**Harnessing Machine Learning for Diabetes Risk Prediction**

Machine Learning

Final Project

Dr. Ali Zaart

By:

Hadi Hammoud

Table of Contents

[Fraud Detection Model Report 1](#_Toc180949414)

[Business Problem 3](#_Toc180949415)

[Understanding the Dataset Features 3](#_Toc180949416)

[Data Exploration and Insights 4](#_Toc180949417)

[Distribution of target variable 4](#_Toc180949418)

[Correlation Heatmap 4](#_Toc180949419)

[Decision tree Visualization 5](#_Toc180949420)

[Model Performance and Suggestions for Improvement 7](#_Toc180949421)

[Recommendations: 7](#_Toc180949422)

# Business Problem:

A healthcare organization wants to improve preventive care by identifying individuals at high risk of developing diabetes. By predicting diabetes risk based on demographic, lifestyle, and medical data, the organization aims to prioritize at-risk individuals with tailored interventions, thereby reducing the prevalence of diabetes and improving overall patient outcomes.

**1. Collecting Data:**

Dataset:  
<https://www.kaggle.com/datasets/rcratos/diabetes-risk-prediction/data>

# Dataset Features:

1. **Age**: Age of the individual.
2. **Gender**: Gender of the individual (e.g., Male, Female).
3. **Polyuria**: Frequent urination (Yes/No).
4. **Polydipsia**: Excessive thirst (Yes/No).
5. **Sudden weight loss**: Indicates sudden and unexplained weight loss (Yes/No).
6. **Weakness**: General weakness or fatigue (Yes/No).
7. **Polyphagia**: Excessive hunger (Yes/No).
8. **Genital thrush**: Presence of genital thrush (Yes/No).
9. **Visual blurring**: Blurred vision (Yes/No).
10. **Itching**: Skin itching (Yes/No).
11. **Irritability**: Mood irritability (Yes/No).
12. **Delayed healing**: Delayed wound or cut healing (Yes/No).
13. **Partial paresis**: Weakness or partial paralysis (Yes/No).
14. **Muscle stiffness**: Muscle stiffness or tightness (Yes/No).
15. **Alopecia**: Hair loss or baldness (Yes/No).
16. **Obesity**: Presence of obesity (Yes/No).
17. **Class**: Target variable indicating diabetes status (Positive/Negative).

**2. Preparing the Data:**

1. Data Info:

df.head()

A black screen with white text

Description automatically generated

df.info()

A screenshot of a computer

Description automatically generated

From the initial inspection of the dataset, we noticed that the first few rows predominantly display positive cases, indicating the need to shuffle the data for a balanced representation before splitting. Additionally, the data contains several columns with object types, which will need to be transformed into numerical values for effective analysis.

1. Data cleaning:

**checking for null values:**

df.isnull().sum()

cases:

for Numerical attributes

* when the number of the missing value is low we can study the central tendency of the data if
* the standard deviation is low(which means that the variability of between the data is normal
* we find the mean and assign it to the missing value otherwise we find the mode which is not affected by deviation(it is not affected by the outlier).
* when we have high number of missing values and the column or the row is not essential we might just drop the rows

for Categorical attributes:

* The missing values can be filled using the mode (most frequent value).
* If there are significant missing values, the data can be dropped.

A screen shot of a computer

Description automatically generated

We did not have missing data in our dataset.

**Checking for duplicates:**

# Count occurrences of each row

duplicates\_count = df[df.duplicated(keep=False)]  # Select all rows that are duplicates

duplicates\_summary = duplicates\_count.value\_counts().reset\_index(name='Count')

# Display duplicates with their counts

if not duplicates\_summary.empty:

    print("Duplicate rows and their counts:")

    display(duplicates\_summary)

else:

    print("No duplicate rows found.")

# here required the duplicates removal

A screenshot of a computer

Description automatically generated

Deleting the duplicates:

# Print the number of rows before removing duplicates

print(f"Number of rows before removing duplicates: {df.shape[0]}")

# Remove duplicate rows, keeping only the first occurrence

df\_cleaned = df.drop\_duplicates(keep='first').copy()

# Print the number of rows after removing duplicates

print(f"Number of rows after removing duplicates: {df\_cleaned.shape[0]}")

# Display the cleaned dataset

df\_cleaned.head()

A screenshot of a black and white screen

Description automatically generated

**Outliers Detection:**

# Create boxplots for each numeric column

numeric\_columns = df\_cleaned.select\_dtypes(include=['number']).columns  # Select only numeric columns

# Loop through each numeric column and create a boxplot

for column in numeric\_columns:

    plt.figure(figsize=(8, 5))

    sns.boxplot(x=df[column])

    plt.title(f"Boxplot of {column}")

    plt.show()

#here requires the removing of the outlier

**A graph of a box plot

Description automatically generated**

**A blue rectangular object with white text

Description automatically generated** **A blue rectangular object with numbers

Description automatically generated**

A blue rectangular object with white text

Description automatically generated A blue rectangular graph with white text

Description automatically generated

A graph with a number of lines

Description automatically generated with medium confidence …ect

After identifying outliers in the dataset, we preserve the dataset size by avoiding row removal and instead use methods like regression or clipping to handle the outliers. Regression involves finding the attribute most correlated with the outlier's attribute, building a regression model to estimate a better value, while clipping replaces values below the lower bound with the lower bound and values above the upper bound with the upper bound.

In our dataset, the `Age` attribute contains outliers, and since it is numeric and operates within ranges, clipping is the preferred approach for handling these outliers.

Q1 = df\_cleaned['Age'].quantile(0.25)

Q3 = df\_cleaned['Age'].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

print(f"Lower Bound for Age: {lower\_bound}, Upper Bound for Age: {upper\_bound}")

# Clip Age values to handle outliers

df\_cleaned['Age'] = df\_cleaned['Age'].clip(lower=lower\_bound, upper=upper\_bound)

# Verify the changes

print("Summary statistics for Age after handling outliers:")

print(df\_cleaned['Age'].describe())

# Plot the updated boxplot for Age

plt.figure(figsize=(8, 5))

sns.boxplot(x=df\_cleaned['Age'])

plt.title("Boxplot of Age After Removing Outliers")

plt.show()

A screenshot of a computer

Description automatically generated

A blue rectangular bar graph

Description automatically generated with medium confidence

for the binary columns (Obesity, Irritabbility and Genital thrush)

we will use the logistic regression

# for the binary columns (Obesity, Irritabbility and Genital thrush)

# we will use the logistic regression

from sklearn.linear\_model import LogisticRegression

import seaborn as sns

import matplotlib.pyplot as plt

# Define a function to detect outliers in binary columns

def detect\_outliers\_binary(df, column):

    # Outliers for binary columns are values other than 0 or 1

    return (df[column] != 0) & (df[column] != 1)

# Define a function to handle binary outliers using Logistic Regression

def handle\_binary\_outliers\_with\_regression(df, target\_column):

    # Detect outliers in the target column

    outliers = detect\_outliers\_binary(df, target\_column)

    # Check if there are any outliers

    if outliers.sum() == 0:

        print(f"No outliers detected in column '{target\_column}'. Skipping...")

        return

    # Prepare training data (non-outlier rows)

    predictors = df.drop(columns=[target\_column])  # Drop target column for predictors

    target = df[target\_column]

    predictors\_non\_outliers = predictors[~outliers]  # Non-outlier predictors

    target\_non\_outliers = target[~outliers]  # Non-outlier target values

    # Train logistic regression on non-outlier data

    log\_reg = LogisticRegression()

    log\_reg.fit(predictors\_non\_outliers, target\_non\_outliers)

    # Predict values for outliers

    predictors\_outliers = predictors[outliers]  # Outlier predictors

    if predictors\_outliers.empty:

        print(f"No valid predictors for outliers in column '{target\_column}'. Skipping...")

        return

    predicted\_values = log\_reg.predict(predictors\_outliers)

    # Replace outlier values with predicted values

    df.loc[outliers, target\_column] = predicted\_values

# List of binary columns with outliers to fix

binary\_columns = ['Obesity', 'Irritability', 'Genital thrush']

# Apply logistic regression-based outlier handling for each binary column in df\_cleaned

for column in binary\_columns:

    print(f"Fixing outliers for {column}...")

    handle\_binary\_outliers\_with\_regression(df\_cleaned, column)

# Display the updated dataset

print("Binary outliers handled.")

print(df\_cleaned[binary\_columns].head())

# Plot the boxplots for each binary column after handling outliers

for column in binary\_columns:

    plt.figure(figsize=(8, 5))

    sns.boxplot(x=df\_cleaned[column])

    plt.title(f"Boxplot of {column} After Handling Outliers")

    plt.show()

A screenshot of a computer program

Description automatically generated

Final Clean shape :

A screen shot of a computer

Description automatically generated

1. Data Visualization

In this code, we are shuffling the dataset to ensure the data is randomized before splitting it into training and testing sets.

This helps prevent the issue of class domination in the testing set, ensuring a fair evaluation of the model.

# View the first 5 rows before shuffling

print("Before shuffling:")

display(df.head())

# Shuffle the DataFrame

shuffled\_df = df.sample(frac=1, random\_state=42).reset\_index(drop=True)

# View the first 5 rows after shuffling

print("\nAfter shuffling:")

display(shuffled\_df.head())



1. Data Transformation:

encoder = LabelEncoder()

for i in df\_cleaned.columns[1:]:

    df\_cleaned[i]= encoder.fit\_transform(df\_cleaned[i])

df\_cleaned.head()

A black screen with white text

Description automatically generated

1. Splitting:

x= df\_cleaned.iloc[:,:-1]

y= df\_cleaned['class']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, train\_size=0.8,random\_state= 0)

The first two lines separate the features (x) and the target variable (y) from the dataset, with x containing all columns except the last and y containing the 'class' column. The data is then split into training (80%) and testing (20%) sets.

A computer screen shot of a program code

Description automatically generated

Class distribution:

A graph of a class distribution

Description automatically generated

A black screen with white text

Description automatically generated

**3. Choosing our model :**

**Random forest**

**Heat map :**

numeric\_columns = df.select\_dtypes(include=['number']).columns

df[numeric\_columns] = df[numeric\_columns].fillna(df[numeric\_columns].mean())

# Encode categorical variables (if needed)

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

for col in df.select\_dtypes(include=['object']).columns:

    df[col] = encoder.fit\_transform(df[col])

# Compute the correlation matrix

corr\_matrix = df.corr()

# Display the heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')

plt.title("Correlation Heatmap")

plt.show()

A diagram of a heatmap

Description automatically generated

The heatmap shows the correlation between attributes, with the highest correlation being 0.67 between polyuria and the target class, indicating that polyuria is an important feature for class detection. The other correlations are relatively low, suggesting no redundant data.

Since tree-based models select features based on their contribution to reducing impurity during splits, rather than their correlation, there is no need to drop or merge any attributes.

To better visualize the feature importance

importance = rf.feature\_importances\_

features = x.columns

plt.figure(figsize=(10, 6))

sns.barplot(x=importance, y=features)

plt.title("Feature Importance")

plt.xlabel("Importance Score")

plt.ylabel("Features")

plt.show()

A graph of a bar graph

Description automatically generated

With polydipsia polyuria and age being the most contributors  
some relevant EDA analysis:

1. **Polyuria vs Class**

A graph of different colored rectangular bars

Description automatically generated with medium confidence

This shows the relationship between the symptom **Polyuria** (frequent urination) and diabetes diagnosis (class).

Most individuals with Polyuria are diabetic (Positive), while most without Polyuria are non-diabetic (Negative).

**Insight:** Polyuria is a **strong predictor** of diabetes, as a significant proportion of positive cases exhibit this symptom. This reinforces the importance of this feature in the model's predictions.

1. **Age Distribution**

A graph of age distribution

Description automatically generated

This histogram represents the distribution of ages in the dataset.

The age range is from approximately 20 to 90 years, with most individuals between 40 and 60 years old.

**Insight**: The dataset has a normal age distribution, with the majority of cases occurring in middle-aged individuals. Older individuals may have a higher risk of diabetes, which could make Age an important feature for predictions.

# Selecting the Best ML Model Type:

Based on the goal of predicting diabetes risk (positive/negative), the problem is best suited for **Supervised Learning**.

Justification to why Supervised Learning:

* **Target Variable:** The dataset has a clear target variable (class), which labels individuals as "Positive" (diabetes) or "Negative" (no diabetes).
* **Objective**: Predict specific outcomes (whether an individual is at risk of diabetes) based on the provided features.
* **Learning Type**: Supervised Learning is appropriate since labeled data is available to train the model.

# Algorithm Selection and Justification:

**Random Forest**: An ensemble model that improves accuracy by combining multiple decision trees.

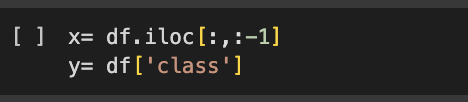
**Why?**

* **Handles Mixed Data Types**:
  + Works effectively with both categorical (e.g., Polyuria, Polydipsia) and numerical features (e.g., Age) without extensive preprocessing.
* **Reduces Overfitting**:
  + Uses an ensemble of decision trees, averaging results to improve generalization and robustness against overfitting.
* **Identifies Key Features**:
  + Provides feature importance scores, highlighting the most influential factors for diabetes risk, aiding in actionable healthcare decisions.
* **Captures Complex Relationships**:
  + Models non-linear relationships between features and the target variable, which is crucial in healthcare datasets.
* **Better Alternative to Other Models**:
  + Logistic Regression assumes linear relationships, which may not capture the complexity of non-linear patterns in the dataset, such as interactions between symptoms like Polyuria and Polydipsia.
  + Decision trees are prone to overfitting, especially with categorical features and small datasets, reducing their ability to generalize.
  + Gradient Boosting provides strong performance but requires extensive tuning, which isn't ideal for a dataset of this size and complexity.

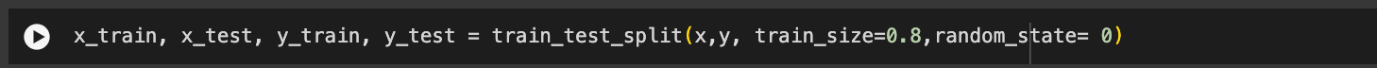
# Implementation: Training the model:

**Model Training and Evaluation**:

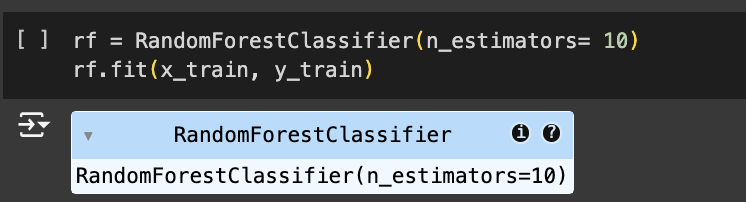
1. Feature and Target Selection:



1. Data splitting: Splits the data into:
   1. **Training Set**: 80% of the data, used to train the model.
   2. **Test Set**: 20% of the data, used to evaluate the model's performance.



1. Model Initialization and Training:
   1. Initializes a Random Forest model with 10 decision trees.
   2. Trains the Random Forest model using the training data (x\_train and y\_train).



**Performance Assessment:**

**Before cleaning:**

A screenshot of a computer program

Description automatically generated

# Assuming y\_test and y\_pred are already defined

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Plotting the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

A blue squares with white text

Description automatically generated

To study the tradeoffs we visualized **ROC (Receiver Operating Characteristic) Curve** and the **AUC (Area Under the Curve)**

* The ROC curve shows the trade-off between the True Positive Rate (TPR) and False Positive Rate (FPR) at various thresholds.
* The AUC score quantifies the model's ability to distinguish between classes (1 being perfect, 0.5 being random).

y\_pred\_proba = rf.predict\_proba(x\_test)[:, 1]  # Assuming 'rf' is your Random Forest model

# Compute ROC curve and AUC score

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)  # False Positive Rate, True Positive Rate

roc\_auc = auc(fpr, tpr)  # Area Under the Curve

# Plot the ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Dashed diagonal line (random classifier)

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

plt.grid()

A graph with a line

Description automatically generated

The ROC curve is plotted, and its proximity to the top-left corner signifies strong model performance. The dashed diagonal line represents a random classifier for comparison.

Then we re plotted the ROC Curve to highlight specific threshold points to provide more detailed insights into the model's performance.

# Compute ROC curve and AUC score

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

roc\_auc = auc(fpr, tpr)

# Plot the ROC curve

plt.figure(figsize=(10, 6))

plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--', label="Random Classifier")  # Dashed diagonal line

plt.scatter(fpr, tpr, color='red', label='Threshold Points')  # Highlight TPR and FPR points

# Highlight a specific point on the curve (example: midpoint)

highlight\_index = len(fpr) // 2  # Example: Highlighting the middle point

plt.scatter(fpr[highlight\_index], tpr[highlight\_index], color='green', s=100, label='Highlighted Point')

plt.annotate(

    f'TPR={tpr[highlight\_index]:.2f}, FPR={fpr[highlight\_index]:.2f}',

    (fpr[highlight\_index], tpr[highlight\_index]),

    textcoords="offset points",

    xytext=(-30, 10),

    ha='center',

    color='green'

)

# Add axis labels, title, and legend

plt.xlabel('False Positive Rate (FPR)')

plt.ylabel('True Positive Rate (TPR)')

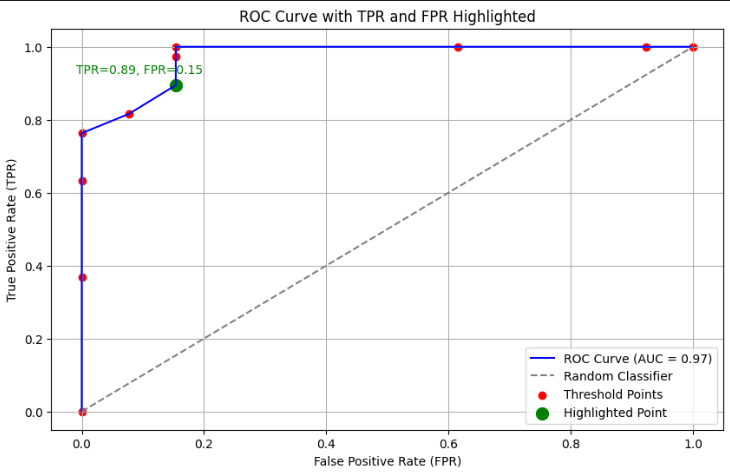
plt.title('ROC Curve with TPR and FPR Highlighted')

plt.legend(loc='lower right')

plt.grid()

# Show the plot

plt.show()



The ROC curve demonstrates excellent model performance with an AUC of 0.97, indicating a strong ability to distinguish between classes. The curve rises sharply towards the top-left corner, reflecting a high True Positive Rate (TPR) and low False Positive Rate (FPR). The highlighted point (TPR=0.89, FPR=0.15) shows the model balances accuracy well at this threshold, making it a potentially optimal decision point. Overall, the model significantly outperforms a random classifier, as seen by its distance from the diagonal line.

**After cleaning we had over fitting due to class imbalance   
A screenshot of a computer screen

Description automatically generated**

**We studied this imbalance**

# Verify the distribution of classes in each split

print("Original class distribution:")

print(y.value\_counts())

print("\nTraining set class distribution:")

print(y\_train.value\_counts())

print("\nTesting set class distribution:")

print(y\_test.value\_counts())

A screenshot of a computer program

Description automatically generated

Optimized it using SMOTE

from imblearn.over\_sampling import SMOTE

smote = SMOTE(random\_state=0)

x\_train\_balanced, y\_train\_balanced = smote.fit\_resample(x\_train, y\_train)

print("Class distribution after SMOTE:\n", y\_train\_balanced.value\_counts())

A black screen with white text

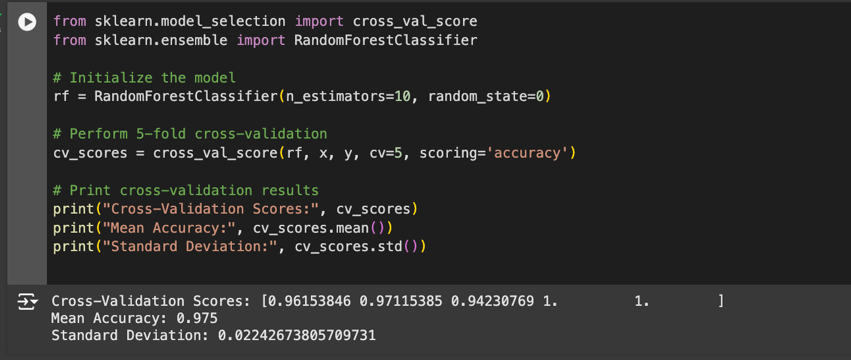
Description automatically generated

# RandomForestClassifier

A screenshot of a graph

Description automatically generated

**Optimization using cross-validation.**



Cross-Validation Scores:

1. [0.96153846, 0.97115385, 0.94230769, 1.0, 1.0]

These represent the accuracy scores for each of the 5 folds in cross-validation.

Accuracy remains consistently high across all folds, ranging from **94.2% to 100%**, indicating the model generalizes well.

1. **Mean Accuracy**: **0.975 (97.5%)**

The average accuracy across all folds. This reflects the model's overall performance and confirms its reliability.

1. **Standard Deviation**: **0.0224 (2.24%)**

This low standard deviation shows that the accuracy is stable and consistent across the folds, meaning the model isn't highly sensitive to different subsets of data.

**Radom Forest visualization**

**A diagram of a diagram

Description automatically generated with medium confidence**

# Trying different models:

**LogisticRegression**

from sklearn.ensemble import VotingClassifier

# Initialize individual models

logistic\_model = LogisticRegression()

decision\_tree\_model = DecisionTreeClassifier()

random\_forest\_model = RandomForestClassifier()

# Combine the models into a VotingClassifier (using soft voting)

ensemble\_model = VotingClassifier(

    estimators=[

        ('Logistic Regression', logistic\_model),

        ('Decision Tree', decision\_tree\_model),

        ('Random Forest', random\_forest\_model)

    ],

    voting='soft'  # Use soft voting to average probabilities

)

# Train the ensemble model on the training data

ensemble\_model.fit(x\_train, y\_train)

# Make predictions on the test set

y\_pred = ensemble\_model.predict(x\_test)

# Evaluate the model

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Calculate and print accuracy

test\_accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Testing Accuracy: {test\_accuracy:.2f}")

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(8, 6))

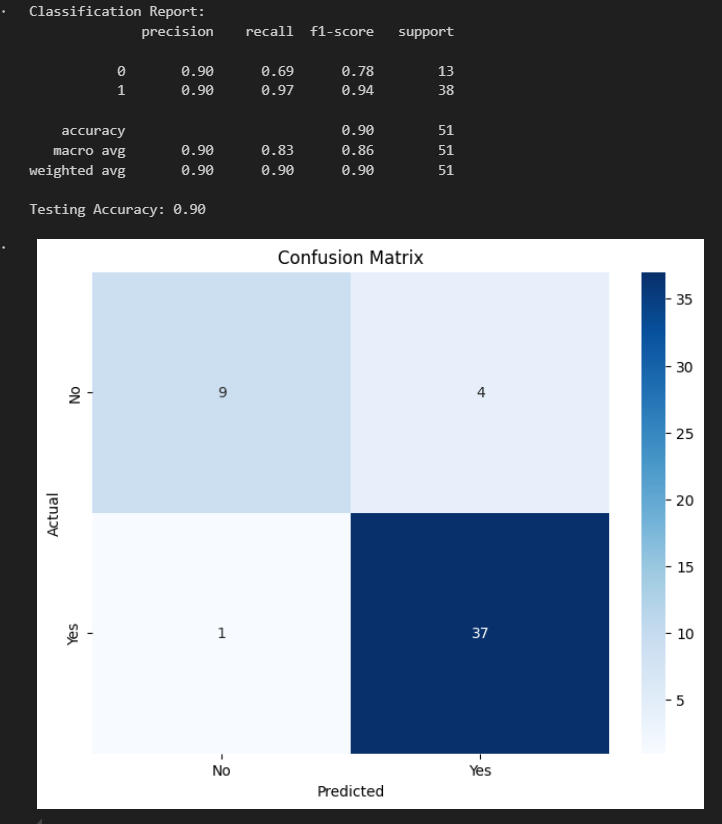
sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()



After SMOTE

# LogisticRegressi

# A screenshot of a graph Description automatically generated

Random forest did better Better optimization

# DecisionTreeClassifier

# Initialize the Decision Tree Classifier

decision\_tree\_model = DecisionTreeClassifier()

# Train the model on the training data

decision\_tree\_model.fit(x\_train, y\_train)

# Make predictions on the test set

y\_pred = decision\_tree\_model.predict(x\_test)

# Evaluate the model

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Calculate and print accuracy

test\_accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Testing Accuracy: {test\_accuracy:.2f}")

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

A screenshot of a graph

Description automatically generated

# Conclusion random forest yielded the best results

# Business Recommendations and Decision-Making

**Translating Insights:**

* The Random Forest model identified key features like **Polyuria**, **Polydipsia**, **Age**, and **Sudden Weight Loss** as the most important predictors of diabetes risk.
* The imbalance in the target variable suggests that most cases are diabetic, emphasizing the need for tailored preventive measures.
* **Recommendations**:
  + **Symptom-Based Screening**:
    - Use key features like **Polyuria** and **Polydipsia** to develop targeted screening tools.
    - Implement symptom checklists in healthcare centers to identify high-risk individuals early.
  + **Age-Focused Outreach**:
    - Prioritize individuals aged 40-60 for awareness campaigns and regular health checkups, as this age group shows the highest concentration of diabetes cases.