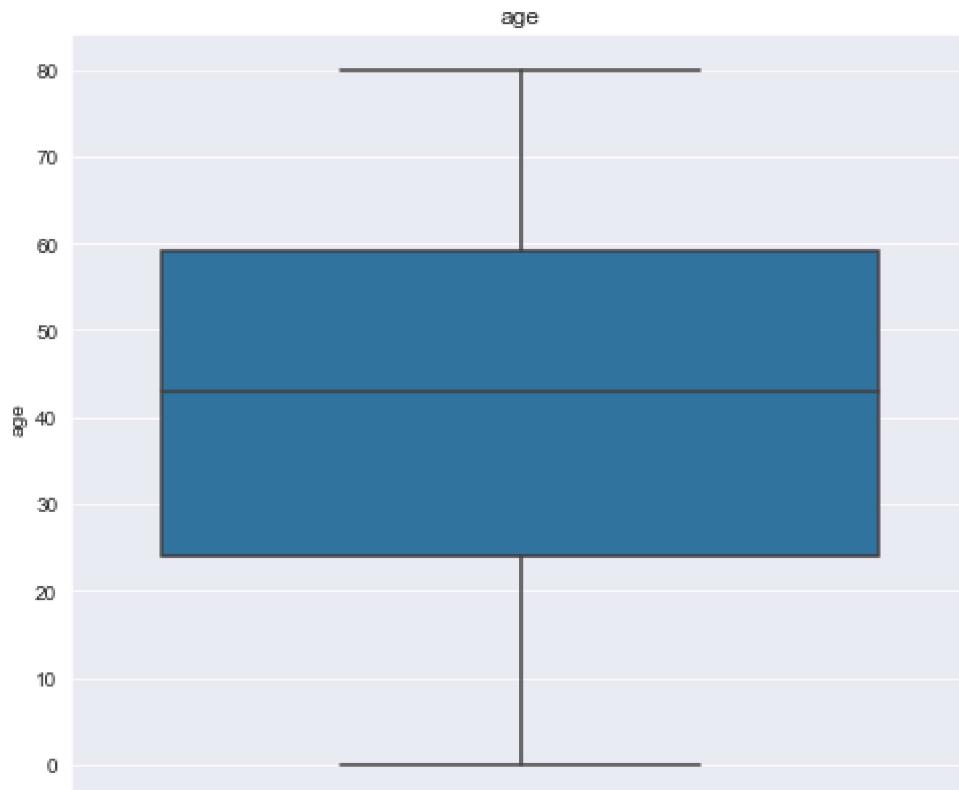
```
fig, ax = plt.subplots(figsize=(8,7))
sns.boxplot(y=col, data=df)
plt.title(f"{col}");
```

In [25]: `box_plot(diabetes_df, "bmi")`



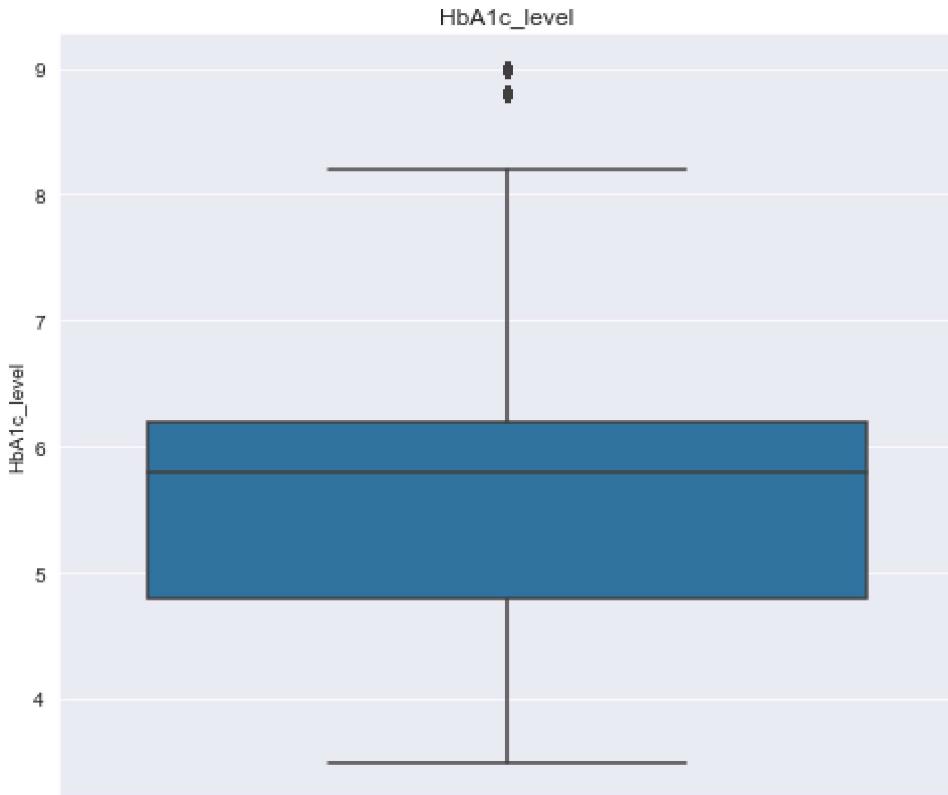
- There are outliers in bmi on both sides of the box plot. These outliers can be useful in modelling so I will keep them.

In [26]: `box_plot(diabetes_df, "age")`



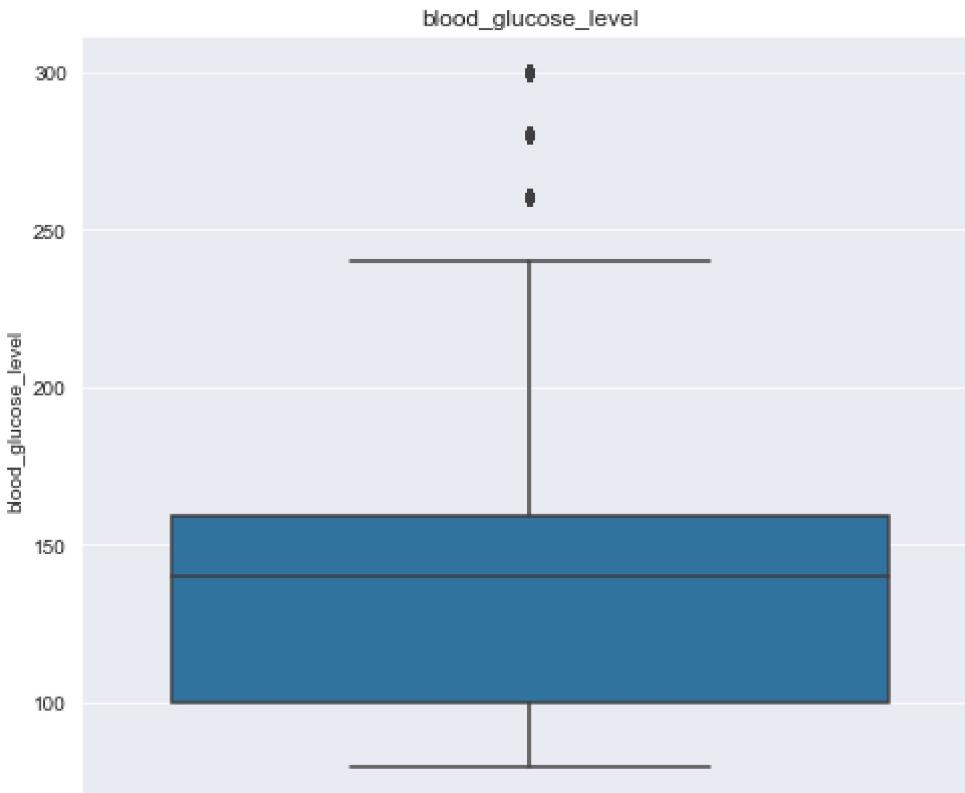
- There are no outliers in age.

```
In [27]: box_plot(diabetes_df, "HbA1c_level")
```



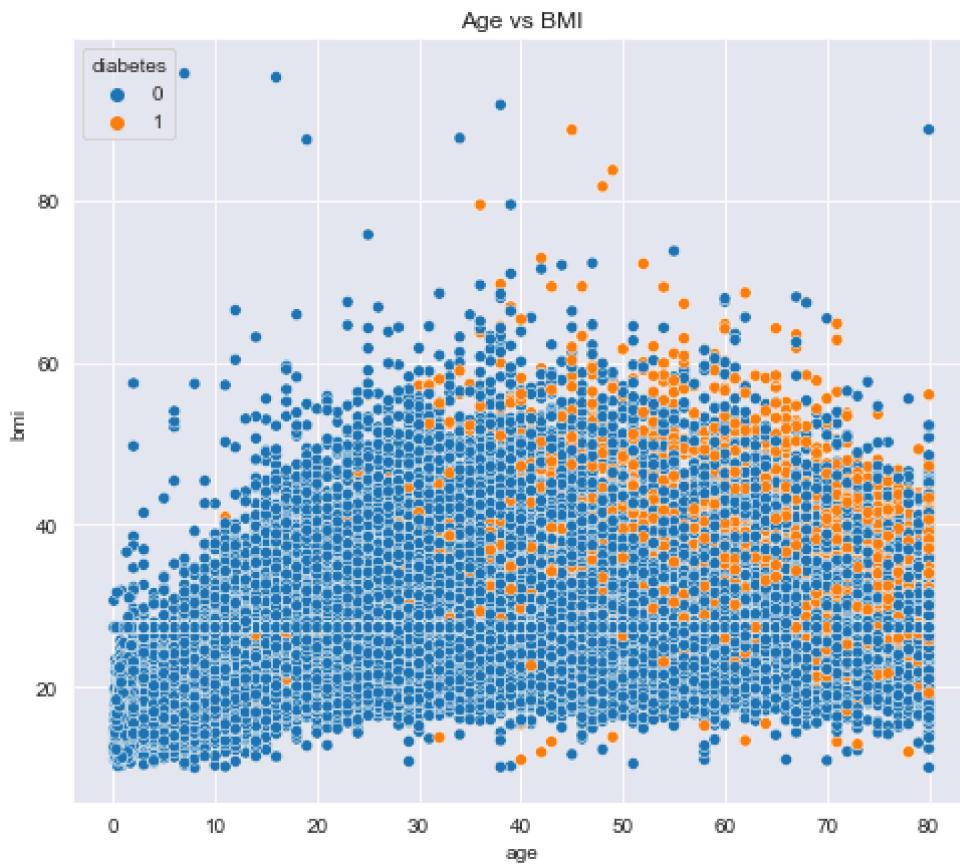
- There are 2 outliers in HbA1C level. I will keep the outliers as they can be helpful during modelling.

```
In [28]: box_plot(diabetes_df, "blood_glucose_level")
```



- There are 3 outliers in blood glucose level. These outliers are important in modelling. I will keep them as they will not have a huge impact to the models.

```
In [29]: # create scatterplot of Age vs BMI colored by Diabetes
fig, ax = plt.subplots(figsize=(8,7))
sns.scatterplot(x='age', y='bmi', hue='diabetes', data=df)
plt.title('Age vs BMI');
```



- We can see from the scatter plot that most people above 40 have hypertension. Younger people with high bmi do not have any diabetes.

```
In [30]: # Define a function to map the existing categories to new ones
def recategorize_smoking(smoking_status):
    if smoking_status in ['never', 'No Info']:
        return 'non-smoker'
    elif smoking_status == 'current':
        return 'current_smoker'
    elif smoking_status in ['ever', 'former', 'not current']:
        return 'past_smoker'

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

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

non-smoker	67276
past_smoker	19655
current_smoker	9197
Name: smoking_history, dtype:	int64

```
In [31]: # check for correlation
diabetes_df[['age', 'bmi', 'HbA1c_level', 'blood_glucose_level']].corr()
```

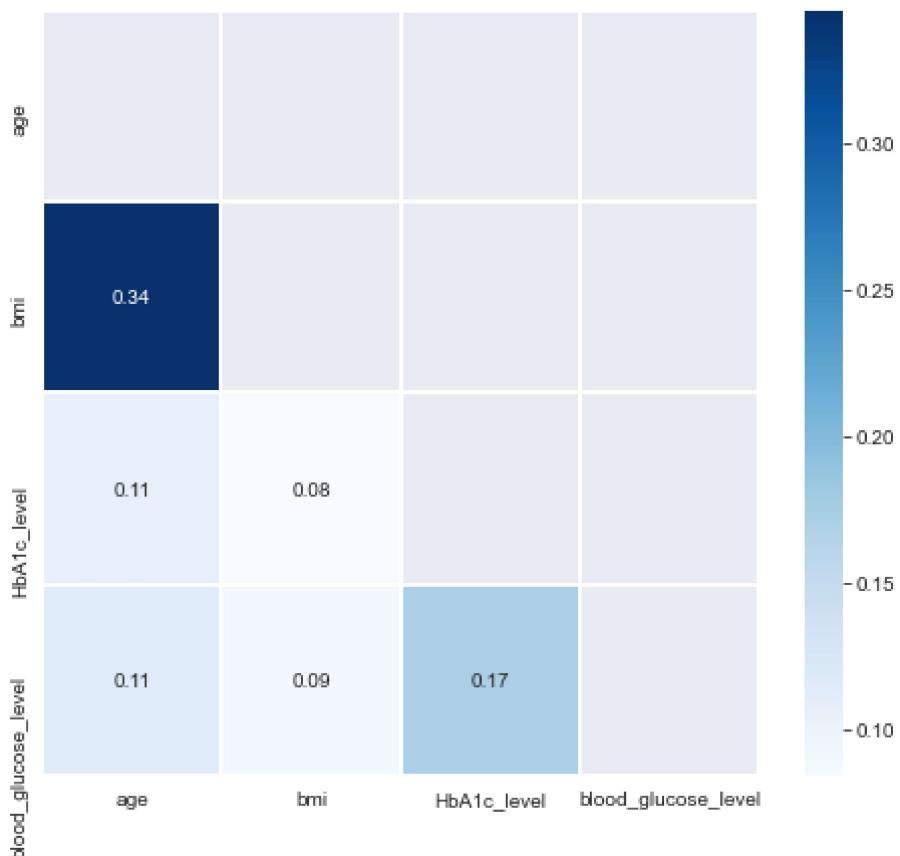
Out[31]:

	age	bmi	HbA1c_level	blood_glucose_level	index
age	1.000000	0.344779	0.106708	0.114323	
bmi	0.344779	1.000000	0.084443	0.092593	
HbA1c_level	0.106708	0.084443	1.000000	0.171717	
blood_glucose_level	0.114323	0.092593	0.171717	1.000000	

In [92]:

```
# heatmap of the correlations
fig, ax = plt.subplots(figsize=(8,7))
corr=diabetes_df[['age', 'bmi', 'HbA1c_level', 'blood_glucose_level']].corr()

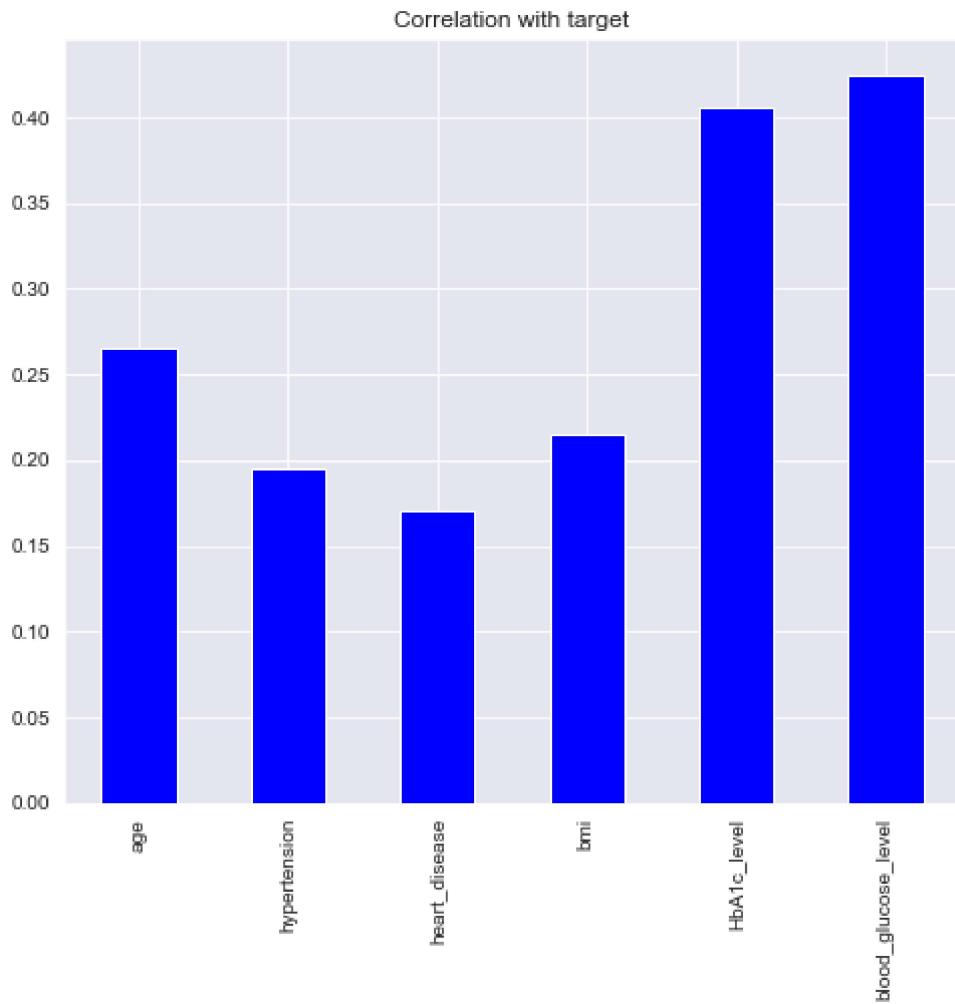
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr, mask=mask, linewidths=1, annot=True, fmt = ".2f", cmap="Blues");
```



- We can see from the above that there is **no multicollinearity** between the predictors

In [33]:

```
# correlations with target
diabetes_df.drop('diabetes', axis=1).corrwith(diabetes_df["diabetes"]).plot(kind='bar')
```



- We can see that **blood glucose level** and **HbA1c level** have a slightly **higher** correlation with the target.

Data preprocessing

In this section I will prepare the data for modelling. This will involve encoding categorical variables and also scaling the features.

```
In [34]: # we first split the data into X and y
X = diabetes_df.drop("diabetes", axis=1)
y = diabetes_df["diabetes"]

# split the data using a test size of 30%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# check the shape of the split data
len(X_train), len(X_test), len(y_train), len(y_test)
```

Out[34]: (67289, 28839, 67289, 28839)

- Splitting the data into train and test sets will help **prevent leakage** during preproceessing.
- The **test** set will eventually be used to **evaluate** how the **model performs**.

```
In [35]: # check the categorical variables to encode
# we saw we have 2 categorical variables, gender and smoking history
```

```

print(diabetes_df['gender'].value_counts())
print(".....")
print(diabetes_df['smoking_history'].value_counts())

```

	index
Female	56161
Male	39967
Name: gender, dtype: int64	
.....	
non-smoker	67276
past_smoker	19655
current_smoker	9197
Name: smoking_history, dtype: int64	

- I can see we have 2 categories in gender and 3 categories in smoking history. I will encode gender with pandas get dummies and onehot encoder for smoking history.

In [36]: `X_train = pd.get_dummies(X_train, columns=['gender'], sparse=False, drop_first=True)`
X_train

Out[36]:

	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	...
49226	43.0	0	0	non-smoker	27.32	6.5	100	...
30774	47.0	0	0	non-smoker	24.90	6.6	85	...
92991	33.0	0	0	past_smoker	40.08	6.2	200	...
39196	21.0	0	0	non-smoker	29.26	6.2	160	...
69480	59.0	0	0	past_smoker	30.93	6.1	85	...
...
6284	12.0	0	0	non-smoker	20.56	6.0	160	...
56192	31.0	0	0	past_smoker	24.34	4.5	85	...
79371	69.0	0	0	non-smoker	28.71	6.0	90	...
860	37.0	0	0	non-smoker	24.96	6.2	158	...
15912	23.0	0	0	non-smoker	27.32	5.8	85	...

67289 rows × 8 columns

In [40]: `smoking_train = X_train[['smoking_history']]`
Instantiate a OneHotEncoder
`ohe = OneHotEncoder(categories="auto", sparse=False, handle_unknown='ignore')`
(3) Fit the encoder on fireplace_qu_train
`ohe.fit(smoking_train)`
Inspect the categories of the fitted encoder
`ohe.categories_`

Out[40]: `[array(['current_smoker', 'non-smoker', 'past_smoker'], dtype=object)]`

In [41]: `smoking_encoded_train = ohe.transform(smoking_train)`
Visually inspect smoking_encoded_train
`smoking_encoded_train`

```
Out[41]: array([[0., 1., 0.],
   [0., 1., 0.],
   [0., 0., 1.],
   ...,
   [0., 1., 0.],
   [0., 1., 0.],
   [0., 1., 0.]])
```

```
In [42]: smoking_encoded_train = pd.DataFrame(
    # Pass in NumPy array
    smoking_encoded_train,
    # Set the column names to the categories found by OHE
    columns=ohe.categories_[0],
    # Set the index to match X_train's index
    index=X_train.index
)

# Visually inspect new dataframe
smoking_encoded_train
```

	current_smoker	non-smoker	past_smoker
49226	0.0	1.0	0.0
30774	0.0	1.0	0.0
92991	0.0	0.0	1.0
39196	0.0	1.0	0.0
69480	0.0	0.0	1.0
...
6284	0.0	1.0	0.0
56192	0.0	0.0	1.0
79371	0.0	1.0	0.0
860	0.0	1.0	0.0
15912	0.0	1.0	0.0

67289 rows × 3 columns

```
In [43]: X_train.drop("smoking_history", axis=1, inplace=True)

# Visually inspect X_train
X_train
```

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	gender_Male
49226	43.0	0	0	27.32	6.5	100	1
30774	47.0	0	0	24.90	6.6	85	1
92991	33.0	0	0	40.08	6.2	200	0
39196	21.0	0	0	29.26	6.2	160	0
69480	59.0	0	0	30.93	6.1	85	0
...
6284	12.0	0	0	20.56	6.0	160	0
56192	31.0	0	0	24.34	4.5	85	0

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	gender_Male
79371	69.0	0	0	28.71	6.0	90	0
860	37.0	0	0	24.96	6.2	158	0
15912	23.0	0	0	27.32	5.8	85	1

67289 rows × 7 columns

```
In [44]: X_train = pd.concat([X_train, smoking_encoded_train], axis=1)

# Visually inspect X_train
X_train
```

Out[44]:

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	gender_Male	curr
49226	43.0	0	0	27.32	6.5	100	1	
30774	47.0	0	0	24.90	6.6	85	1	
92991	33.0	0	0	40.08	6.2	200	0	
39196	21.0	0	0	29.26	6.2	160	0	
69480	59.0	0	0	30.93	6.1	85	0	
...
6284	12.0	0	0	20.56	6.0	160	0	
56192	31.0	0	0	24.34	4.5	85	0	
79371	69.0	0	0	28.71	6.0	90	0	
860	37.0	0	0	24.96	6.2	158	0	
15912	23.0	0	0	27.32	5.8	85	1	

67289 rows × 10 columns



In [45]:

```
# perform preprocessing on test set
# Encode gender type
X_test = pd.get_dummies(X_test, columns=['gender'], sparse=False, drop_first=True)

# One-hot encode smoking history
smoking_test = X_test[['smoking_history']]
smoking_encoded_test = ohe.transform(smoking_test)
smoking_encoded_test = pd.DataFrame(
    smoking_encoded_test,
    columns=ohe.categories_[0],
    index=X_test.index
)
X_test.drop("smoking_history", axis=1, inplace=True)
X_test = pd.concat([X_test, smoking_encoded_test], axis=1)

# Visually inspect X_test
X_test
```

Out[45]:

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	gender_Male	curr
2547	5.0	0	0	15.14	4.5	90	0	
47513	26.0	0	0	18.63	4.0	85	0	
15486	5.0	0	0	14.64	5.0	158	0	
92933	77.0	0	1	29.84	4.0	90	1	
67275	23.0	0	0	21.05	6.5	130	1	
...
81429	21.0	0	0	21.84	3.5	130	0	
29022	42.0	0	0	27.79	4.8	158	0	
35324	70.0	0	0	27.32	6.2	80	1	
56258	19.0	0	0	26.81	4.8	90	0	
39705	31.0	0	0	27.32	6.2	145	0	

28839 rows × 10 columns

Fit a base model

A base model is fit which will help us identify if other models are performing better

In [46]:

```
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)
```

- I use **smote** to ensure there is balance in the data

In [90]:

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

Out[90]: LogisticRegression()

In [48]:

```
# predict for test set
test_preds = lr.predict(X_test)
accuracy = accuracy_score(y_test, test_preds)
accuracy
```

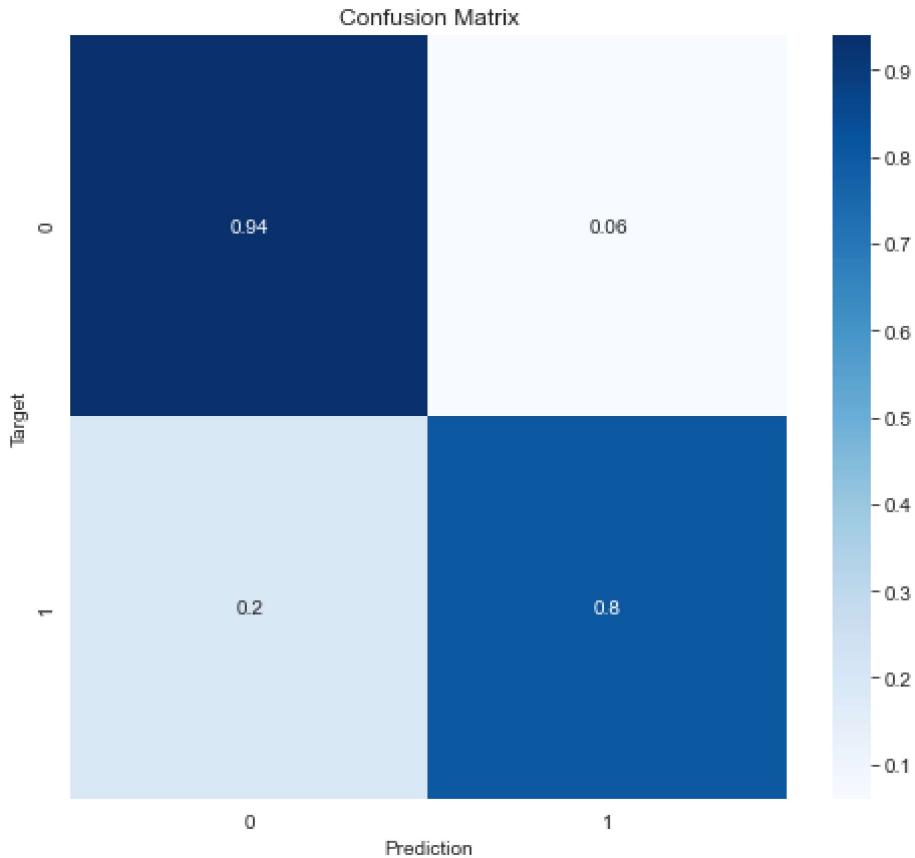
Out[48]: 0.8795034501889802

- The **base model** has an accuracy of around 87% on the test set. More models will be fit to find an improvement in performance in the test set.

In [93]:

```
def plot_confusion(tests, preds):
    fig, ax = plt.subplots(figsize=(8,7))
    cf = confusion_matrix(tests, preds, normalize='true')
    sns.heatmap(cf, annot=True, cmap="Blues")
    plt.xlabel('Prediction')
    plt.ylabel('Target')
    plt.title('Confusion Matrix');
```

```
In [94]: plot_confusion(y_test, test_preds)
```



Below is an image showing the components of a confusion matrix, which are:

1. True Negative
2. False Positive
3. False Negative
4. True positive



```
In [51]: # scale the data for both train and test sets and see if we have an improvement in  
scaler = StandardScaler()  
X_train_scaled = scaler.fit_transform(X_train)  
X_test_scaled = scaler.transform(X_test)
```

```
In [52]: # inspect the scaled train features  
X_train_scaled
```

```
Out[52]: array([[-3.46676162e-01, -2.88640742e-01, -1.99767758e-01, ...,  
-3.55291237e-01, 7.80932543e-01, -6.18915398e-01],  
[-1.60691829e-01, -2.88640742e-01, -1.99767758e-01, ...,  
-3.55291237e-01, 7.80932543e-01, -6.18915398e-01],  
[-8.11636995e-01, -2.88640742e-01, -1.99767758e-01, ...,  
-3.55291237e-01, -1.45358536e+00, 1.84764099e+00],  
...,  
[ 1.06400987e+00, -2.88640742e-01, -1.99767758e-01, ...,  
-3.55291237e-01, -1.45358536e+00, 1.84764099e+00],  
[ 7.79511114e-01, 3.46451438e+00, 5.00581280e+00, ...,  
-3.55291237e-01, -9.59485689e-01, 1.30223265e+00],  
[-9.22789877e-01, -2.88640742e-01, -1.99767758e-01, ...,  
8.97873559e-01, 4.11026050e-03, -6.18915398e-01]])
```

```
In [53]: X_test_scaled
```

```
Out[53]: array([[-2.11352733, -0.28864074, -0.19976776, ..., -0.35529124,  
0.78093254, -0.6189154 ],  
[-1.13710958, -0.28864074, -0.19976776, ..., -0.35529124,  
0.78093254, -0.6189154 ],  
[-2.11352733, -0.28864074, -0.19976776, ..., -0.35529124,  
0.78093254, -0.6189154 ],  
...,  
[ 0.90871809, -0.28864074, -0.19976776, ..., -0.35529124,  
-1.45358536, 1.84764099],  
[-1.46258216, -0.28864074, -0.19976776, ..., -0.35529124,  
0.78093254, -0.6189154 ],  
[-0.90462916, -0.28864074, -0.19976776, ..., -0.35529124,  
0.78093254, -0.6189154 ]])
```

With our features scaled we can apply more classification algorithms and tune the models to see if we achieve a higher prediction accuracy.

In [54]:

```
# create a function to train and evaluate a model

def train_and_evaluate(ModelClass, **params):
    """
    Trains a model and evaluates its performance on train and test data.

    Parameters:
    - ModelClass: The class of the machine learning model (e.g., LogisticRegression)
    - params: Parameters to initialize the model.

    Returns:
    - A string reporting training and test accuracy.
    """

    model = ModelClass(**params).fit(X_train_scaled, y_train)

    # Compute train predictions and accuracy
    train_preds = model.predict(X_train_scaled)
    train_acc = accuracy_score(y_train, train_preds)

    # compute test predictions and accuracy
    test_predictions = model.predict(X_test_scaled)
    test_acc = accuracy_score(y_test, test_predictions)

    print(f"Evaluation of model:")

    print(classification_report(y_test, test_predictions))

    return f"Train prediction accuracy: {train_acc:.2f} Test prediction accuracy: {t
```

In [55]:

```
# Define the parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100], # Regularization strength
    'solver': ['liblinear', 'lbfgs'], # Solvers
    'penalty': ['l2', 'l1'] # Regularization penalty
}

# Initialize Logistic Regression
log_reg = LogisticRegression(max_iter=1000)

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=log_reg,
    param_grid=param_grid,

    cv=5, # 5-fold cross-validation
    verbose=1,
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done 26 tasks | elapsed: 37.9s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 1.9min finished

Out[55]: GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=1000), n_jobs=-1, param_grid={'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l2', 'l1'], 'solver': ['liblinear', 'lbfgs']}, verbose=1)

In [56]:

```
# Get the best parameters and model
best_params = grid_search.best_params_
```

```
best_params
```

```
Out[56]: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
```

```
In [57]: best_score = grid_search.best_score_
best_score
```

```
Out[57]: 0.889345063538612
```

```
In [58]: lr_tuned = train_and_evaluate(LogisticRegression, C= 0.01, penalty= 'l2', solver= 'l
lr_tuned
```

Evaluation of model:				
	precision	recall	f1-score	support
0	0.98	0.88	0.93	26266
1	0.42	0.85	0.56	2573
accuracy			0.88	28839
macro avg	0.70	0.87	0.74	28839
weighted avg	0.93	0.88	0.90	28839

```
Out[58]: 'Train prediction accuracy: 0.89 Test prediction accuracy: 0.88'
```

Fiting decision trees

We will fit decision trees to check whether our model improves.

```
In [59]: # fit a simple dt with no tuned hyperparameters
```

```
dt = train_and_evaluate(DecisionTreeClassifier)
dt
```

Evaluation of model:				
	precision	recall	f1-score	support
0	0.97	0.97	0.97	26266
1	0.68	0.74	0.71	2573
accuracy			0.95	28839
macro avg	0.83	0.85	0.84	28839
weighted avg	0.95	0.95	0.95	28839

```
Out[59]: 'Train prediction accuracy: 1.00 Test prediction accuracy: 0.95'
```

We get a test prediction accuracy of 95% in the test set which is less than our base model.

We can also see that the model is overfitting in the train set with prediction accuracy of 100%.

We will use grid search to find the best parameters for the decision tree.

```
In [60]: dtc = DecisionTreeClassifier()
```

```
parameters = {
    'criterion':['gini','entropy'],
    'max_depth':range(2,10,1),
    'min_samples_leaf':range(1,5),
    'min_samples_split':range(2,5),
    #'splitter':['best', 'random']
}
```

```
grid_search_dt = GridSearchCV(dtc, parameters, cv=5, n_jobs=-1, verbose=1)
grid_search_dt.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks    | elapsed:   0.7s
[Parallel(n_jobs=-1)]: Done 176 tasks    | elapsed:   4.3s
[Parallel(n_jobs=-1)]: Done 426 tasks    | elapsed:  12.5s
[Parallel(n_jobs=-1)]: Done 776 tasks    | elapsed:  23.4s
[Parallel(n_jobs=-1)]: Done 960 out of 960 | elapsed:  30.4s finished
```

```
Out[60]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': range(2, 10),
                                  'min_samples_leaf': range(1, 5),
                                  'min_samples_split': range(2, 5)},
                      verbose=1)
```

```
In [61]: grid_search_dt.best_params_
```

```
Out[61]: {'criterion': 'gini',
          'max_depth': 9,
          'min_samples_leaf': 3,
          'min_samples_split': 2}
```

```
In [62]: grid_search_dt.best_score_
```

```
Out[62]: 0.9260101010101011
```

```
In [78]: dt_tuned = train_and_evaluate(DecisionTreeClassifier, criterion = 'gini', max_depth
                                    min_samples_split = 2)
dt_tuned
```

Evaluation of model:				
	precision	recall	f1-score	support
0	0.98	0.90	0.94	26266
1	0.46	0.84	0.60	2573
accuracy			0.90	28839
macro avg	0.72	0.87	0.77	28839
weighted avg	0.94	0.90	0.91	28839

```
Out[78]: 'Train prediction accuracy: 0.93 Test prediction accuracy: 0.90'
```

- We get a 90% accuracy which is an improvement from our base model.

```
In [79]: # fit the decision tree manually to plot the tree
dt = DecisionTreeClassifier(criterion = 'gini', max_depth = 9, min_samples_leaf = 3,
                            min_samples_split = 2)

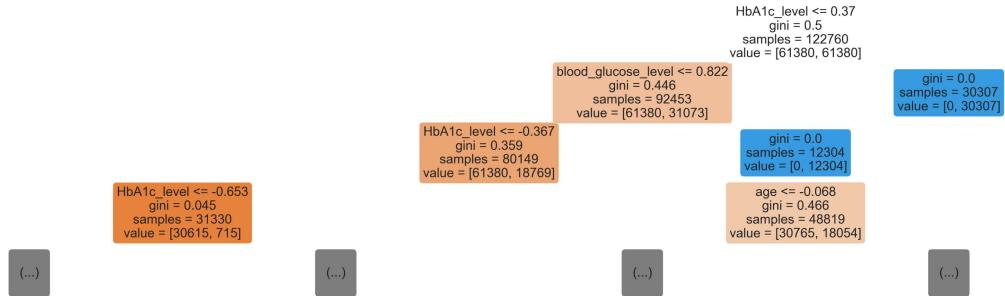
dt.fit(X_train_scaled, y_train)
```

```
Out[79]: DecisionTreeClassifier(max_depth=9, min_samples_leaf=3)
```

```
In [80]: dt.classes_
```

```
Out[80]: array([0, 1], dtype=int64)
```

```
In [81]: plt.figure(figsize=(80, 20))
plot_tree(dt, feature_names= X_train.columns, filled= True, rounded= True, max_depth
```



- From the tree we can clearly **visualize** how the splits are happening.
- We start with **HbA1c level** as the most important factor used to split based on the gini value of 0.5.

Advantages of the decision trees over our base logistic regression model

- Decision trees can handle **non linear relationships** which can help capture complex patterns in the data.
- Decision trees provide a clear indication of which features are most important for prediction.

Fit an ensemble model: Random forest

Random forests is an ensemble model which is based on bagging. Bagging is where multiple models are trained on different subsets of the training data. In this case with random forest, multiple decision trees are trained during the training phase. This can help reduce the risk of overfitting and improves predictions.

The random forest will help me decide whether to use the decision trees or use the random forest as my final model.

```
In [67]: # fit a random forest without tuning
rf = train_and_evaluate(RandomForestClassifier)
rf
```

Evaluation of model:				
	precision	recall	f1-score	support
0	0.97	0.98	0.98	26266
1	0.82	0.70	0.76	2573
accuracy			0.96	28839
macro avg	0.89	0.84	0.87	28839
weighted avg	0.96	0.96	0.96	28839

```
Out[67]: 'Train prediction accuracy: 1.00 Test prediction accuracy: 0.96'
```

- I can see a **prediction accuracy** of about 96%. I will try using **hyperparameters** to find the best fit.

```
In [68]: rfc = RandomForestClassifier()
parameters = {
```

```

    'criterion':['gini','entropy'],
    'max_depth':range(2,10,1),
    'min_samples_leaf':range(1,5,1),
}

grid_search_rf = GridSearchCV(rfc, parameters, cv=3, n_jobs=-1, verbose=1)
grid_search_rf.fit(X_train, y_train)

```

Fitting 3 folds for each of 64 candidates, totalling 192 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done 26 tasks | elapsed: 16.8s
[Parallel(n_jobs=-1)]: Done 192 out of 192 | elapsed: 2.3min finished

Out[68]: GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
param_grid={'criterion': ['gini', 'entropy'],
'max_depth': range(2, 10),
'min_samples_leaf': range(1, 5)},
verbose=1)

In [69]: grid_search_rf.best_params_

Out[69]: {'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 2}

In [70]: grid_search_rf.best_score_

Out[70]: 0.9379846855653308

In [82]: rfc = train_and_evaluate(RandomForestClassifier, criterion = 'gini', max_depth = 9,
rfc

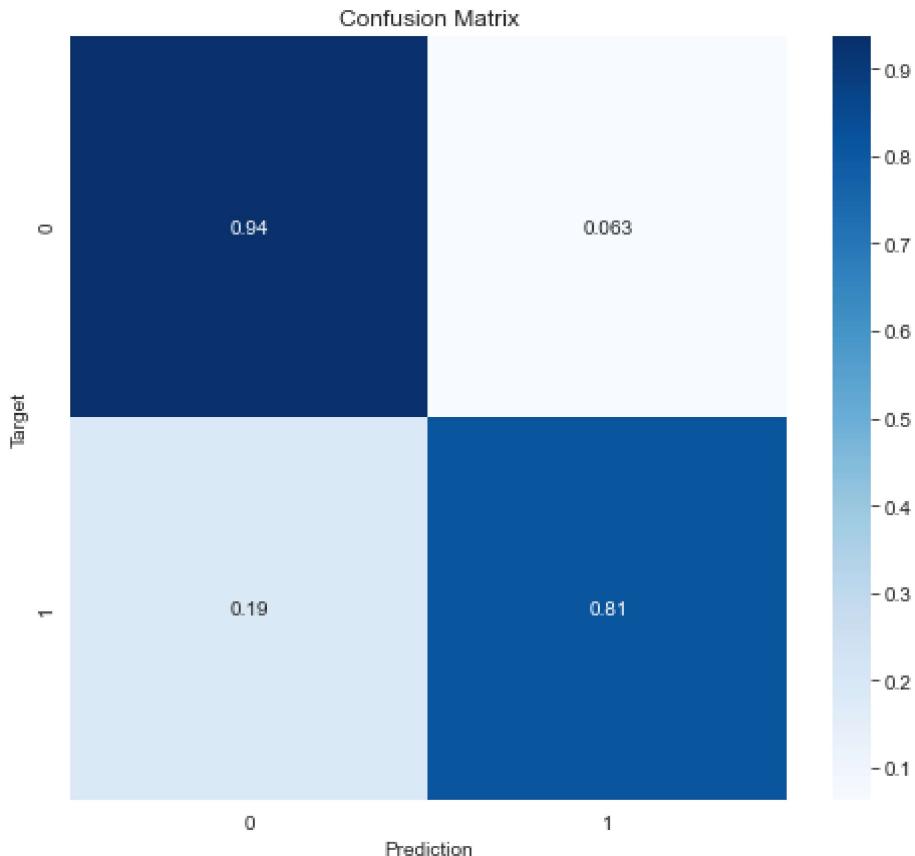
Evaluation of model:				
	precision	recall	f1-score	support
0	0.98	0.93	0.96	26266
1	0.55	0.81	0.65	2573
accuracy			0.92	28839
macro avg	0.76	0.87	0.81	28839
weighted avg	0.94	0.92	0.93	28839

Out[82]: 'Train prediction accuracy: 0.94 Test prediction accuracy: 0.92'

- I see a change in the prediction accuracy. I will go with the random forest with tuned parameters as the best model for predicting whether a patient has diabetes or not.
- Our trained random forest model achieved an accuracy of around 92%. This indicates that the model correctly classified around 92% of all cases in the test set.
- The model has a **precision** of (0.98) for class 0, meaning that among all instances where the model predicted non-diabetes, 98% were indeed non-diabetes. The **recall** for class 0 is also high (0.93). This means that our model correctly identified 93% of all actual non-diabetes cases in the dataset.
- The **precision** for class 1 is around (0.55), which indicates that when the model predicted diabetes, it was correct around 55% of the time. However, the **recall** is low around (0.81). This means that our model was able to capture around 81% of all actual diabetes cases.

In [95]: rf = RandomForestClassifier(criterion = 'gini', max_depth = 9, min_samples_leaf = 2)
rf.fit(X_train_scaled, y_train)

```
test_preds = rf.predict(X_test_scaled)
plot_confusion(y_test, test_preds)
```



In [89]: `## check the most important features for the random forest
rf.feature_importances_`

Out[89]: `array([1.41423647e-01, 3.70586676e-04, 1.84102573e-04, 5.19592951e-02,
 4.20181438e-01, 2.66202343e-01, 3.19249047e-03, 1.41629540e-02,
 5.92982526e-02, 4.30248907e-02])`

In [88]: `X_train.columns`

Out[88]: `Index(['age', 'hypertension', 'heart_disease', 'bmi', 'HbA1c_level',
 'blood_glucose_level', 'gender_Male', 'current_smoker', 'non-smoker',
 'past_smoker'],
 dtype='object')`

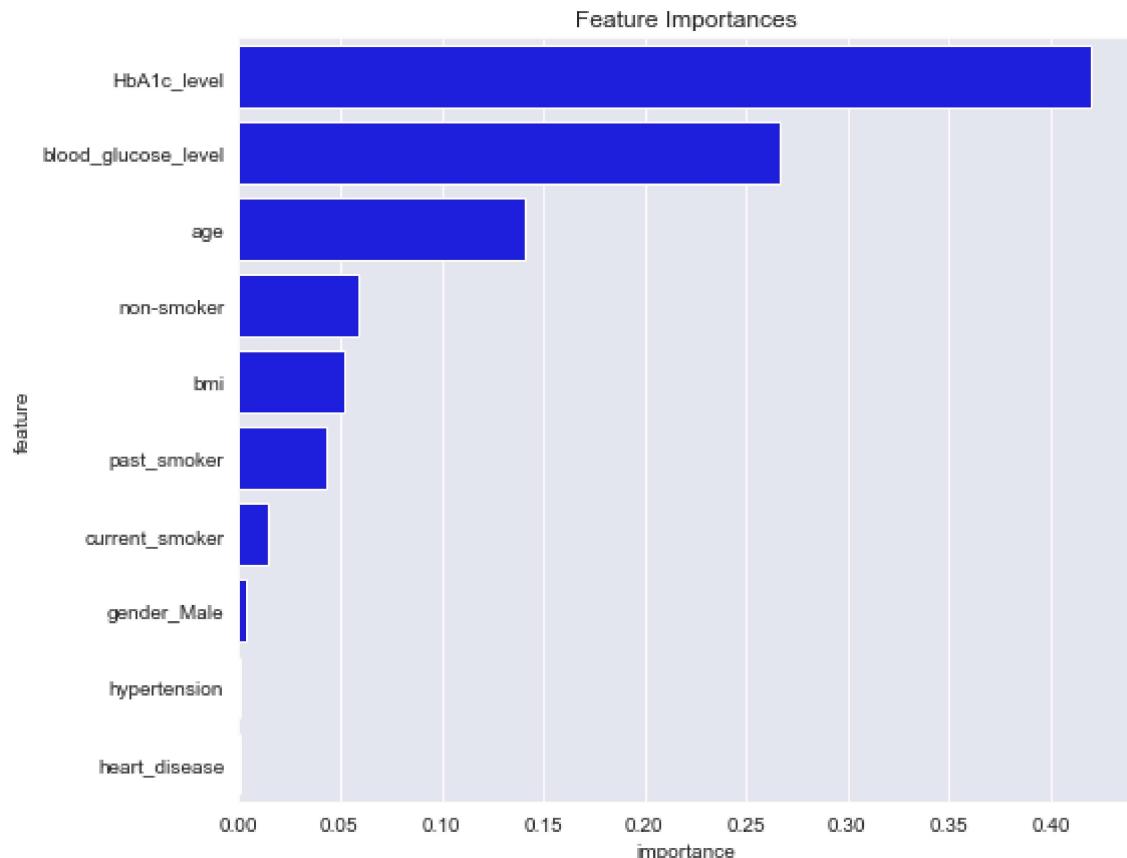
In [86]: `importance_df = pd.DataFrame({
 'feature': X_train.columns,
 'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)
importance_df`

Out[86]:

	feature	importance
4	HbA1c_level	0.420181
5	blood_glucose_level	0.266202
0	age	0.141424
8	non-smoker	0.059298
3	bmi	0.051959
9	past_smoker	0.043025

	feature	importance
7	current_smoker	0.014163
6	gender_Male	0.003192
1	hypertension	0.000371
2	heart_disease	0.000184

```
In [96]: # Plot the feature importances
fig, ax = plt.subplots(figsize=(8, 7))
sns.barplot(x='importance', y='feature', data=importance_df, color='blue')
plt.title('Feature Importances');
```



The feature importance results provide insight into which features are most influential in predicting diabetes using our decision tree model. The importance of a feature is calculated based on how much the tree nodes that use that feature reduce impurity across all trees in the forest.

We can see that:

- **HbA1c_level** is the most important feature with an importance of 0.42. HbA1c is a measure of the average levels of blood glucose over the past 2 to 3 months, so it is a significant predictor of diabetes.
- The **blood_glucose_level** the second most important feature with an importance of 0.26. This aligns with medical knowledge, as blood glucose levels are directly used to diagnose diabetes.
- **Age** the third most important feature with an importance of 0.0154. It is known that the risk of type 2 diabetes increases as you get older.

- **Non smoker** and **BMI** are fourth and fifth. Body Mass Index is a key risk factor for diabetes, and its role is well documented in medical literature.
- Other factors show slight importances.

Conclusions

- The analysis employed a **random forest classifier** as the best model to predict diabetes based on various health indicators and lifestyle factors. The model was trained and evaluated on a dataset of 100,000 records, and hyperparameter tuning was performed to optimize the model's performance.
- The model achieved an **accuracy** of approximately 92%, with precision of 0.98 for class 0 (non-diabetic) and 0.55 for class 1 (diabetic). It was also able to recall 93% of non-diabetic cases and 81% of diabetic cases correctly. The relatively high accuracy and balanced performance on both classes indicate that the model is well-tuned and robust.
- **Feature importance** analysis highlighted HbA1c_level and blood_glucose_level as the most critical factors in predicting Diabetes. Age and BMI also showed significant importance. However, some features such as gender, had minimal or no impact on the model's predictions.

Recommendations

- Despite the model achieving a good accuracy of 92%, more data should be collected for patients with diabetes.
- Addition of more features which are used to create the models. More features can help improve recall, especially for cases where the patient has diabetes.

Suggestions for future improvement

- Collection of more data on things such as family history, physical activity level and diet information could help improve the model's predictive accuracy.
- Training could be done on other machine learning models. This can help in improving the accuracy of predictions.
- Feature engineering such as interaction terms and polynomial features could help improve the model's performance.