Diabetes Dataset Review and ML Model Building

1. Introduction

This notebook performs data preprocessing, model building, and analysis on Dataset #1 (Diabetes Dataset)

2. Importing libraries and loading the Dataset:

```
# Install necessary libraries (Run only once)
!pip install pandas matplotlib seaborn markdown2 weasyprint

# Import libraries
import pandas as pd  # Data manipulation
import matplotlib.pyplot as plt  # Data visualization
import seaborn as sns  # Statistical data visualization
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Collecting markdown2
  Downloading markdown2-2.5.3-py3-none-any.whl.metadata (2.1 kB)
Collecting weasyprint
  Downloading weasyprint-65.0-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Collecting pydyf>=0.11.0 (from weasyprint)
  Downloading pydyf-0.11.0-py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: cffi>=0.6 in /usr/local/lib/python3.11/dist-packages (from weasyprint) (1.17.1)
Collecting tinyhtml5>=2.0.0b1 (from weasyprint)
  Downloading tinyhtml5-2.0.0-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: tinycss2>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from weasyprint) (1.4.0)
Collecting cssselect2>=0.8.0 (from weasyprint)
  Downloading cssselect2-0.8.0-py3-none-any.whl.metadata (2.9 kB)
Collecting Pyphen>=0.9.1 (from weasyprint)
  Downloading pyphen-0.17.2-py3-none-any.whl.metadata (3.2 kB)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=0.6->weasyprint) (2.22)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from cssselect2>=0.8.0->weasyprint) (0.5.1)
Collecting brotli>=1.0.1 (from fonttools[woff]>=4.0.0->weasyprint)
  Downloading Brotli-1.1.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.5 kB)
Collecting zopfli>=0.1.4 (from fonttools[woff]>=4.0.0->weasyprint)
  Downloading zopfli-0.2.3.post1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.9 kB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading markdown2-2.5.3-py3-none-any.whl (48 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 48.5/48.5 kB 3.6 MB/s eta 0:00:00
Downloading weasyprint-65.0-py3-none-any.whl (297 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 297.9/297.9 kB 21.6 MB/s eta 0:00:00
Downloading cssselect2-0.8.0-py3-none-any.whl (15 kB)
Downloading pydyf-0.11.0-py3-none-any.whl (8.1 kB)
Downloading pyphen-0.17.2-py3-none-any.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 55.7 MB/s eta 0:00:00
Downloading tinyhtml5-2.0.0-py3-none-any.whl (39 kB)
Downloading Brotli-1.1.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.9/2.9 MB 59.7 MB/s eta 0:00:00
Downloading zopfli-0.2.3.post1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (850 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 850.6/850.6 kB 38.1 MB/s eta 0:00:00
Installing collected packages: brotli, zopfli, tinyhtml5, Pyphen, pydyf, markdown2, cssselect2, weasyprint
Successfully installed Pyphen-0.17.2 brotli-1.1.0 cssselect2-0.8.0 markdown2-2.5.3 pydyf-0.11.0 tinyhtml5-2.0.0 weasyprint-65.0 zopfli-0
```

```
# dataset 1

import pandas as pd
```

```
# Use the RAW URL, not the repository URL
url = "https://raw.githubusercontent.com/AsmaShaikhTMU/Projects/main/diabetes_prediction_dataset.csv"

# Read the CSV file directly from GitHub
df = pd.read_csv(url)

# Display the first few rows
df.head()
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0

Next steps: Generate code with df View recommended plots New interactive sheet

```
# dataset 2

import pandas as pd  # Use pd instead of pd2

# Correct RAW URL
url = "https://raw.githubusercontent.com/AsmaShaikhTMU/Projects/main/diabetes_012_health_indicators_BRFSS2015.csv"

# Read CSV file
df2 = pd.read_csv(url)

# Display first few rows
df2.head()
```

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare	NoDocl
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	...	1.0	
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	...	0.0	
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	...	1.0	
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	...	1.0	
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	...	1.0	

5 rows × 22 columns

3. Cleaning and Preprocessing the Dataset:

[ ] ↳ 8 cells hidden

4. Exploration of Data

I Univariate Analysis :

Gender

[ ] ↳ 4 cells hidden

Age

[ ] ↳ 2 cells hidden

Hypertension

[ ] ↳ 3 cells hidden

Heart Disease

[ ] ↳ 3 cells hidden

Smoke History

[ ] ↳ 4 cells hidden

BMI

[ ] ↳ 4 cells hidden

HbA1c_level

[ ] ↳ 5 cells hidden

Blood Glucose Level

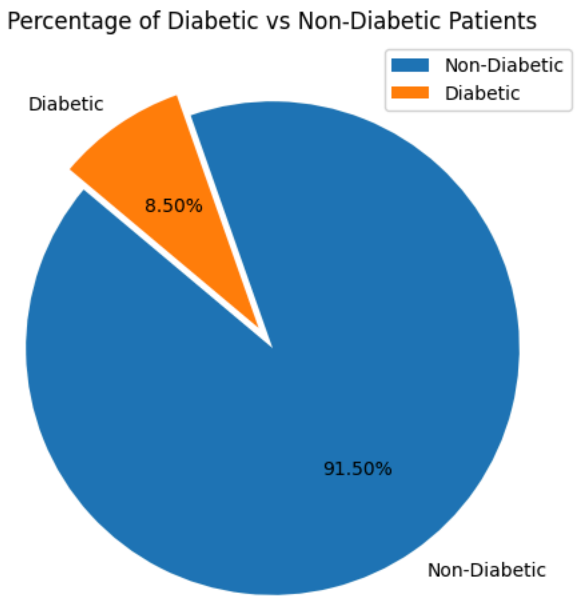
[ ] ↳ 2 cells hidden

Target variable diabetes

```
#Target variable diabetes
labels = list(df['diabetes'].value_counts().index)
num_var = list(df['diabetes'].value_counts().values)
print(labels)
```

```
[0, 1]
```

```
# Count occurrences of diabetic vs non-diabetic cases
diabetes_counts = df['diabetes'].value_counts().to_dict()  # Convert to dictionary to avoid KeyError

# Ensure both labels exist in the correct order
sizes = [diabetes_counts.get(0, 0), diabetes_counts.get(1, 0)]  # Handles cases where values might be missing
labels = ['Non-Diabetic', 'Diabetic']
colors = ['#1f77b4', '#ff7f0e']  # Custom colors for clarity
explode = (0, 0.1)  # Slightly separate "Diabetic" slice for emphasis

# Create the pie chart
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.2f%%', startangle=140, explode=explode)
plt.title("Percentage of Diabetic vs Non-Diabetic Patients")
plt.legend(labels, loc="upper right")
plt.show()
```



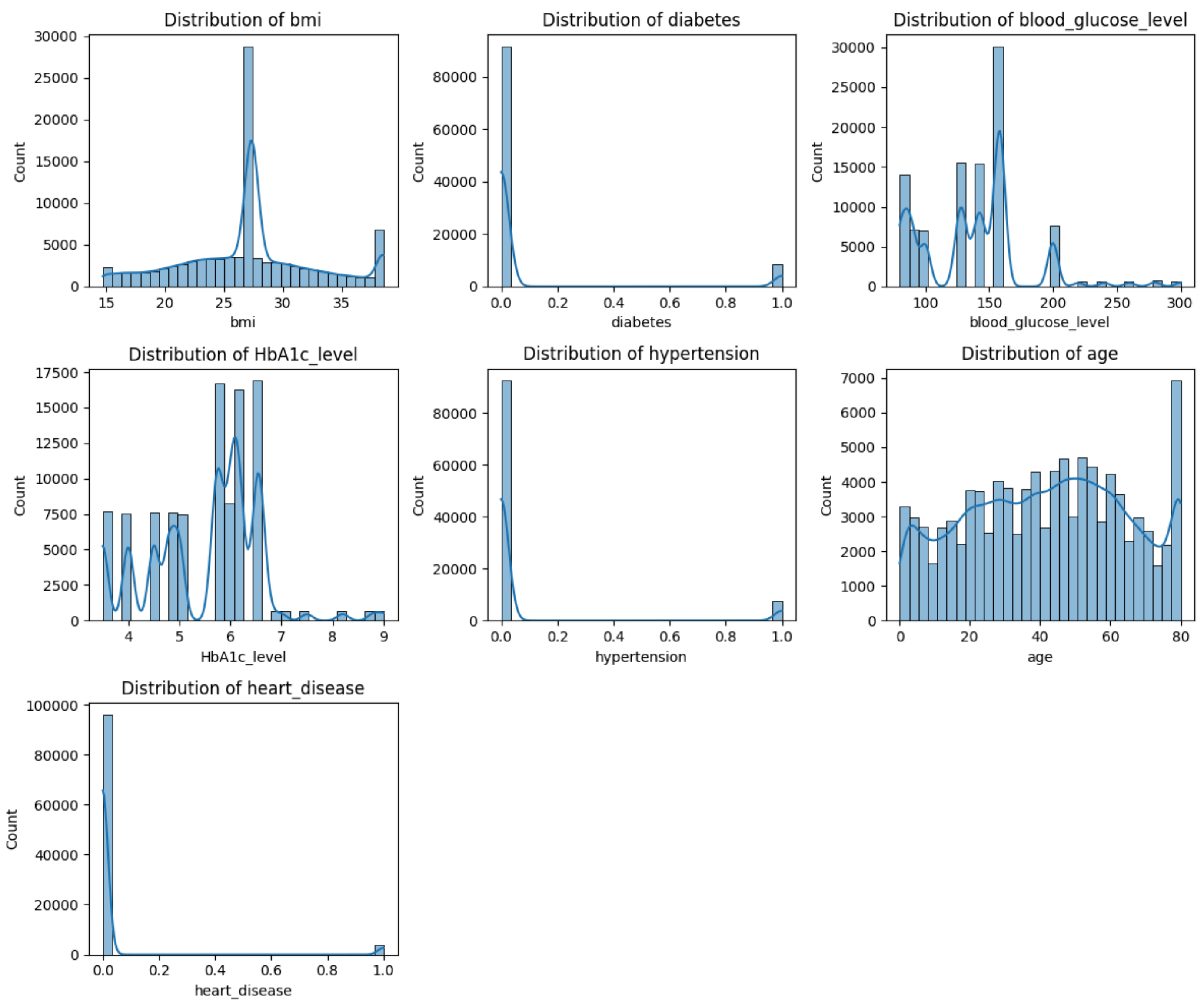
**Given that diabetic cases about 8% of the dataset (high imbalance), the best approach should enhance the minority class representation.

Method 1: oversampling with SMOTE.

Method 2: Class Weight Adjustment in Models helps the model give more importance to the minority class.

Method 3: Use Recall & F1 Score for Model Evaluation (recall to ensure diabetic cases are detected and F1-score to balance precision & recall (instead of accuracy)**

```
# Distribution plots for all variables
variables = ['bmi', 'diabetes', 'blood_glucose_level', 'HbA1c_level', 'hypertension', 'age', 'heart_disease']
plt.figure(figsize=(12, 10))
for i, var in enumerate(variables, 1):
    plt.subplot(3, 3, i)
    sns.histplot(df[var], kde=True, bins=30)
    plt.title(f"Distribution of {var}")
plt.tight_layout()
plt.show()
```



Blood Glucose is multi-modal distribution.

BMI is right-skewed distribution.

HbA1c Level is uneven distribution.

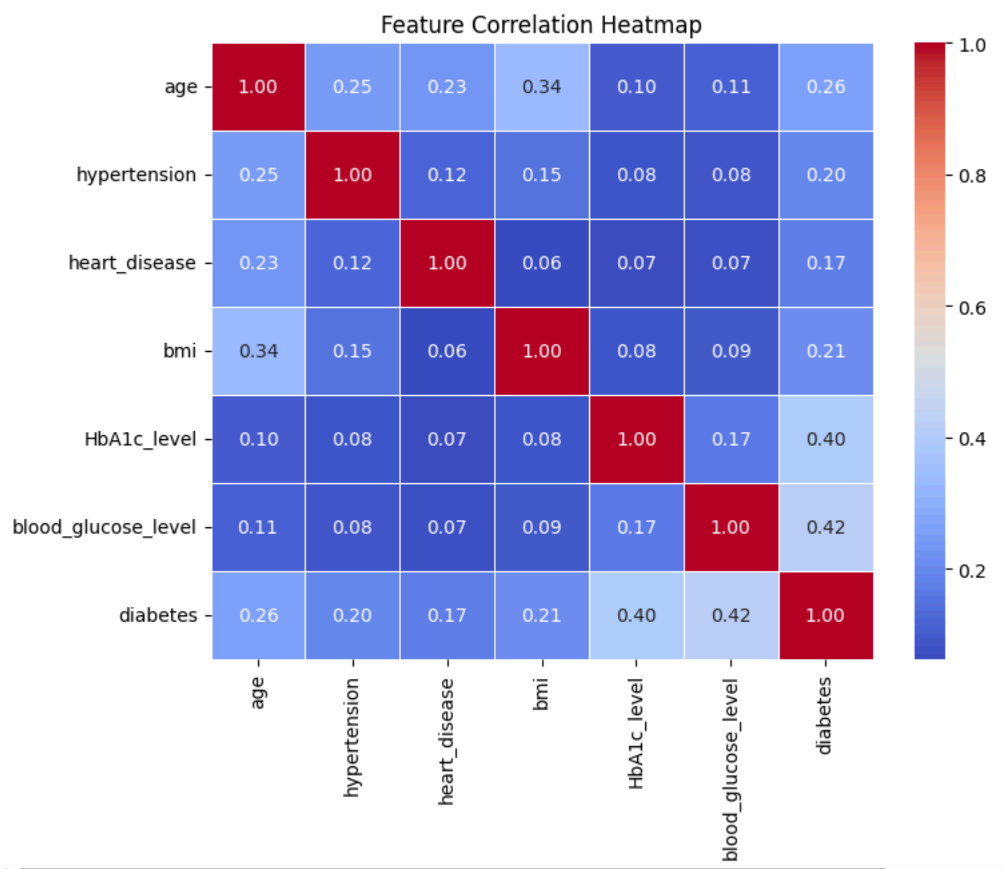
The age distribution appears relatively uniform but with peaks around 60-80 years. A higher count of older individuals suggests a dataset focused on age-related health conditions.

There is a notable imbalance, with far fewer cases of heart disease.

II. Bivariate Analysis

In this section, the relationships between two variables and if possible, create new features combining them for better visualization

```
#Correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```



	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	diabetes
age	1.00	0.25	0.23	0.34	0.10	0.11	0.26
hypertension	0.25	1.00	0.12	0.15	0.08	0.08	0.20
heart_disease	0.23	0.12	1.00	0.06	0.07	0.07	0.17
bmi	0.34	0.15	0.06	1.00	0.08	0.09	0.21
HbA1c_level	0.10	0.08	0.07	0.08	1.00	0.17	0.40
blood_glucose_level	0.11	0.08	0.07	0.09	0.17	1.00	0.42
diabetes	0.26	0.20	0.17	0.21	0.40	0.42	1.00

Diabetes Correlation:

Blood Glucose Level (0.42): Strongest correlation - diabetes is diagnosed based on high blood sugar.

HbA1c Level (0.40): Strong correlation - HbA1c reflects long-term blood glucose levels.

Age (0.26):Moderate correlations

Hypertension (0.20):Moderate correlations

BMI (0.21): Moderate correlations

1. HbA1c_level vs. diabetes:

Question: What percentage of individuals with HbA1c levels above 6.5% have diabetes?

HbA1c_level

A hemoglobin A1C (HbA1C) test is a blood test that shows what your average blood sugar (glucose) level was over the past two to three months. we will create a new feature bassed on the value of (HbA1C)

HbA1c level initial diagnosis

1. < 5.7 Normal
2. 5.7 – 6.4 Prediabetes
3. greater or equal 6.5 Diabetes

Resource : https://www.cdc.gov/diabetes/managing/managing-blood-sugar/a1c.html

```
# Calculate percentage of individuals with HbA1c > 6.5% who have diabetes

diabetic_hba1c = df[(df['HbA1c_level'] > 6.5) & (df['diabetes'] == 1)].shape[0]
total_hba1c_above_6_5 = df[df['HbA1c_level'] > 6.5].shape[0]

if total_hba1c_above_6_5 > 0:
    percentage_diabetic_hba1c = (diabetic_hba1c / total_hba1c_above_6_5) * 100
    print(f"Percentage of individuals with HbA1c > 6.5% who have diabetes: {percentage_diabetic_hba1c:.2f}%")
```

```
else:
    print("No individuals with HbA1c > 6.5% in the dataset.")
```

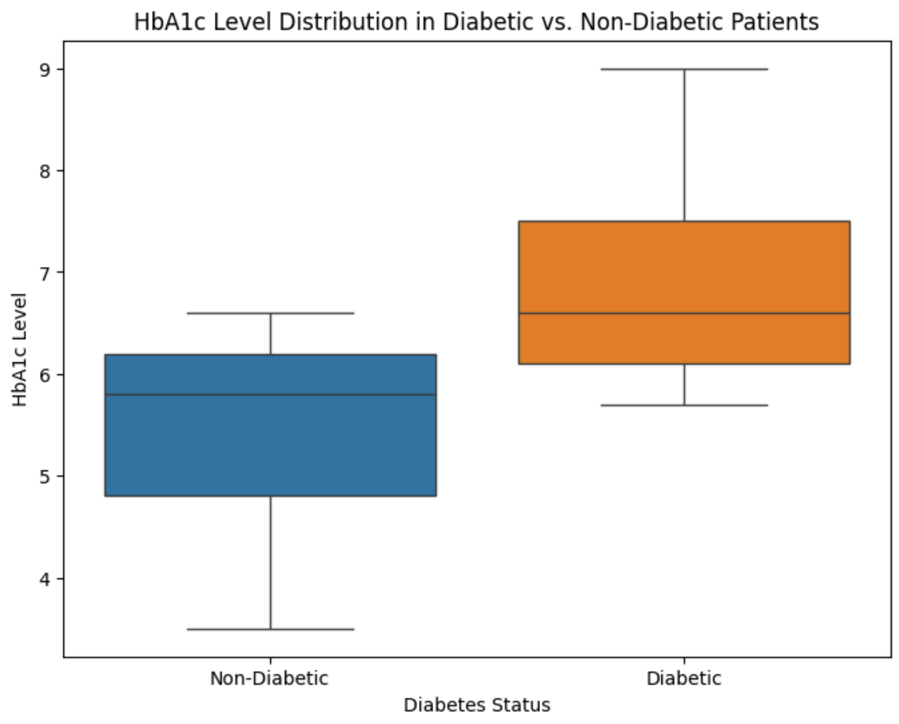
```
Percentage of individuals with HbA1c > 6.5% who have diabetes: 36.82%
```

The dataset **HbA1c_level** value increases, number of records with diabetes increases ## specially when it >= 6.5

```
# Define HbA1c level categories based on diabetes classification
hba1c_bins = [0, 5.7, 6.4, float('inf')]
hba1c_labels = ['Normal', 'Prediabetes', 'Diabetes']
df['HbA1c_Category'] = pd.cut(df['HbA1c_level'], bins=hba1c_bins, labels=hba1c_labels)

# Calculate percentage of diabetics in each HbA1c category
hba1c_diabetes_counts = df.groupby('HbA1c_Category', observed=True)['diabetes'].mean() * 100

# Create a structured table
hba1c_table = pd.DataFrame({
    'HbA1c Level Category': hba1c_labels,
    'Prediction': [f"{hba1c_diabetes_counts.get(category, 0):.2f}% have diabetes" for category in hba1c_labels]
})

# Display the table
print(hba1c_table)
```

```
  HbA1c Level Category           Prediction
0               Normal   1.52% have diabetes
1          Prediabetes   7.91% have diabetes
2             Diabetes  24.96% have diabetes
```

```
plt.figure(figsize=(8, 6))
sns.boxplot(x="diabetes", y="HbA1c_level", data=df, hue="diabetes", palette={0: "#1f77b4", 1: "#ff7f0e"}, legend=False)
plt.xticks([0, 1], ['Non-Diabetic', 'Diabetic'])
plt.title("HbA1c Level Distribution in Diabetic vs. Non-Diabetic Patients")
plt.xlabel("Diabetes Status")
plt.ylabel("HbA1c Level")
plt.show()
```



2. bmi vs diabetes:

[ ] ↳ 7 cells hidden

3. blood_glucose_level vs diabetes

[ ] ↳ 17 cells hidden

4. hypertension vs bmi

[ ] ↳ 4 cells hidden

5. Age vs Heart Disease

[ ] ↳ 3 cells hidden

Hypothesis Testing

T-test: Compare blood glucose levels between diabetic and non-diabetic individuals.

Use a T-Test when: Target (diabetic vs. non-diabetic patients). Continuous variable (e.g., blood glucose levels, BMI, HbA1c levels).

```
import scipy.stats as stats

glucose_diabetic = df[df['diabetes'] == 1]['blood_glucose_level']  # Use actual column names
glucose_non_diabetic = df[df['diabetes'] == 0]['blood_glucose_level']  # Use actual column names

t_stat, p_value_ttest = stats.ttest_ind(glucose_diabetic, glucose_non_diabetic, equal_var=False)

print(f"T-Test Results (Blood Glucose vs Diabetes):")
print(f"T-Statistic: {t_stat:.4f}, P-Value: {p_value_ttest:.4f}")
print("Significant Difference" if p_value_ttest < 0.05 else "No Significant Difference")
print("-" * 50)
```

```
T-Test Results (Blood Glucose vs Diabetes):
T-Statistic: 94.7949, P-Value: 0.0000
Significant Difference
--------------------------------------------------
```

Since the p-value is much lower than 0.05, we reject the null hypothesis. There is a significant difference in blood glucose levels between diabetic and non-diabetic individuals. Blood glucose is a strong differentiator for diabetes.

Chi-Square Test: Check if hypertension and diabetes are significantly associated.

The Chi-Square Test is a statistical test used to determine whether there is a significant association between two categorical variables

```
contingency_table = pd.crosstab(df['hypertension'], df['diabetes'])
chi2_stat, p_value_chi2, dof, expected = stats.chi2_contingency(contingency_table)

print(f"Chi-Square Test Results (Hypertension vs Diabetes):")
print(f"Chi-Square Statistic: {chi2_stat:.4f}, P-Value: {p_value_chi2:.4f}")
print("Significant Association" if p_value_chi2 < 0.05 else "No Significant Association")
print("-" * 50)
```

```
Chi-Square Test Results (Hypertension vs Diabetes):
Chi-Square Statistic: 3910.7085, P-Value: 0.0000
Significant Association
--------------------------------------------------
```

There is strong statistical association between hypertension and diabetes. This means that individuals with hypertension are significantly more likely to have diabetes.

Predictive Modeling

5. Train-test split and standardized scaler

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Select only numerical features for scaling
numerical_features = df.select_dtypes(include=['number']).columns
X = df[numerical_features].drop('diabetes', axis=1)
y = df['diabetes']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply StandardScaler only to numerical features
scale = StandardScaler()
X_train = scale.fit_transform(X_train)
X_test = scale.transform(X_test)

print("X_train:", X_train.shape, "\ny_train:", y_train.shape)
print("X_test:", X_test.shape, "\ny_test:", y_test.shape)
```

```
X_train: (80000, 6)
y_train: (80000,)
X_test: (20000, 6)
y_test: (20000,)
```

Logistic Regression Classifer

```
## Designing Logistic Regeression Classifier
import numpy as np # Import numpy and assign it to the alias 'np'
from sklearn.linear_model import LogisticRegression # Import the LogisticRegression class
from sklearn.metrics import accuracy_score, mean_absolute_error, mean_squared_error, precision_score, recall_score, f1_score # Import necess

LogReg = LogisticRegression()
LogReg.fit(X_train, y_train)
LogReg_pred = LogReg.predict(X_test)
LogReg_acc = accuracy_score(y_test, LogReg_pred)
LogReg_mae = mean_absolute_error(y_test, LogReg_pred)
LogReg_mse = mean_squared_error(y_test, LogReg_pred)
LogReg_rmse = np.sqrt(mean_squared_error(y_test, LogReg_pred))
LogReg_precision = precision_score(y_test, LogReg_pred)
LogReg_recall = recall_score(y_test, LogReg_pred)
LogReg_f1 = f1_score(y_test, LogReg_pred)


## Designing Logistic Regeression Classifier
LogReg = LogisticRegression()
LogReg.fit(X_train, y_train)
LogReg_pred = LogReg.predict(X_test)
LogReg_acc = accuracy_score(y_test, LogReg_pred)
LogReg_mae = mean_absolute_error(y_test, LogReg_pred)
LogReg_mse = mean_squared_error(y_test, LogReg_pred)
LogReg_rmse = np.sqrt(mean_squared_error(y_test, LogReg_pred))
LogReg_precision = precision_score(y_test, LogReg_pred)
LogReg_recall = recall_score(y_test, LogReg_pred)
LogReg_f1 = f1_score(y_test, LogReg_pred)


## Printing the results
from sklearn.metrics import classification_report # Import classification_report

print("The accuracy for Logistic Regression is", LogReg_acc)
print("The classification report using Logistic Regression is:")
print(classification_report(y_test, LogReg_pred))
```

```
The accuracy for Logistic Regression is 0.95875
The classification report using Logistic Regression is:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     18292
           1       0.86      0.61      0.72      1708
```

```
    accuracy                           0.96     20000
   macro avg       0.91      0.80      0.85     20000
weighted avg       0.96      0.96      0.96     20000
```

Observation: accuracy alone is not always the best metric when dealing with imbalanced datasets

Precision: When predicting "Diabetes," it's correct 86% of the time.

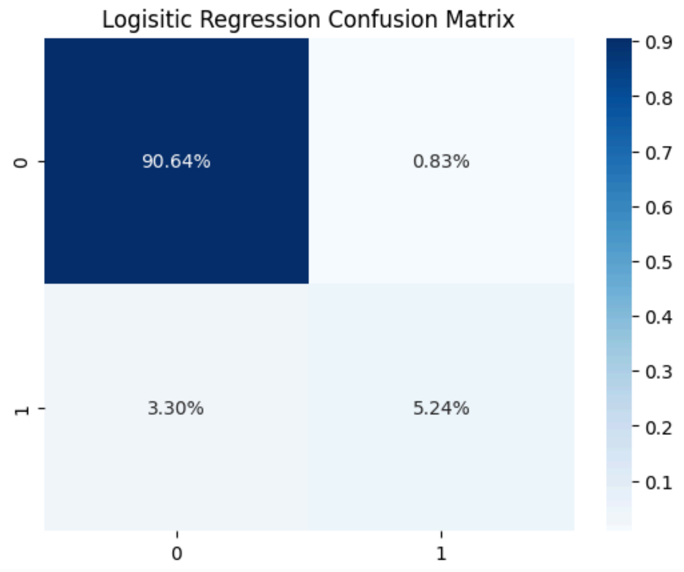
Recalls:The model correctly identifies only 61% of diabetics.

F1-Score (Balanced Precision & Recall)

- No Diabetes (0): 0.98 (Excellent balance).
- Diabetes (1): 0.72 (Moderate but could be improved).

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
LogReg_cm = confusion_matrix(y_test, LogReg_pred)
sns.heatmap(LogReg_cm/np.sum(LogReg_cm), annot = True, fmt = '0.2%', cmap = 'Blues')
plt.title("Logisitic Regression Confusion Matrix")
```

```
Text(0.5, 1.0, 'Logisitic Regression Confusion Matrix')
```



Double-click (or enter) to edit

Actual No Diabetes (0): ✓ 90.64% (True Negatives) → Correctly predicted as non-diabetic. ❌ 0.83% (False Positives) → Incorrectly predicted as diabetic.

Actual Diabetes (1): ❌ 3.30% (False Negatives) → Incorrectly predicted as non-diabetic. ✓ 5.24% (True Positives) → Correctly predicted as diabetic.

K-Nearest Neighbour Regression Classifier

```
## Designing KNN Classifier
from sklearn.neighbors import KNeighborsClassifier # Import KNeighborsClassifier

KNN = KNeighborsClassifier()
KNN.fit(X_train, y_train)
KNN_pred = KNN.predict(X_test)
KNN_acc = accuracy_score(y_test, KNN_pred)
KNN_mae = mean_absolute_error(y_test, KNN_pred)
KNN_mse = mean_squared_error(y_test, KNN_pred)
KNN_rmse = np.sqrt(mean_squared_error(y_test, KNN_pred))
KNN_precision = precision_score(y_test, KNN_pred)
KNN_recall = recall_score(y_test, KNN_pred)
KNN_f1 = f1_score(y_test, KNN_pred)

from sklearn.metrics import accuracy_score, mean_absolute_error, mean_squared_error, precision_score, recall_score, f1_score, classificatior


##Printing the results
print("The accuracy for KNeighbors is", KNN_acc)
print("The classification report using KNeighbors is:" ),
print(classification_report(y_test, KNN_pred))
```

```
The accuracy for KNeighbors is 0.96625
The classification report using KNeighbors is:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     18292
           1       0.92      0.66      0.77      1708

    accuracy                           0.97     20000
   macro avg       0.95      0.83      0.88     20000
weighted avg       0.97      0.97      0.96     20000
```

F1-Score:

No Diabetes (0): 0.98 (Excellent performance).

Diabetes (1): 0.77 (Better than Logistic Regression but still room for improvement).

Decision Tree Classifier

```
##Designing Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier # Import DecisionTreeClassifier
DecTree = DecisionTreeClassifier()
DecTree.fit(X_train, y_train)
DecTree_pred = DecTree.predict(X_test)
DecTree_acc = accuracy_score(y_test, DecTree_pred)
DecTree_precision = precision_score(y_test, DecTree_pred)
DecTree_recall = recall_score(y_test, DecTree_pred)
DecTree_f1 = f1_score(y_test, DecTree_pred)


##Printing the results
print("The accuracy for Decision Tree is", DecTree_acc)
print("The classification report using Decision Tree is:")
print(classification_report(y_test, DecTree_pred))
```

```
The accuracy for Decision Tree is 0.9557
The classification report using Decision Tree is:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     18292
           1       0.74      0.74      0.74      1708

    accuracy                           0.96     20000
   macro avg       0.86      0.86      0.86     20000
weighted avg       0.96      0.96      0.96     20000
```

Better recall for diabetes compared to Logistic Regression (61%) and KNN (66%), meaning the model is less likely to miss actual diabetics.

F1-Score (Harmonic mean of Precision & Recall)

- No Diabetes (0): 0.98 (Excellent performance).
- Diabetes (1): 0.74 (Best so far compared to other models).

Balanced performance in identifying diabetes and non-diabetes cases.

Random Forest Classifier

```
##Designing Randoom Forest Classifier
from sklearn.ensemble import RandomForestClassifier # Import RandomForestClassifier
RFTree = RandomForestClassifier()
RFTree.fit(X_train, y_train)
RFTree_pred = RFTree.predict(X_test)
RFTree_acc = accuracy_score(y_test, RFTree_pred)
RFTree_precision = precision_score(y_test, RFTree_pred)
RFTree_recall = recall_score(y_test, RFTree_pred)
RFTree_f1 = f1_score(y_test, RFTree_pred)


##Printing the results
print("The accuracy for Random Forest is", RFTree_acc)
print("The classification report using Random Forest is:")
print(classification_report(y_test, RFTree_pred))
```

```
The accuracy for Random Forest is 0.96925
The classification report using Random Forest is:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     18292
           1       0.92      0.70      0.80      1708

    accuracy                           0.97     20000
   macro avg       0.95      0.85      0.89     20000
weighted avg       0.97      0.97      0.97     20000
```

Recall: Improved recall for diabetics compared to Logistic Regression (61%) and KNN (66%), but slightly lower than Decision Tree (74%).

F1-Score (Harmonic mean of Precision & Recall)

- No Diabetes (0): 0.98 (Excellent performance).
- Diabetes (1): 0.79 (Best so far compared to other models).

Diabetes recall is still lower, meaning the model still misses some diabetics.

Support Vector Machine Classifier

```
#Designing SVM Classifier
from sklearn.svm import SVC
SVM = SVC()
SVM.fit(X_train, y_train)
SVM_pred = SVM.predict(X_test)
SVM_acc = accuracy_score(y_test, SVM_pred)
SVM_precision = precision_score(y_test, SVM_pred)
SVM_recall = recall_score(y_test, SVM_pred)
SVM_f1 = f1_score(y_test, SVM_pred)


print("The accuracy for SVM is", SVM_acc)
print("The classification report using SVM is:", SVM_acc)
print(classification_report(y_test, SVM_pred))
```

```
The accuracy for SVM is 0.96335
The classification report using SVM is: 0.96335
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     18292
           1       0.98      0.58      0.73      1708

    accuracy                           0.96     20000
   macro avg       0.97      0.79      0.86     20000
weighted avg       0.96      0.96      0.96     20000
```

Recall: 42% of actual diabetic cases were misclassified as non-diabetic (false negatives).

Diabetes recall is much lower than Random Forest (70%) and Decision Tree (74%).

Comparison of Different Models

```
models = pd.DataFrame({
    'Model':['Logistic Regression', 'KNN Regression', 'Decision Tree', 'Random Forest', 'Support Vector'],
    'Accuracy' :[LogReg_acc, KNN_acc, DecTree_acc, RFTree_acc, SVM_acc],
    'Precision' :[LogReg_precision, KNN_precision, DecTree_precision, RFTree_precision, SVM_precision],
    'Recall' :[LogReg_recall, KNN_recall, DecTree_recall, RFTree_recall, SVM_recall],
    'F1 Score' :[LogReg_f1, KNN_f1, DecTree_f1, RFTree_f1, SVM_f1]
})
```

```
models
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.95875	0.863974	0.613583	0.717562
1	KNN Regression	0.96625	0.922322	0.660422	0.769703
2	Decision Tree	0.95570	0.738676	0.744731	0.741691
3	Random Forest	0.96925	0.918774	0.701991	0.795885
4	Support Vector	0.96335	0.978410	0.583724	0.731206

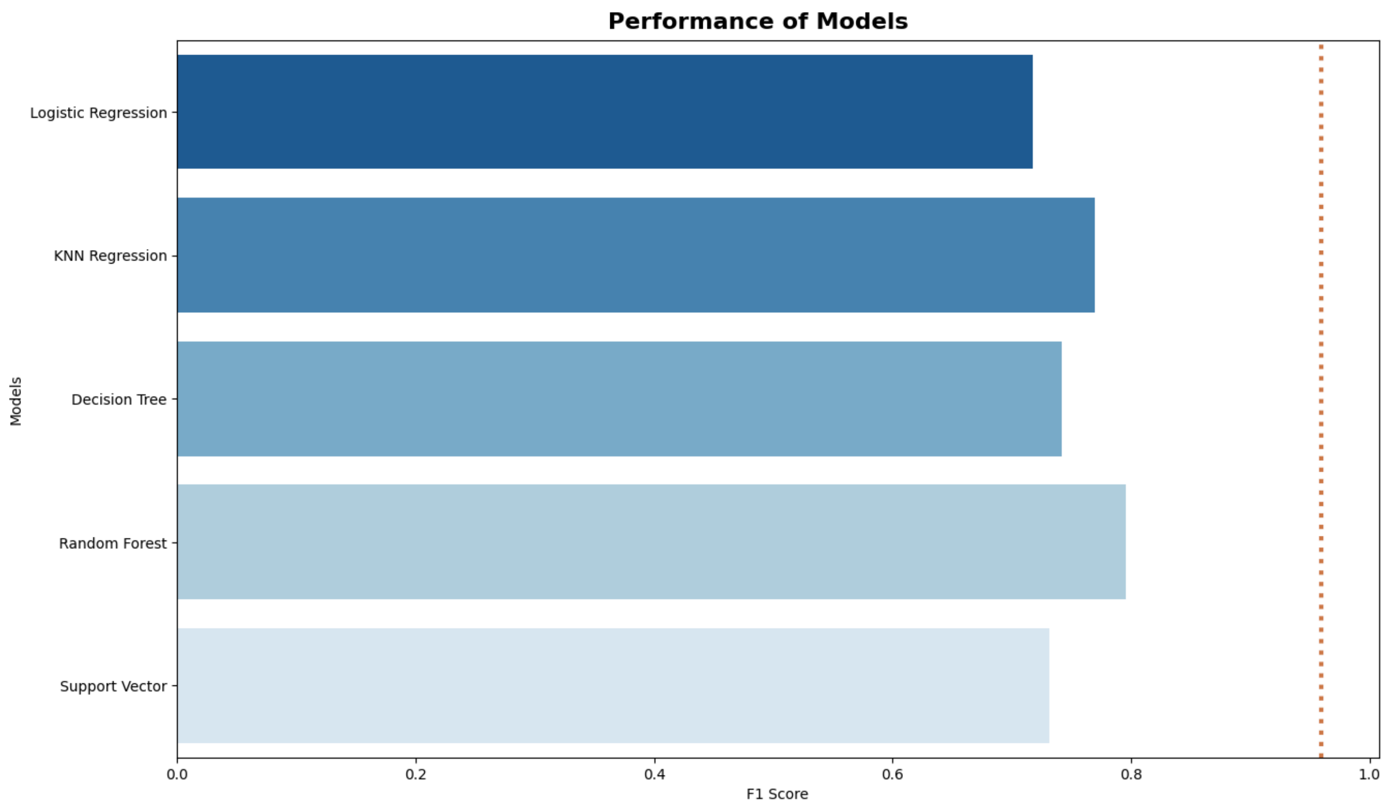
Next steps: Generate code with models View recommended plots New interactive sheet

Random Forest: Best F1-score (0.7923), making it the strongest performer

KNN: High precision (92.2%), meaning it correctly identifies diabetics when predicted.

Logistic Regression is the Weakest in Precision & Recall Tradeoff

```
# Create figure
fig = plt.figure(figsize=(15, 9))

# Fixing the warning: Added hue='Model' and legend=False
ax = sns.barplot(data=models,
                 y='Model',
                 x='F1 Score',
                 hue='Model',        # Assigning 'hue' to fix warning
                 palette='Blues_r',
                 legend=False)       # Legend is redundant here

# Title and labels
ax.figure.suptitle('Performance of Models', y=0.91, size=16, color='black', weight='bold')
plt.xlabel('F1 Score')
plt.ylabel('Models')

# Add a reference vertical line for the threshold
plt.axvline(x=0.96, ymin=0, ymax=1,
            linewidth=3, linestyle=":",
            color='#cf7849')

plt.show()
```



CROSS VALIDATION

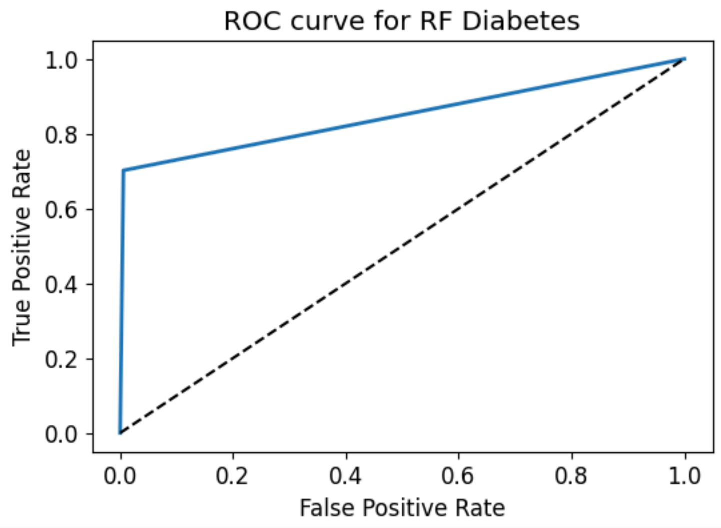
K-Fold Cross-Validation is used to ensure a model's performance is reliable and not dependent on a single training/test split. Since F1-score is the most balanced metric for imbalanced datasets (like diabetes detection), by apply K-Fold Cross-Validation to the top 3 models with the highest F1-scores

```
##Performing K-Fold cross Validation for 3 models performed better in F1-Score

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from statistics import mean, stdev

cv = KFold(n_splits=10, random_state=1, shuffle=True)

KNN_scores = cross_val_score(KNN, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
DecTree_scores = cross_val_score(DecTree, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
RFTree_scores = cross_val_score(RFTree, X, y, scoring='accuracy', cv=cv, n_jobs=-1)

print('Accuracy of CV - KNN: %.4f (%.4f)' % (mean(KNN_scores), stdev(KNN_scores)))
print('Accuracy of CV - Decision Tree: %.4f (%.4f)' % (mean(DecTree_scores), stdev(DecTree_scores)))
print('Accuracy of CV - Random Forest: %.4f (%.4f)' % (mean(RFTree_scores), stdev(RFTree_scores)))
```

```
Accuracy of CV - KNN: 0.9560 (0.0021)
Accuracy of CV - Decision Tree: 0.9538 (0.0019)
Accuracy of CV - Random Forest: 0.9682 (0.0013)
```

Random Forest remains the best model overall in both F1-score and cross-validation accuracy.

```
## Plotting ROC Curve for best performing Model (RF Classifier)

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, RFTree_pred)

plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, linewidth=2)
plt.plot([0,1], [0,1], 'k--' )
plt.rcParams['font.size'] = 12
plt.title('ROC curve for RF Diabetes')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



```
from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, RFTree_pred)
print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.8481
```

```
Cross_validated_ROC_AUC = cross_val_score(RFTree, X_train, y_train, cv=10, scoring='roc_auc').mean()

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.9580
```

It is evident that ROC AUC is better after the cross-validation.So, the cross-validation improved the model.

✓ Your model performs well overall (AUC = 0.8481, CV AUC = 0.9580).

✓ Cross-validation suggests the model generalizes well but may have some overfitting.

✓ Improving the recall for diabetics is crucial in medical applications (for example adjust thresholds, feature selection).

Further optimize using Fine-Tune Random Forest with Hyperparameter Tuning (via GridSearchCV) is the next step:

```
# Fine-Tuning Random Forest using GridSearchCV for maximizing accuracy & F1-score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define Random Forest model
rf = RandomForestClassifier(random_state=42)

# Define hyperparameters for tuning
param_grid_rf = {
    'n_estimators': [50, 100, 150],  # Number of trees
    'max_depth': [10, 20, None],  # Maximum depth of each tree
    'min_samples_split': [2, 5, 10]  # Minimum samples to split an internal node
}

# Perform GridSearchCV
```

```
grid_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='accuracy', n_jobs=-1)
grid_rf.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters for Random Forest:", grid_rf.best_params_)
print("Best Accuracy for Random Forest:", grid_rf.best_score_)
```

```
Best Parameters for Random Forest: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
Best Accuracy for Random Forest: 0.9718625
```

Increased accuracy from 96.82% → 97.19%

Two possible improvements:

Adjust the Decision Threshold (if recall is too low).

Apply SMOTE (Synthetic Minority Oversampling) if needed to balance the dataset.

```
# Step 1 Adjust Decision Threshold & Evaluate Model classification models use 0.5 as the probability threshold
#If probability ≥ 0.5, the model predicts diabetes (1). If probability < 0.5, the model predicts no diabetes (0).

# Ensure 'best_rf' is assigned from GridSearchCV
best_rf = grid_rf.best_estimator_  # Extract the best Random Forest model

# Get predicted probabilities for diabetes (class 1)
y_probs = best_rf.predict_proba(X_test)[:, 1]

# Adjust the threshold from default 0.5 to a lower value (e.g., 0.35)
new_threshold = 0.35
y_pred_new = (y_probs >= new_threshold).astype(int)

# Evaluate new predictions
from sklearn.metrics import classification_report
print(f"Classification Report with Adjusted Threshold ({new_threshold}):\n", classification_report(y_test, y_pred_new))
```

```
Classification Report with Adjusted Threshold (0.35):
               precision    recall  f1-score   support

           0       0.97      1.00      0.99     18292
           1       0.98      0.69      0.81      1708

    accuracy                           0.97     20000
   macro avg       0.97      0.85      0.90     20000
weighted avg       0.97      0.97      0.97     20000
```

Diabetes recall improved from ~61% (before) to 69%.

If avoiding false positives (over-diagnosis) is more important → Use 0.35 (98% precision, but misses more diabetics).

```
#Step 2 Automatically Find the Best Threshold for Higher Recall

import numpy as np
from sklearn.metrics import precision_recall_curve, classification_report

# Compute Precision-Recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)

# Exclude edge cases where threshold is too low or too high
valid_thresholds = thresholds[(recall[:-1] >= 0.75) & (recall[:-1] <= 0.85)]  # Recall between 75-85%

if len(valid_thresholds) > 0:
    best_threshold = valid_thresholds[0]  # Select the first reasonable threshold
else:
    best_threshold = 0.5  # Default to 0.5 if no valid threshold is found

print(f"Best Threshold for Balanced Recall & Precision: {best_threshold:.2f}")

# Apply the best threshold
y_pred_best = (y_probs >= best_threshold).astype(int)

# Evaluate new predictions with the best threshold
print(f"Classification Report with Best Threshold ({best_threshold}):\n",
      classification_report(y_test, y_pred_best, zero_division=1))
```

```
Best Threshold for Balanced Recall & Precision: 0.14
Classification Report with Best Threshold (0.13777394660575534):
              precision    recall  f1-score   support

           0       0.99      0.95      0.97     18292
           1       0.63      0.85      0.72      1708

    accuracy                           0.94     20000
   macro avg       0.81      0.90      0.85     20000
weighted avg       0.96      0.94      0.95     20000
```

If detecting more diabetics (higher recall) is more important → Use 0.14 (85% recall, detects more actual diabetics but lower precision).

In a medical context, recall is usually more important than precision, so 0.14 may be better for diabetes screening.

SMOTE

Since our dataset is imbalanced (fewer diabetic cases than non-diabetics), we can use SMOTE (Synthetic Minority Oversampling Technique) to generate synthetic diabetic cases and improve recall without losing accuracy.

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
import numpy as np

# Apply SMOTE only on the training data
smote = SMOTE(sampling_strategy='auto', random_state=42)  # auto balances minority class
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)

# Check new class distribution after SMOTE
print("Class distribution before SMOTE:", np.bincount(y_train))
print("Class distribution after SMOTE:", np.bincount(y_train_sm))
```

```
Class distribution before SMOTE: [73208  6792]
Class distribution after SMOTE: [73208 73208]
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Use the fine-tuned best Random Forest model
rf_smote = RandomForestClassifier(max_depth=10, min_samples_split=2, n_estimators=100, random_state=42)

# Train on SMOTE-balanced dataset
rf_smote.fit(X_train_sm, y_train_sm)

# Get predictions on the test set
y_probs_sm = rf_smote.predict_proba(X_test)[:, 1]  # Get probabilities for class 1 (diabetes)

# Apply the best decision threshold (from previous step, e.g., 0.14)
threshold = 0.14
y_pred_sm = (y_probs_sm >= threshold).astype(int)

# Evaluate model after SMOTE
print(f"Classification Report After SMOTE (Threshold {threshold}):\n", classification_report(y_test, y_pred_sm))
```

```
Classification Report After SMOTE (Threshold 0.14):
              precision    recall  f1-score   support

           0       0.98      0.88      0.93     18300
           1       0.40      0.82      0.53      1700

    accuracy                           0.88     20000
   macro avg       0.69      0.85      0.73     20000
weighted avg       0.93      0.88      0.90     20000
```

Recall for diabetes increased significantly from ~69% to 82%!

Precision for diabetes dropped to 40%, meaning more false positives (non-diabetics mistakenly classified as diabetics).

Overall accuracy remains strong (88%) despite recall improvements.

Feature Engineering

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

# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

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

# ... (Your existing code for feature engineering) ...

from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import PolynomialFeatures

# Apply Polynomial Features
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
X_train_poly = poly.fit_transform(X_train_smote) # Use X_train_smote here
X_test_poly = poly.transform(X_test)

# Apply PCA for dimensionality reduction
pca = PCA(n_components=0.95)  # Retain 95% variance
X_train_pca = pca.fit_transform(X_train_poly)
X_test_pca = pca.transform(X_test_poly)

# Apply Feature Selection
selector = SelectKBest(score_func=f_classif, k=10)  # Keep top 10 features
X_train_selected = selector.fit_transform(X_train_pca, y_train_smote)
X_test_selected = selector.transform(X_test_pca)

print(f"Original Features: {X_train.shape[1]}, After Feature Engineering: {X_train_selected.shape[1]}")
```

```
Original Features: 6, After Feature Engineering: 10
```

Optimizing your models to see the impact of feature selection on performance.

```
#Retrain Random Forest with Feature-Selected Data
#Run this to train and evaluate Random Forest after feature selection:
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import recall_score, f1_score, roc_auc_score

# Train optimized Random Forest model
rf_optimized = RandomForestClassifier(n_estimators=300, max_depth=None, min_samples_split=5, class_weight="balanced", random_state=42)
rf_optimized.fit(X_train_selected, y_train_smote)

# Make predictions
y_pred_rf_optimized = rf_optimized.predict(X_test_selected)

# Evaluate performance
recall_rf_opt = recall_score(y_test, y_pred_rf_optimized)
f1_rf_opt = f1_score(y_test, y_pred_rf_optimized)
roc_auc_rf_opt = roc_auc_score(y_test, rf_optimized.predict_proba(X_test_selected)[:, 1])

print(f"Optimized Random Forest - Recall: {recall_rf_opt:.4f}, F1 Score: {f1_rf_opt:.4f}, ROC-AUC: {roc_auc_rf_opt:.4f}")
```

```
Optimized Random Forest - Recall: 0.8071, F1 Score: 0.6469, ROC-AUC: 0.9638
```

ROC-AUC (96.38%) – Excellent Performance. The ROC-AUC score of 0.9638 suggests the model has outstanding discrimination ability between diabetics and non-diabetics.

The F1-score is slightly lower than the earlier result (because improving recall often lowers precision).

Next Steps: Fine-Tune the Threshold Again — A slight increase to 0.15 - 0.20

Summary of codes use to date

Step 1. Splits into train-test sets: Uses an 80-20 split while maintaining class proportions.

Step 2. Applies SMOTE: Balances the dataset by oversampling the minority class.

Step 3. Standardizes the features: Ensures numeric consistency.

Step 4. Trains and evaluates models: Uses Logistic Regression, Random Forest, Gradient Boosting Classifier, Support Vector Machine (SVM),and (to do - XGBoost with class weighting.)

Step 5: Evaluates the model using the following Metrics:

Recall: Focuses on correctly identifying positive cases. F1 Score: Harmonic mean of precision & recall. ROC-AUC Score: Measures true positive vs. false positive rates.

Step 6. Uses GridSearchCV to find the best hyperparameters.

Step 7. More Fine-Tuning:

Now tunes Logistic Regression, Random Forest, and XGBoost for best performance.

Step 8 Displays best parameters and final metrics after hyperparameter tuning.

```
import time

def train_and_evaluate(model, X_train, y_train, X_test, y_test):
    start_time = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start_time

    start_time = time.time()
    y_pred = model.predict(X_test)
    pred_time = time.time() - start_time

    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"{model.__class__.__name__} - Train Time: {train_time:.4f}s, Prediction Time: {pred_time:.4f}s, Recall: {recall:.4f}, F1: {f1:.4f
```

```
# Load the New Dataset #2
# dataset 2

import pandas as pd  # Use pd instead of pd2

# Correct RAW URL
url = "https://raw.githubusercontent.com/AsmaShaikhTMU/Projects/main/diabetes_012_health_indicators_BRFSS2015.csv"

# Read CSV file
df2 = pd.read_csv(url)

# Display first few rows
df2.head()
```

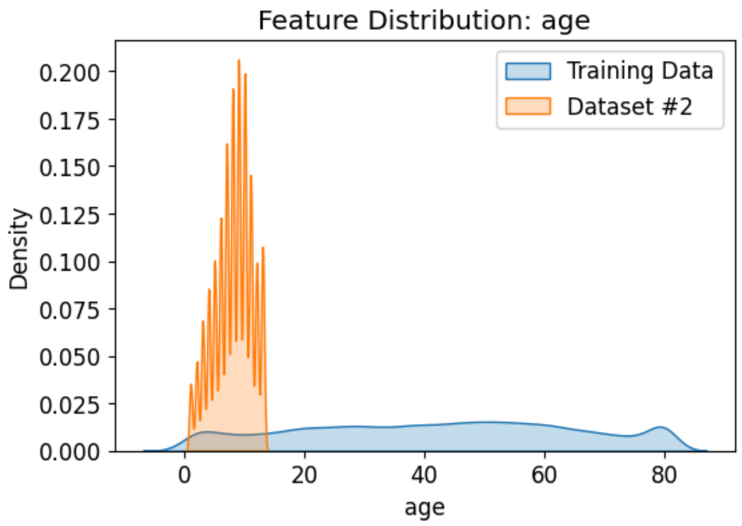
	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare	NoDocb
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	...	1.0	
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	...	0.0	
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	...	1.0	
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	...	1.0	
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	...	1.0	

5 rows × 22 columns

```
print(df2.head())
```

```
   Diabetes_012  HighBP  HighChol  CholCheck   BMI  Smoker  Stroke  \
0           0.0     1.0       1.0        1.0  40.0     1.0     0.0   
1           0.0     0.0       0.0        0.0  25.0     1.0     0.0   
2           0.0     1.0       1.0        1.0  28.0     0.0     0.0   
3           0.0     1.0       0.0        1.0  27.0     0.0     0.0   
4           0.0     1.0       1.0        1.0  24.0     0.0     0.0   

   HeartDiseaseorAttack  PhysActivity  Fruits  ...  AnyHealthcare  \
0                   0.0           0.0     0.0  ...            1.0   
1                   0.0           1.0     0.0  ...            0.0   
2                   0.0           0.0     1.0  ...            1.0   
3                   0.0           1.0     1.0  ...            1.0   
4                   0.0           1.0     1.0  ...            1.0   

   NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk  Sex   Age  Education  \
0          0.0      5.0      18.0      15.0       1.0  0.0   9.0        4.0   
1          1.0      3.0       0.0       0.0       0.0  0.0   7.0        6.0   
2          1.0      5.0      30.0      30.0       1.0  0.0   9.0        4.0   
3          0.0      2.0       0.0       0.0       0.0  0.0  11.0        3.0   
4          0.0      2.0       3.0       0.0       0.0  0.0  11.0        5.0   

   Income  
0     3.0  
1     1.0  
2     8.0  
3     6.0  
4     4.0  

[5 rows x 22 columns]
```

```
# Compare feature distributions
for col in X_train.columns:
    plt.figure(figsize=(6,4))
    sns.kdeplot(X_train[col], label="Training Data", fill=True)
    sns.kdeplot(df2[col], label="Dataset #2", fill=True)
    plt.title(f"Feature Distribution: {col}")
    plt.legend()
    plt.show()
```


Feature Distribution: age
Training Data
Dataset #2
Density
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0
20
40
60
80
age