```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error, r2_score, roc_auc_score, roc_curve, classification_report

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import KFold
```

```
import warnings
warnings.simplefilter(action = "ignore")
```

```
# Load the data
data = pd.read_csv('/content/diabetes_prediction_dataset.csv')
df = data
```

```
# Display the first few rows and basic information about the dataset
print(data.head())
```

```
    gender   age  hypertension  heart_disease smoking_history    bmi  \
0  Female  80.0             0              1           never  25.19
1  Female  54.0             0              0         No Info  27.32
2    Male  28.0             0              0           never  27.32
3  Female  36.0             0              0         current  23.45
4    Male  76.0             1              1         current  20.14

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

```
print(data.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
None
```

```
# Check for missing values
print(data.isnull().sum())
```

```
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
```

```
# Encode categorical variables
df['gender'] = df['gender'].map({'Female': 0, 'Male': 1})
df['smoking_history'] = df['smoking_history'].astype('category').cat.codes
```

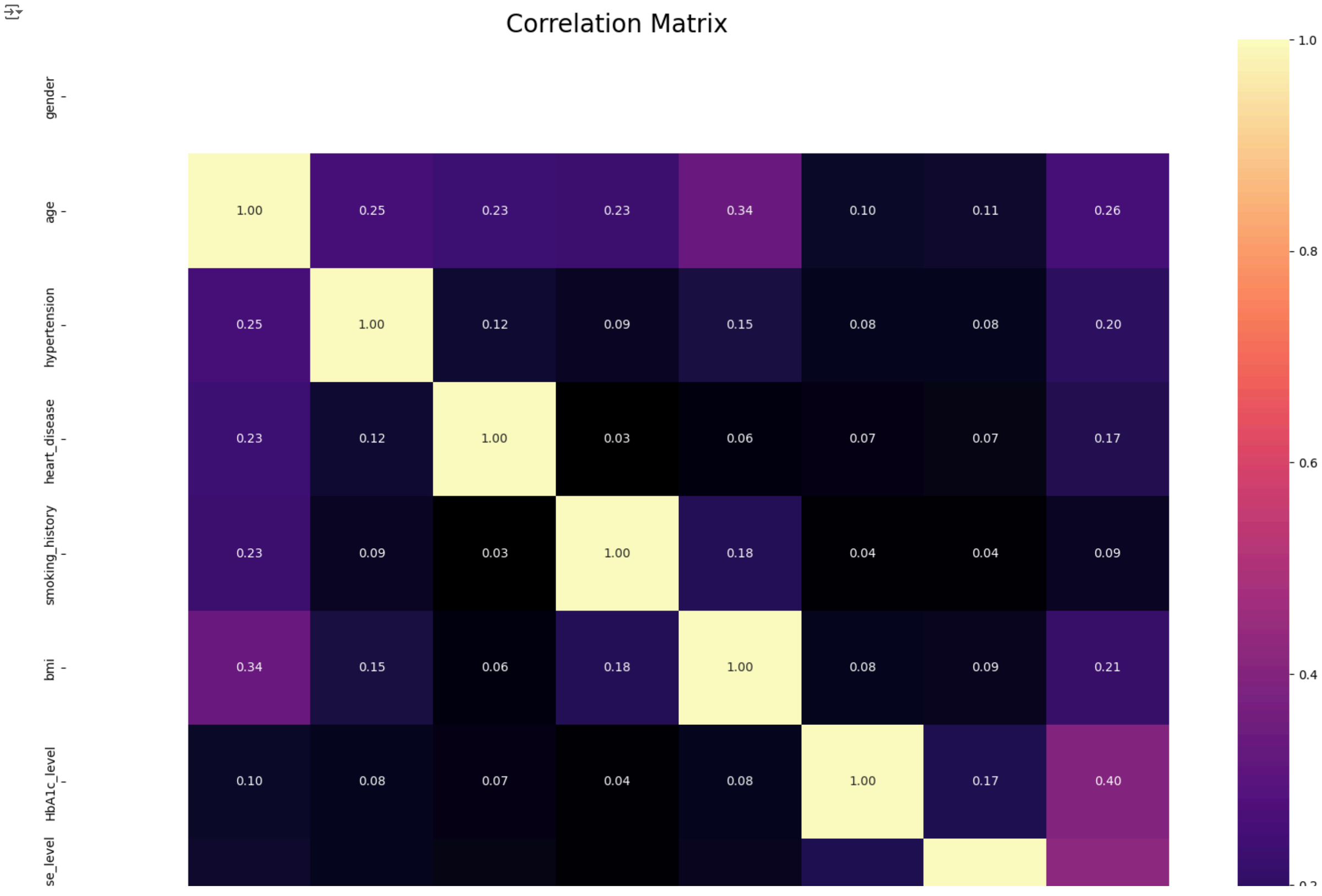
```
# Descriptive statistics of the data set
print(df.describe([0.10,0.25,0.50,0.75,0.90,0.95,0.99]).T)
```

```
                      count        mean        std    min    10%     25%  \
gender                  0.0         NaN        NaN    NaN    NaN     NaN
age                100000.0   41.885856  22.516840   0.08  10.00   24.00
hypertension       100000.0    0.074850   0.263150   0.00   0.00    0.00
heart_disease      100000.0    0.039420   0.194593   0.00   0.00    0.00
smoking_history    100000.0    2.179650   1.889659   0.00   0.00    0.00
bmi                100000.0   27.320767   6.636783  10.01  19.18   23.63
HbA1c_level        100000.0    5.527507   1.070672   3.50   4.00    4.80
blood_glucose_level 100000.0  138.058060  40.708136  80.00  85.00  100.00
diabetes           100000.0    0.085000   0.278883   0.00   0.00    0.00

                      50%     75%     90%     95%       99%     max
gender                NaN     NaN     NaN     NaN       NaN     NaN
age                 43.00   60.00   73.00   80.00   80.0000   80.00
hypertension         0.00    0.00    0.00    1.00    1.0000    1.00
heart_disease        0.00    0.00    0.00    0.00    1.0000    1.00
smoking_history      3.00    4.00    4.00    5.00    5.0000    5.00
bmi                 27.32   29.58   35.47   39.49   48.7901   95.69
HbA1c_level          5.80    6.20    6.60    6.60    8.8000    9.00
blood_glucose_level 140.00  159.00  200.00  200.00  280.0000  300.00
diabetes             0.00    0.00    0.00    1.00    1.0000    1.00
```

```
# Distribution of the diabetes variable
print(df["diabetes"].value_counts()*100/len(df))
```

```
diabetes
0    91.5
1     8.5
Name: count, dtype: float64
```

```
# Correlation matrix
plt.figure(figsize=[20,15])
sns.heatmap(df.corr(), annot=True, fmt=".2f", ax=plt.axes(), cmap="magma")
plt.title("Correlation Matrix", fontsize=20)
plt.show()
```


Correlation Matrix
gender
age
hypertension
heart_disease
smoking_history
bmi
HbA1c_level
se_level
1.00 0.25 0.23 0.23 0.34 0.10 0.11 0.26
0.25 1.00 0.12 0.09 0.15 0.08 0.08 0.20
0.23 0.12 1.00 0.03 0.06 0.07 0.07 0.17
0.23 0.09 0.03 1.00 0.18 0.04 0.04 0.09
0.34 0.15 0.06 0.18 1.00 0.08 0.09 0.21
0.10 0.08 0.07 0.04 0.08 1.00 0.17 0.40
1.0
0.8
0.6
0.4

```
# Data Preprocessing
X = df.drop("diabetes", axis=1)
y = df["diabetes"]
```

```
# Standardization
scaler = StandardScaler()
X = scaler.fit_transform(X)
X = pd.DataFrame(X, columns=df.columns.drop("diabetes"))
```

```
# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 42)
```

```
# Model training and evaluation
models = [
    ('LR', LogisticRegression(random_state=42)),
    ('KNN', KNeighborsClassifier()),
    ('CART', DecisionTreeClassifier(random_state=42)),
    ('RF', RandomForestClassifier(random_state=42)),
    ('SVM', SVC(gamma='auto', random_state=42)),
    ('XGB', XGBClassifier(random_state=42)),

]
```

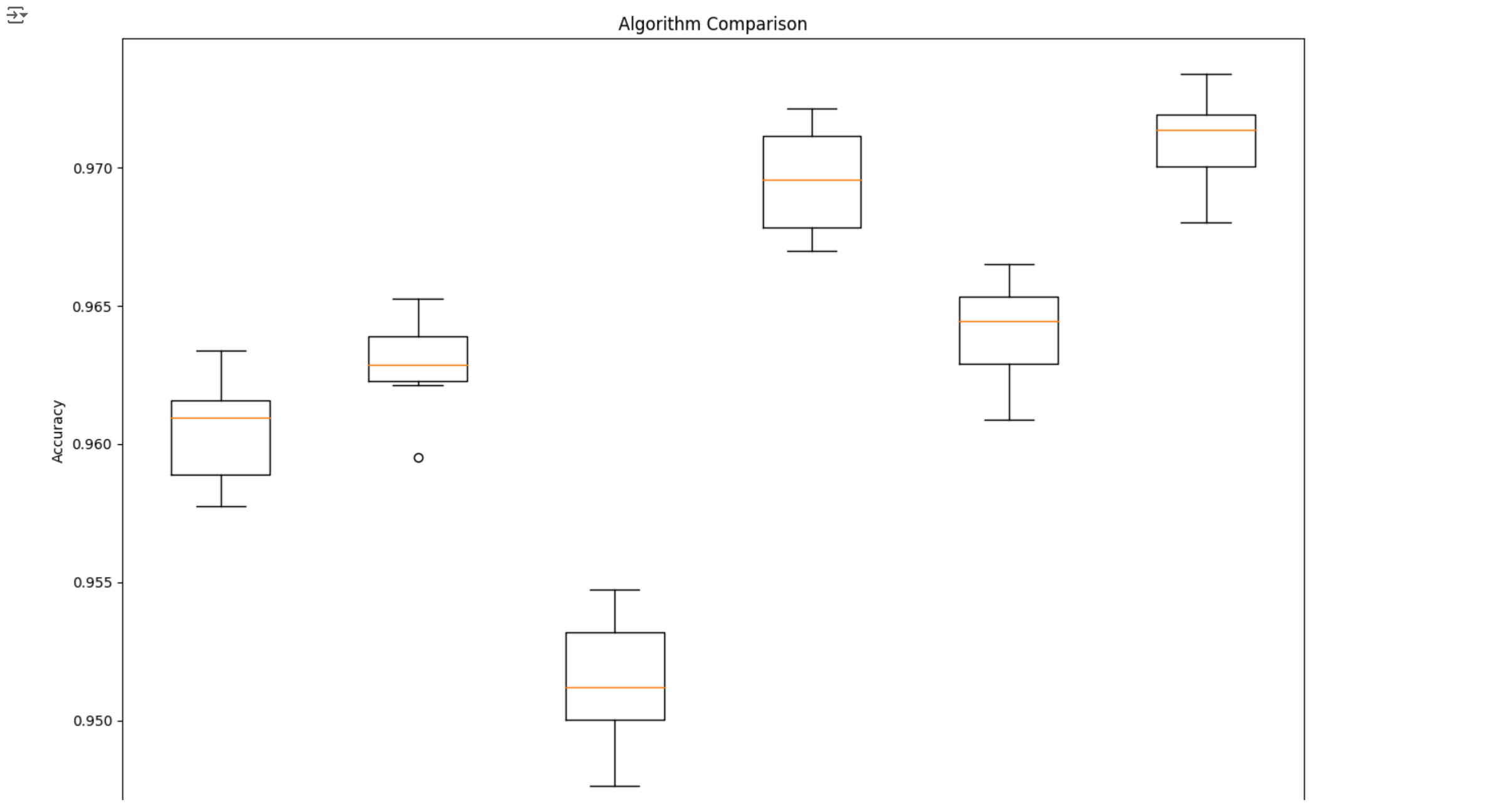
```
results = []
names = []
```

```
from sklearn.impute import SimpleImputer

# Create an imputer object with a strategy (mean, median, or most_frequent)
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and transform it
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)
```

```
for name, model in models:
    kfold = KFold(n_splits=10, random_state=42, shuffle=True)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    print(f"{name}: {cv_results.mean():.4f} ({cv_results.std():.4f})")
```

```
LR: 0.9605 (0.0018)
KNN: 0.9629 (0.0015)
CART: 0.9513 (0.0023)
RF: 0.9695 (0.0018)
SVM: 0.9641 (0.0017)
XGB: 0.9710 (0.0016)
```

```
# Boxplot algorithm comparison
plt.figure(figsize=(15,10))
plt.title('Algorithm Comparison')
plt.boxplot(results)
plt.xticks(range(1, len(names) + 1), names)
plt.ylabel('Accuracy')
plt.show()
```



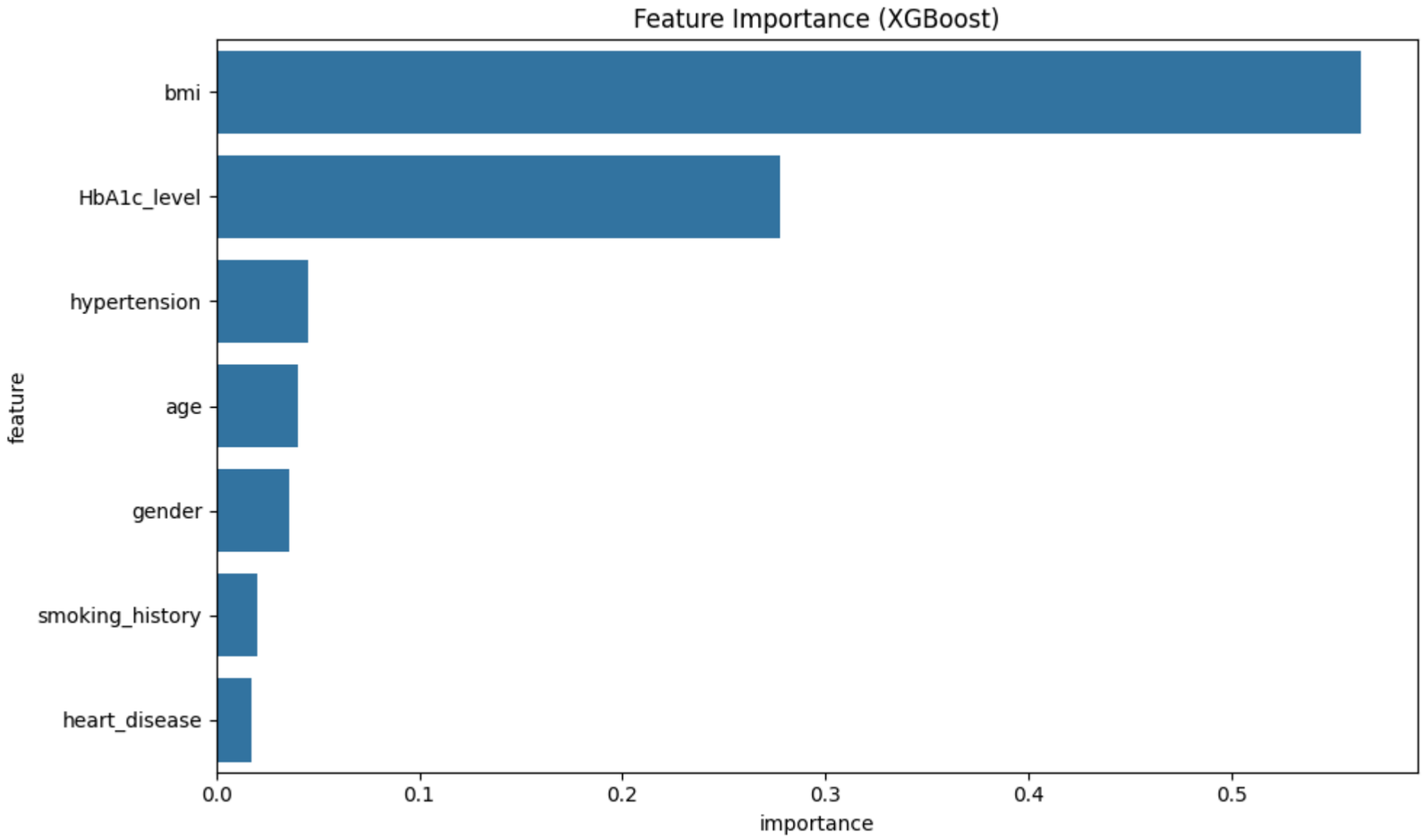
```
# XGBoost model
xgb = XGBClassifier(random_state=42)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
```

```
print("\nXGBoost Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
XGBoost Performance:
Accuracy: 0.9712

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     18292
           1       0.96      0.70      0.81      1708

    accuracy                           0.97     20000
   macro avg       0.96      0.85      0.89     20000
weighted avg       0.97      0.97      0.97     20000
```

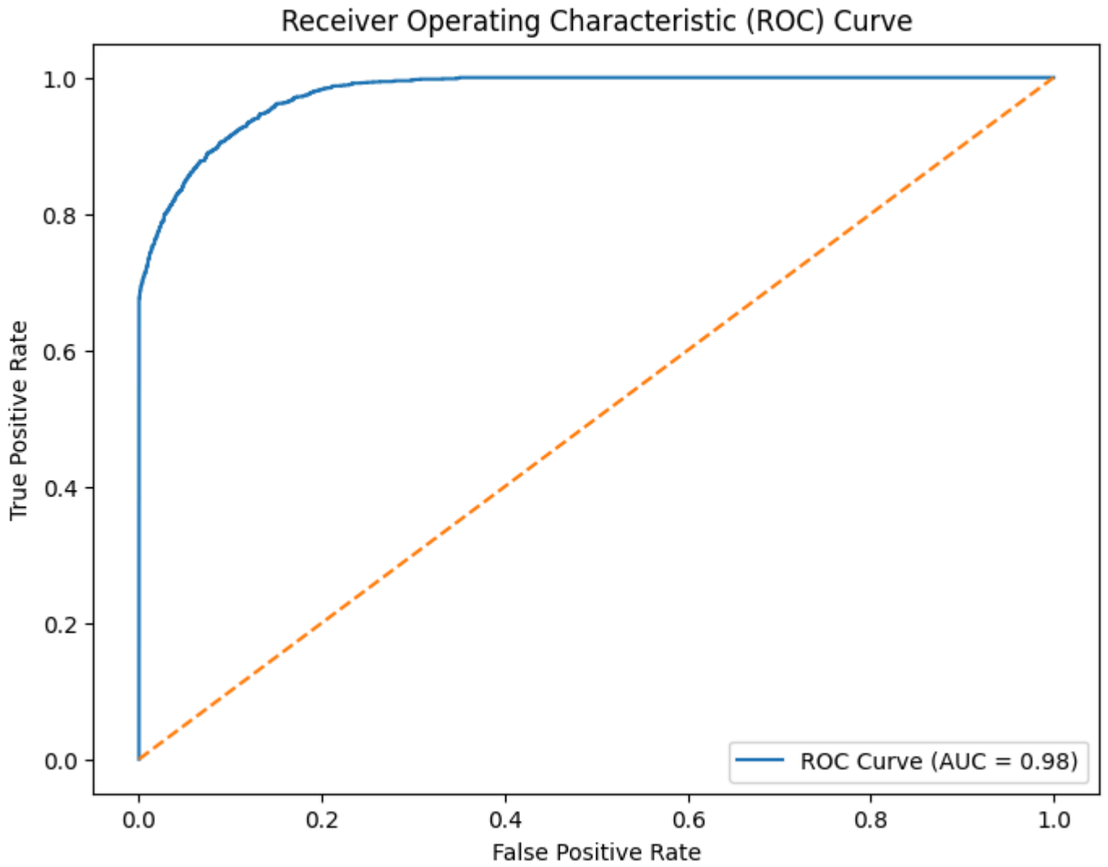
```
# Feature importance
feature_importance = xgb.feature_importances_
feature_importance_df = pd.DataFrame({'feature': X.columns[0:len(X.columns)-1], 'importance': feature_importance})
feature_importance_df = feature_importance_df.sort_values('importance', ascending=False)
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance_df)
plt.title('Feature Importance (XGBoost)')
plt.tight_layout()
plt.show()
```




```
# ROC Curve
y_pred_proba = xgb.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)
```

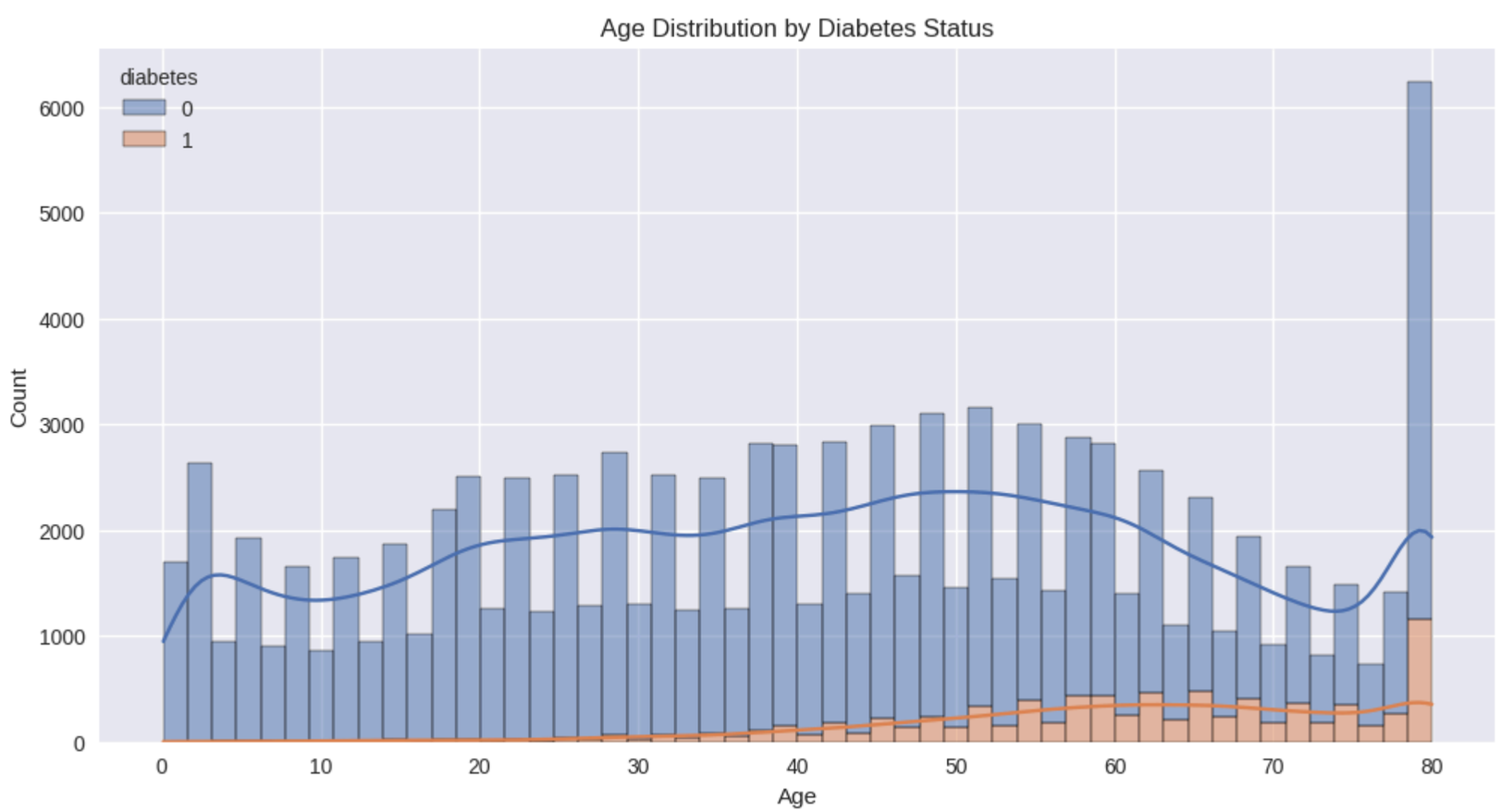
```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



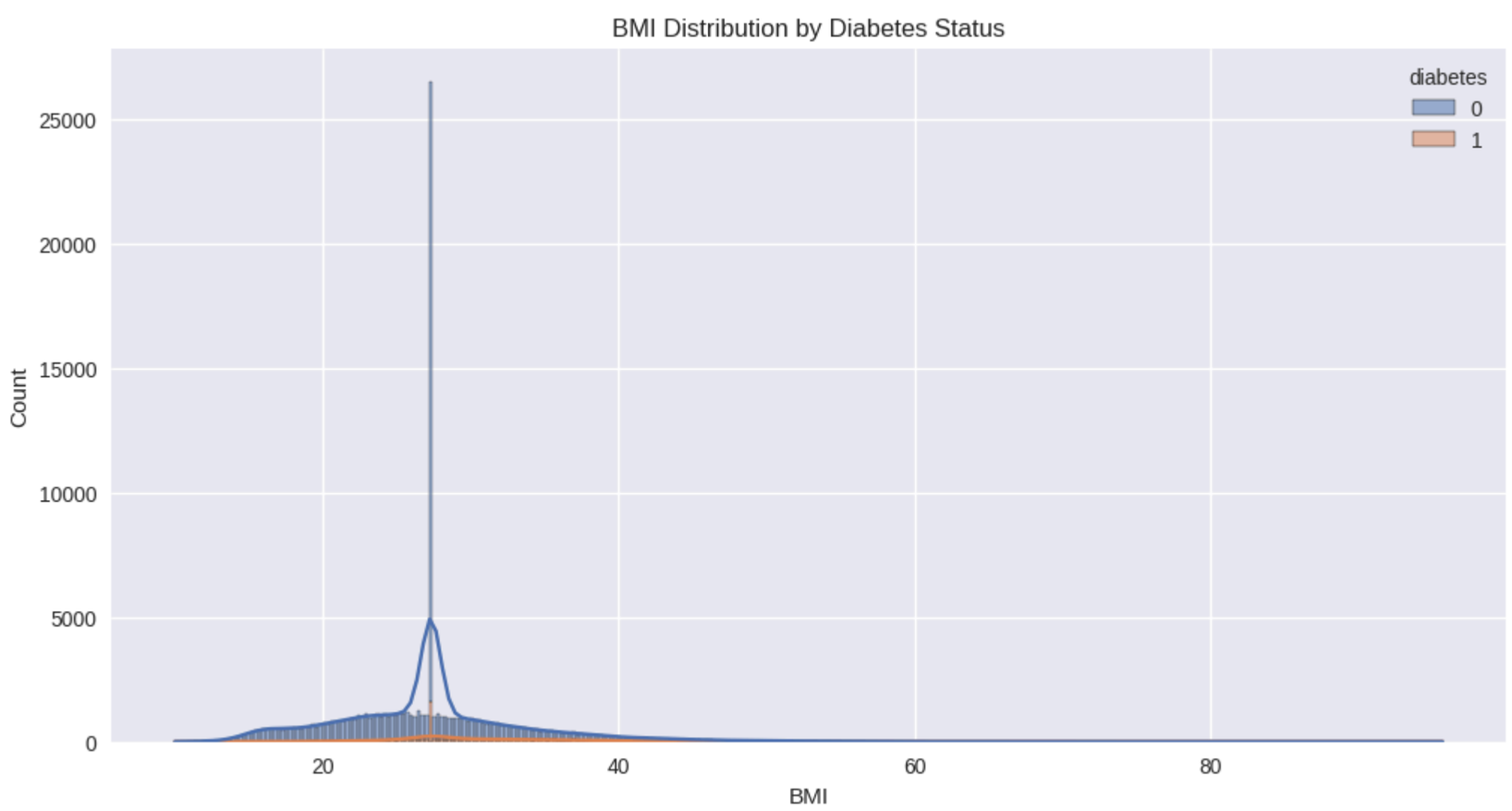

```
'''  oh shit '''
```

```
# Set up the plotting style
plt.style.use('seaborn')
sns.set_palette("deep")
```

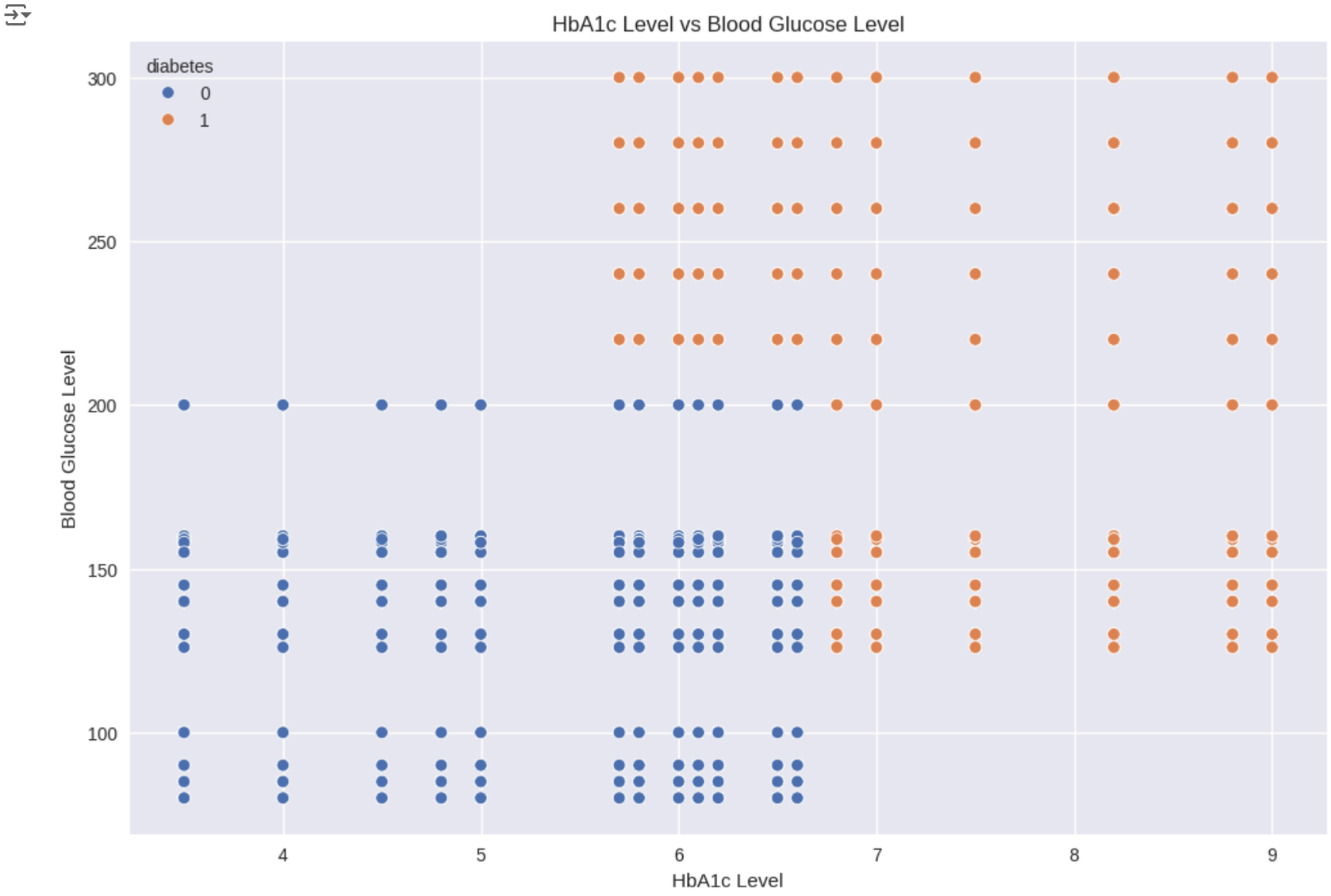
```
# 1. Age Distribution
plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='age', hue='diabetes', kde=True, multiple="stack")
plt.title('Age Distribution by Diabetes Status')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```




```
# 2. BMI Distribution
plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='bmi', hue='diabetes', kde=True, multiple="stack")
plt.title('BMI Distribution by Diabetes Status')
plt.xlabel('BMI')
plt.ylabel('Count')
plt.show()
```



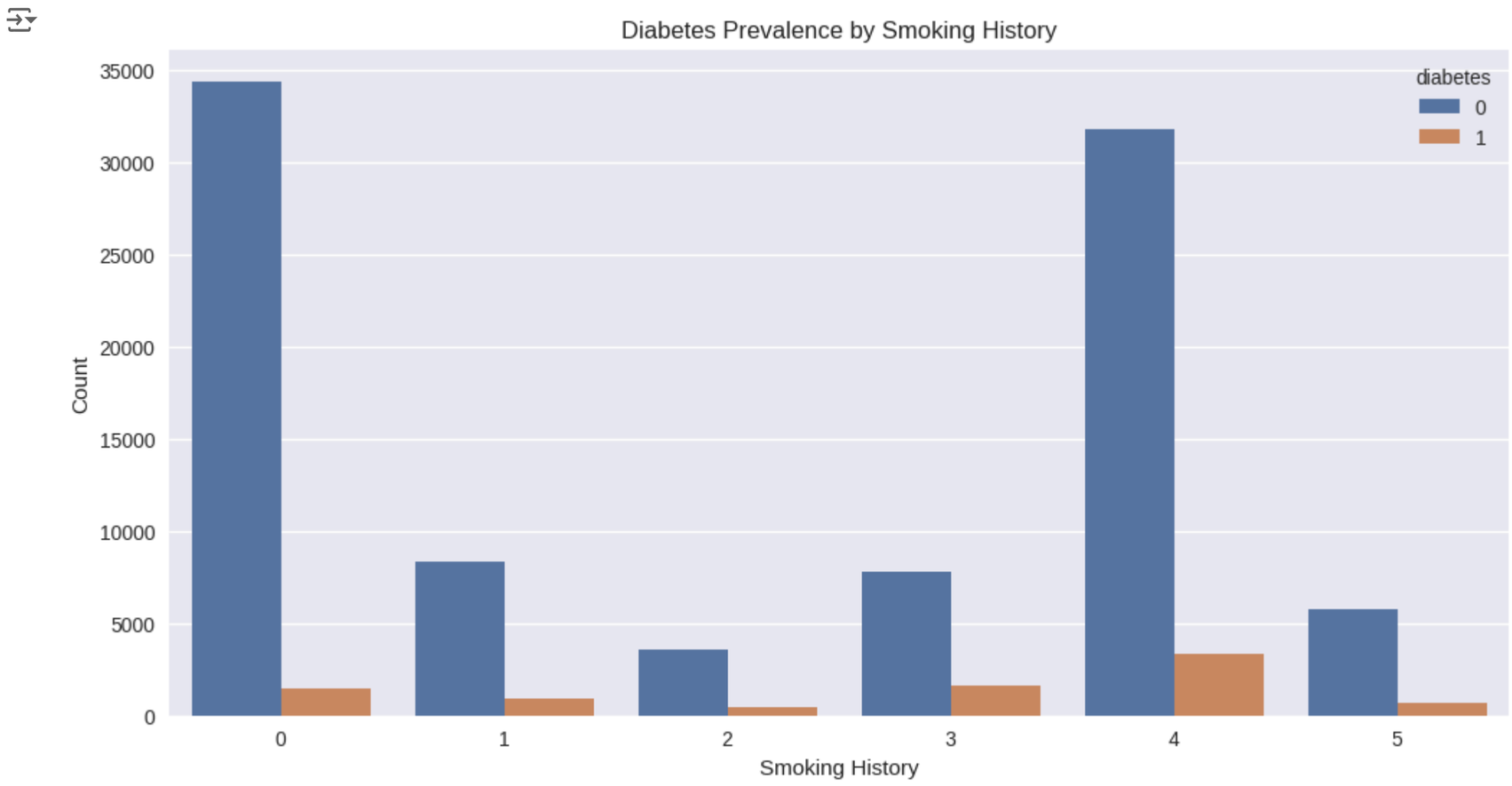

```
# 3. HbA1c Level vs Blood Glucose Level
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df, x='HbA1c_level', y='blood_glucose_level', hue='diabetes')
plt.title('HbA1c Level vs Blood Glucose Level')
plt.xlabel('HbA1c Level')
plt.ylabel('Blood Glucose Level')
plt.show()
```



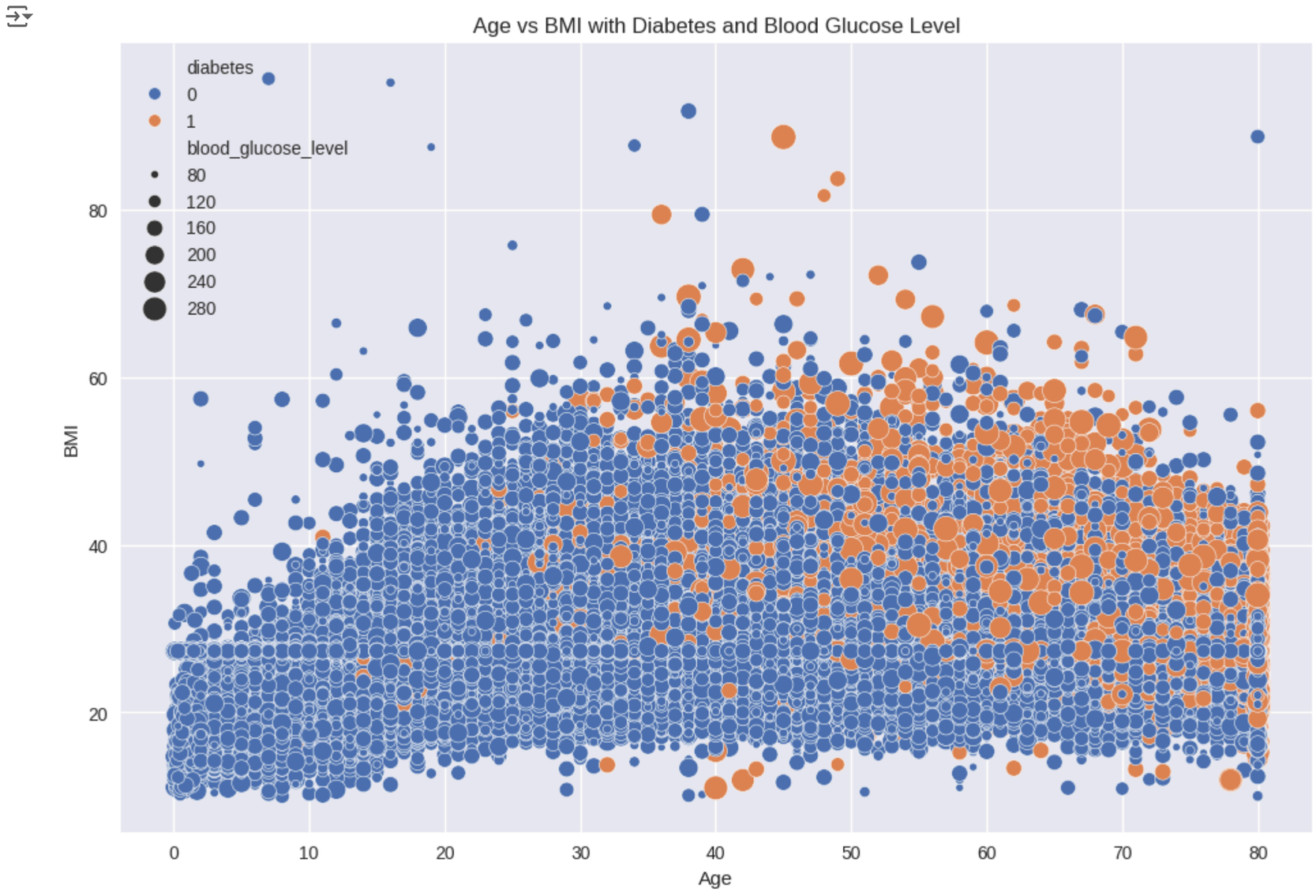

```
# 4. Gender and Diabetes
'''plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='gender', hue='diabetes')
plt.title('Diabetes Prevalence by Gender')
plt.xlabel('Gender (0: Female, 1: Male)')
plt.ylabel('Count')
plt.show()  '''
```

```
'plt.figure(figsize=(10, 6))\nsns.countplot(data=df, x='gender', hue='diabetes')\nplt.title('Diabetes Prevalence by Gender')\nplt.xlabel('Gender (0: Female, 1: Male)')\nplt.ylabel('Coun
t')\nplt.show()  '
```

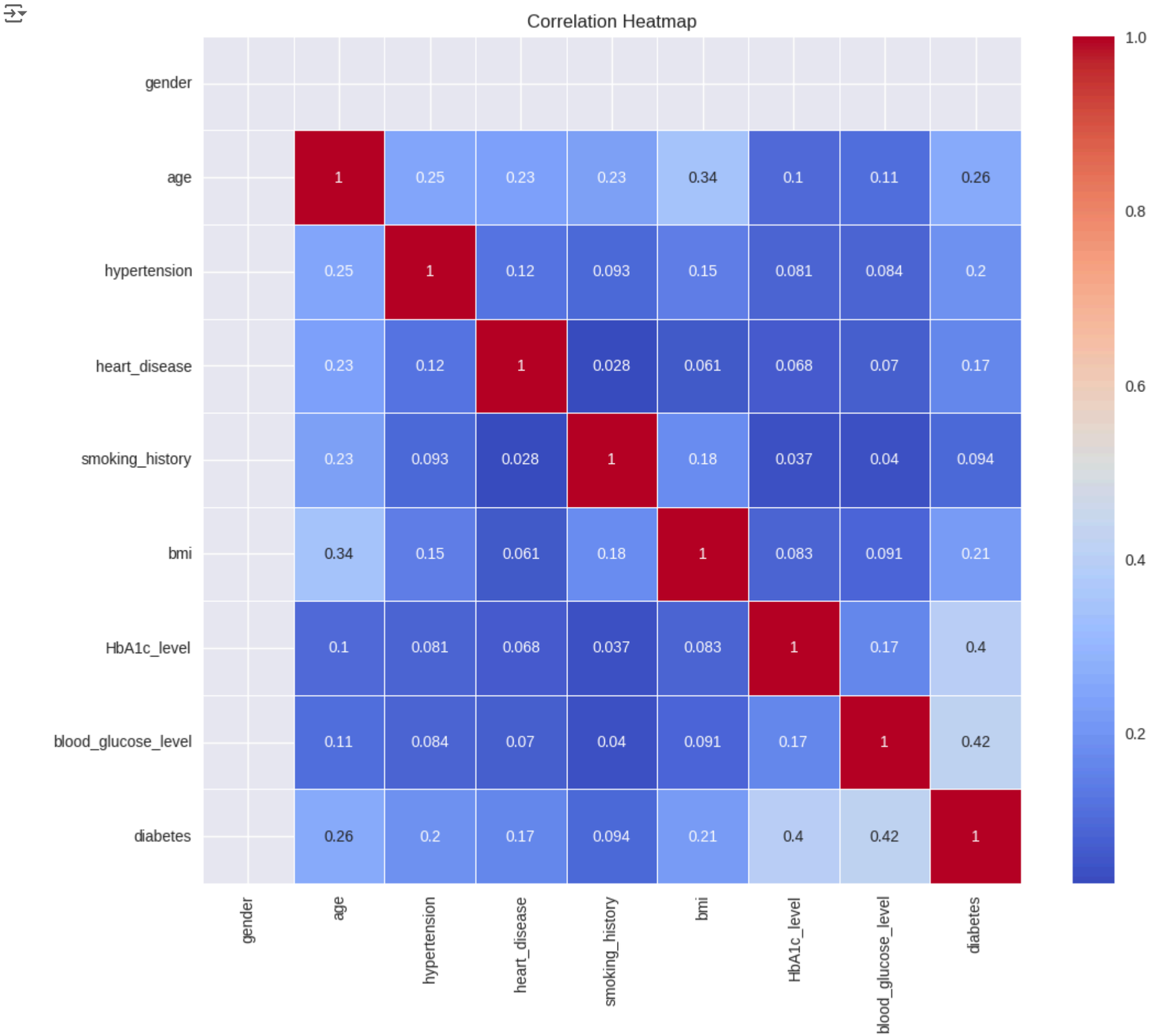
```
# 5. Smoking History and Diabetes
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='smoking_history', hue='diabetes')
plt.title('Diabetes Prevalence by Smoking History')
plt.xlabel('Smoking History')
plt.ylabel('Count')
plt.show()
```




```
# 6. Age vs BMI with Diabetes
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df, x='age', y='bmi', hue='diabetes', size='blood_glucose_level', sizes=(20, 200))
plt.title('Age vs BMI with Diabetes and Blood Glucose Level')
plt.xlabel('Age')
plt.ylabel('BMI')
plt.show()
```




```
# 7. Correlation Heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```


Correlation Heatmap
gender
age
hypertension
heart_disease
smoking_history
bmi
HbA1c_level
blood_glucose_level
diabetes
1 0.25 0.23 0.23 0.34 0.1 0.11 0.26
0.25 1 0.12 0.093 0.15 0.081 0.084 0.2
0.23 0.12 1 0.028 0.061 0.068 0.07 0.17
0.23 0.093 0.028 1 0.18 0.037 0.04 0.094
0.34 0.15 0.061 0.18 1 0.083 0.091 0.21
0.1 0.081 0.068 0.037 0.083 1 0.17 0.4
0.11 0.084 0.07 0.04 0.091 0.17 1 0.42
0.26 0.2 0.17 0.094 0.21 0.4 0.42 1
1.0
0.8
0.6
0.4
0.2

```
# 8. Pairplot
sns.pairplot(df, hue='diabetes', vars=['age', 'bmi', 'HbA1c_level', 'blood_glucose_level'])
plt.suptitle('Pairplot of Key Variables', y=1.02)
plt.show()
```