GlucoSense : AI-Powered Diabetes Detection for Early Intervention

A Project Report

Submitted to

Under the supervision of

RAVI

Submitted By

MAHESHKUMAR S(730923632022)

Excel Engineering College

NH-544 Komarapalayam , namakkal-dt.

Table of Contents

Abstract

Diabetes is a major global health challenge, impacting quality of life and straining healthcare systems. Early detection is crucial to prevent severe complications like cardiovascular disease and kidney failure. Traditional diagnostic methods can be inaccessible, particularly in underserved areas. GlucoSense: AI-Powered Diabetes Detection addresses this gap by leveraging AI for efficient and accurate diabetes risk prediction.

Using a dataset with health indicators like glucose levels, BMI, and insulin, exploratory data analysis (EDA) identified glucose levels and BMI as key predictors. After rigorous preprocessing, machine learning models such as Logistic Regression, Random Forest, and SVM were evaluated. Random Forest emerged as the top performer with an F1 score of 0.957, demonstrating its reliability for real-world use.

GlucoSense offers a scalable, accessible solution for early diabetes detection, potentially deployable in healthcare systems and mobile apps. Future efforts will focus on expanding datasets and integrating deep learning to enhance accuracy, paving the way for AI-driven personalized healthcare solutions.

Introduction

Diabetes is a chronic condition affecting millions worldwide, contributing significantly to morbidity and mortality. In 2019, the WHO estimated 463 million adults lived with diabetes, a figure expected to rise. Early detection is critical to managing the disease and reducing complications like cardiovascular disease and kidney failure. However, traditional diagnostic methods, though accurate, are time-consuming and often inaccessible in underserved areas.

The GlucoSense project leverages AI to revolutionize diabetes detection, using machine learning algorithms to predict risk based on patient data like glucose levels, BMI, age, and insulin. After pre-processing the data, various models, including Random Forest, were evaluated. Random Forest demonstrated high accuracy and balanced performance, making it ideal for real-world applications.

GlucoSense offers a scalable, cost-effective solution to democratize diabetes detection, particularly in resource-limited settings. This initiative underscores AI's potential to tackle global health challenges and improve access to preventive care.

Problem Statement

Early detection and intervention are critical to managing diabetes effectively and preventing complications. However, traditional diagnostic methods such as fasting glucose tests, oral glucose tolerance tests, and HbA1c measurements often require access to well-equipped laboratories and trained personnel. These resources are not readily available in rural or underserved areas, leading to delayed diagnoses and treatment. Additionally, many individuals with pre-diabetes remain undiagnosed, missing the opportunity to receive early care that could potentially reverse or slow the progression of the disease.

The lack of scalable and accessible diagnostic tools exacerbates the problem, leaving healthcare providers unable to address the increasing demand for early detection. There is a pressing need for innovative, efficient, and cost-effective solutions that can predict diabetes risk without relying solely on traditional laboratory tests. Predictive models powered by artificial intelligence (AI) offer a promising alternative, leveraging existing health data to identify individuals at risk and prioritize them for further screening or preventive interventions.

This project aims to address these challenges by developing a machine learning-based system capable of predicting diabetes risk using easily accessible patient data. By analyzing features such as glucose levels, BMI, age, and insulin levels, the proposed model seeks to provide accurate predictions that can complement traditional diagnostic methods. This approach has the potential to bridge the gap in healthcare accessibility, particularly for underserved populations, and enable timely interventions that can significantly improve patient outcomes.

The implementation of GlucoSense aligns with the broader goal of leveraging AI in healthcare to address critical challenges. By deploying this model in real-world settings, such as clinics, mobile health applications, or community health programs, it can serve as a practical and scalable solution to combat the diabetes epidemic. Ultimately, this project seeks to contribute to a future where technology-driven solutions enhance healthcare accessibility, efficiency, and equity.

Data Collection and Dataset Details

Data Source:

For the GlucoSense project, the data used for analysis and model development was sourced from Kaggle, utilizing the **"diabetes_dataset.csv"**, which was likely obtained from publicly available repositories, such as Kaggle or research publications, focusing on diabetes risk assessment.

- **Size:**
  - Total Records: 520 (original dataset).
  - Post-Cleaning: 250 records (after handling missing values and outliers).
  - Features: 16 columns (15 independent features and 1 target variable).
- **Objective:** The dataset was designed to classify individuals as diabetic or non-diabetic based on various health attributes.

Content of "diabetes_dataset.csv" file

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Age	Gender	Polyuria	Polydipsia	sudden we	weakness	Polyphagia	Genital thr	visual blurı	Itching	Irritability	delayed he	partial par	muscle stif	Alopecia	Obesity	class
2	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Positive
3	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	No	Yes	No	Positive
4	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Positive
5	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Positive
6	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Positive
7	55	Male	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Positive
8	57	Male	Yes	Yes	No	Yes	Yes	Yes	No	No	No	Yes	Yes	No	No	No	Positive
9	66	Male	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No	No	Positive
10	67	Male	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Positive
11	70	Male	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	Yes	No	Positive
12	44	Male	Yes	Yes	No	Yes	No	Yes	No	No	Yes	Yes	No	Yes	Yes	No	Positive
13	38	Male	Yes	Yes	No	No	Yes	Yes	No	Yes	No	Yes	No	Yes	No	No	Positive
14	35	Male	Yes	No	No	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No	Positive
15	61	Male	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes	Positive
16	60	Male	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	No	No	Positive
17	58	Male	Yes	Yes	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	No	No	Positive
18	54	Male	Yes	Yes	Yes	Yes	No	Yes	No	No	No	Yes	No	Yes	No	No	Positive
19	67	Male	No	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Positive
20	66	Male	Yes	Yes	No	Yes	Yes	No	Yes	No	No	No	Yes	Yes	No	No	Positive
21	43	Male	Yes	Yes	Yes	Yes	No	Yes	No	No	No	No	No	No	No	No	Positive
22	62	Male	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	No	Yes	Yes	No	No	Positive
23	54	Male	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	Positive
24	39	Male	Yes	No	Yes	No	No	Yes	No	Yes	Yes	No	No	No	Yes	No	Positive
25	48	Male	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No	No	No	No	Positive
26	58	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes	No	Yes	Positive

diabetes_risk_prediction_datase

Features in the Dataset:

The dataset contains a mix of categorical and numerical features. Below is a detailed description of the key attributes:

- **Age (Numerical):** Represents the age of the individual in years.
- **Gender (Categorical):** Indicates the gender of the individual (Male/Female).
- **Polyuria (Categorical):** Whether the individual experiences excessive urination (Yes/No).
- **Polydipsia (Categorical):** Indicates excessive thirst (Yes/No).
- **Sudden Weight Loss (Categorical):** Whether the person experienced sudden weight loss (Yes/No).
- **Weakness (Categorical):** Reports general body weakness (Yes/No).
- **Polyphagia (Categorical):** Whether the individual experiences excessive hunger (Yes/No).
- **Genital Thrush (Categorical):** Indicates the presence of genital thrush (Yes/No).
- **Visual Blurring (Categorical):** Whether the person experiences blurred vision (Yes/No).
- **Itching (Categorical):** Indicates skin itching (Yes/No).
- **Irritability (Categorical):** Reports irritability symptoms (Yes/No).
- **Delayed Healing (Categorical):** Indicates delayed wound healing (Yes/No).
- **Partial Paresis (Categorical):** Reports partial paresis or muscle weakness (Yes/No).
- **Muscle Stiffness (Categorical):** Whether muscle stiffness is present (Yes/No).
- **Alopecia (Categorical):** Indicates hair loss or alopecia (Yes/No).
- **Target Variable - Class (Categorical):**
    - Indicates whether the individual has diabetes:
        - Positive (Diabetic)
        - Negative (Non-Diabetic)

Data Preprocessing (EDA) and its Outcomes

Import libraries:

As the first step we need to import libraries. These libraries provide essential tools for handling data, performing computations, creating visualizations, and implementing machine learning algorithms.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE
```

Load the dataset:

```
#load the dataset
df = pd.read_csv('diabetes_dataset.csv')
```

Basic Information:

The dataset consists of 520 rows and 17 columns. There are no missing values in the dataset.

```
#information about dataset
print("Dataset Information:")
print(data.info())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Age                 520 non-null    int64 
 1   Gender              520 non-null    object
 2   Polyuria            520 non-null    object
 3   Polydipsia          520 non-null    object
 4   sudden weight loss  520 non-null    object
 5   weakness            520 non-null    object
 6   Polyphagia          520 non-null    object
 7   Genital thrush      520 non-null    object
 8   visual blurring     520 non-null    object
 9   Itching             520 non-null    object
 10  Irritability        520 non-null    object
 11  delayed healing     520 non-null    object
 12  partial paresis     520 non-null    object
 13  muscle stiffness    520 non-null    object
 14  Alopecia            520 non-null    object
 15  Obesity             520 non-null    object
 16  class               520 non-null    object
dtypes: int64(1), object(16)
memory usage: 69.2+ KB
None
```

Drop the Duplicates:

The above dataset has 269 duplicate rows. Dropping duplicates is essential for ensuring data integrity and improving the accuracy of analyses. Duplicates can lead to biased results, as they may skew statistical calculations, machine learning models, and data visualizations by overrepresenting certain values. Removing duplicates helps in obtaining cleaner, more reliable data, ultimately leading to more accurate insights and predictions.

After dropping the duplicates, the statistics is as follows:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 251 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Age                 251 non-null    int64 
 1   Gender              251 non-null    object
 2   Polyuria            251 non-null    object
 3   Polydipsia          251 non-null    object
 4   sudden weight loss  251 non-null    object
 5   weakness            251 non-null    object
 6   Polyphagia          251 non-null    object
 7   Genital thrush      251 non-null    object
 8   visual blurring     251 non-null    object
 9   Itching             251 non-null    object
 10  Irritability        251 non-null    object
 11  delayed healing     251 non-null    object
 12  partial paresis     251 non-null    object
 13  muscle stiffness    251 non-null    object
 14  Alopecia            251 non-null    object
 15  Obesity             251 non-null    object
 16  class               251 non-null    object
dtypes: int64(1), object(16)
memory usage: 35.3+ KB
```

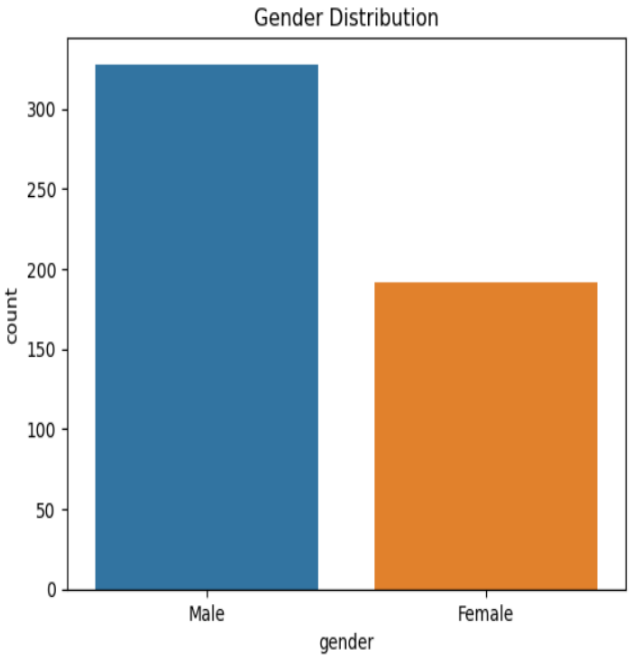
Univariate analysis:

Univariate analysis involves examining a single variable to understand its distribution, central tendency (mean, median, mode), and spread (variance, standard deviation). It uses visualizations like histograms and box plots to identify patterns, outliers, and data quality issues, helping inform decisions for further analysis and preprocessing.

Distribution of Age:

Gender Distribution

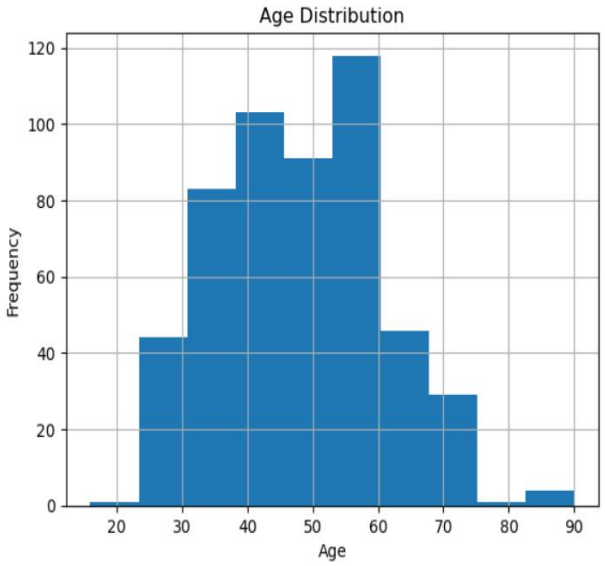
```
[7]: sns.countplot(x='gender', data=df)
     plt.title('Gender Distribution')
     plt.show()
```



Bivariate analysis:

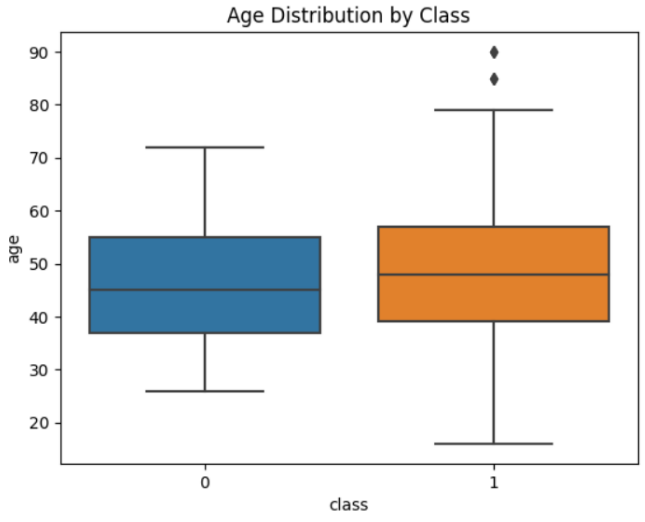
Age Distribution Histogram :

[8]:
```
df['age'].hist(bins=10)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution')
plt.show()
```



Boxplot of Age Distribution by Diabetes Class

[14]:
```
sns.boxplot(x='class', y='age', data=df)
plt.title('Age Distribution by Class')
plt.show()
```



Age Distribution by Diabetes Classification :

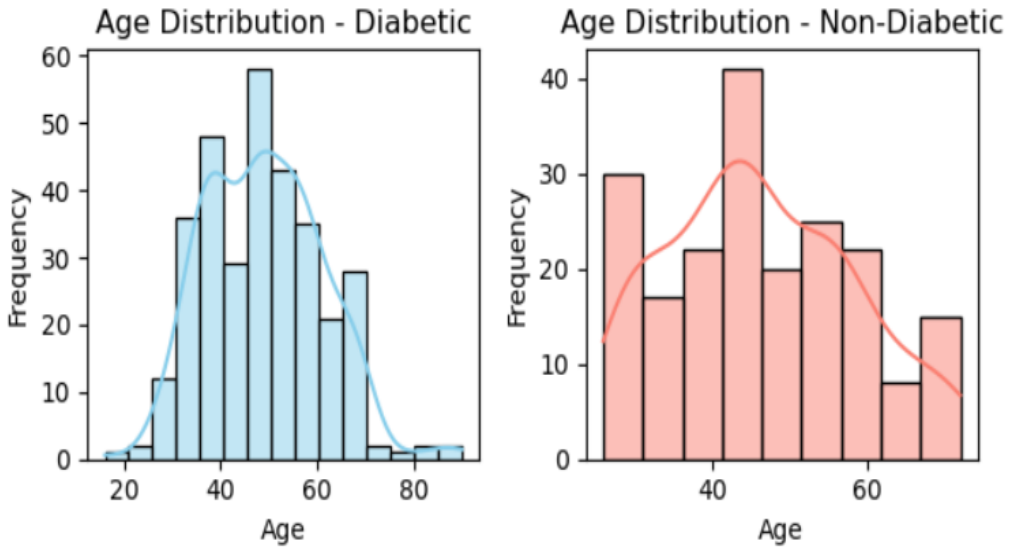
```
# Age distribution by diabetes classification
diabetic_data = df[df['class'] == 1]
non_diabetic_data = df[df['class'] == 0]

plt.figure(figsize=(6, 3))

# Diabetic individuals
plt.subplot(1, 2, 1)
sns.histplot(diabetic_data['age'], kde=True, color='skyblue')
plt.title("Age Distribution - Diabetic")
plt.xlabel("Age")
plt.ylabel("Frequency")

# Non-diabetic individuals
plt.subplot(1, 2, 2)
sns.histplot(non_diabetic_data['age'], kde=True, color='salmon')
plt.title("Age Distribution - Non-Diabetic")
plt.xlabel("Age")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```



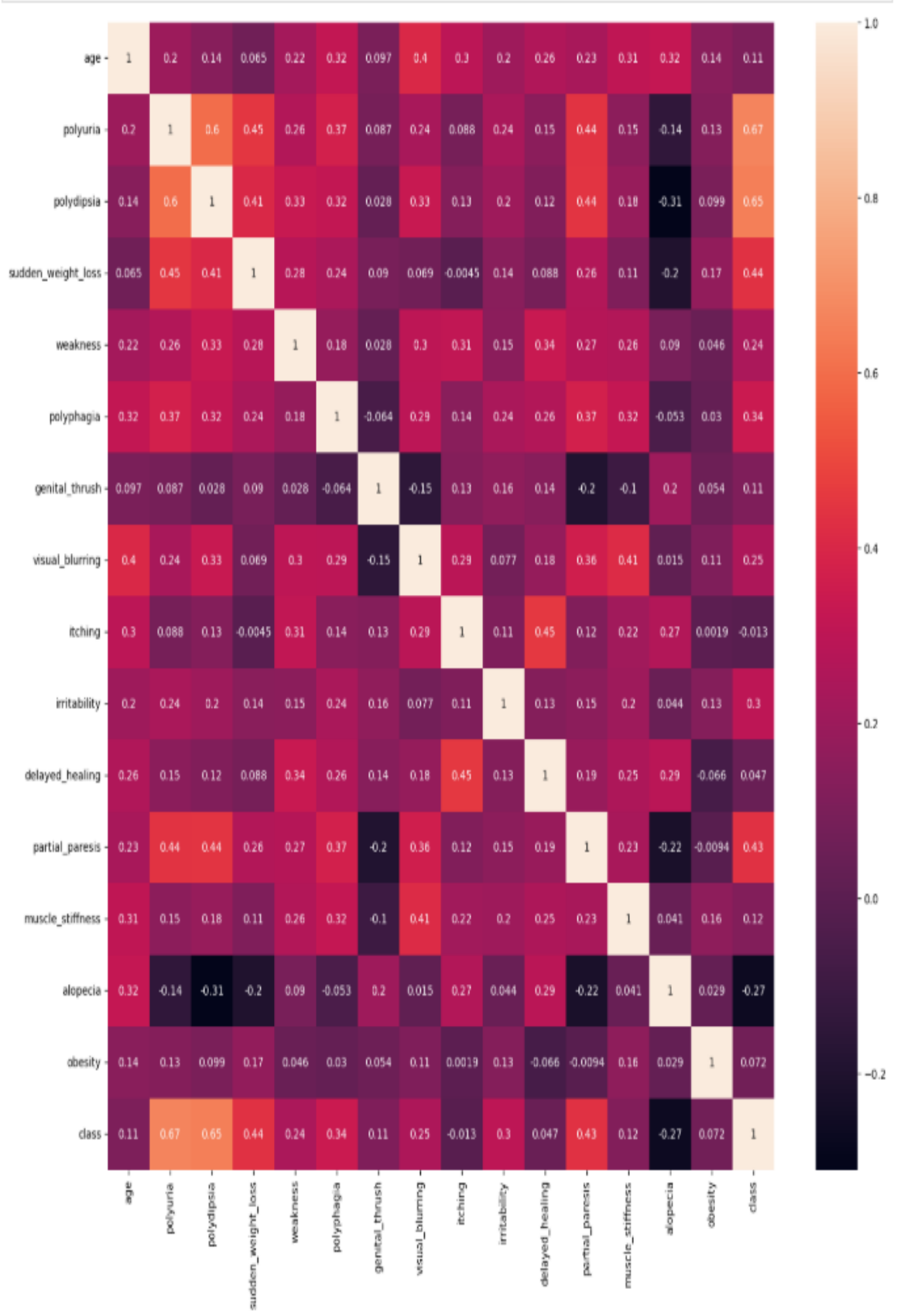
Multivariate Analysis

```
# Filter for numeric columns only
numeric_df = df.select_dtypes(include=['number'])  # Select only numeric columns

# Calculate the correlation matrix for numeric data
corr_matrix = numeric_df.corr()

# Plot heatmap
plt.figure(figsize=(16, 16))
sns.heatmap(corr_matrix, annot=True)
plt.show()
```

Heatmap of Correlation Matrix for Numeric Features



Key Insights:

The features polyuria, polydipsia, sudden weight loss, weakness, and genital thrush show strong distinctions between diabetic and non-diabetic cases, suggesting these are particularly indicative of diabetes. Visual blurring, itching, delayed healing, and polyphagia also demonstrate significant differences and are likely valuable indicators. Features like irritability, partial paresis, and muscle stiffness are moderately associated with diabetes. Gender, alopecia, and obesity appear less indicative on their own but may be relevant when combined with other features.

Correlation matrix:

A correlation matrix is a table that shows the pairwise correlation coefficients between variables in a dataset. It helps identify relationships between variables, indicating how strongly they are related, with values ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation), and 0 indicating no correlation.

Correlation of Obesity with Other Numerical Features

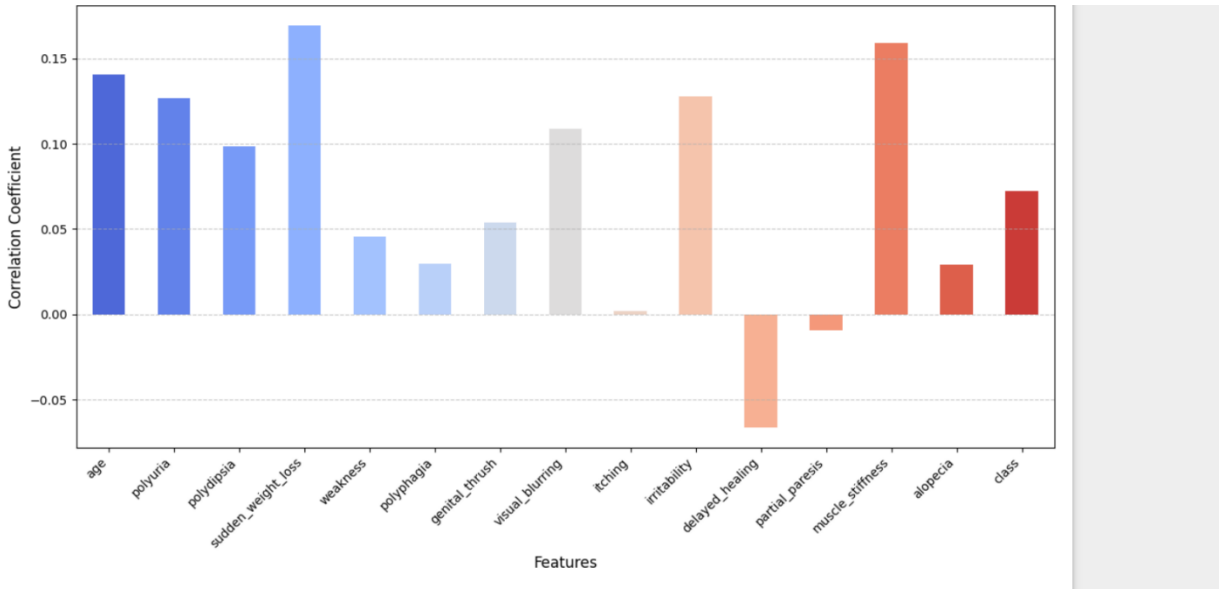
```
# Selecting numerical features
numerical_features = df.select_dtypes(include=np.number).columns
obesity_correlations = df[numerical_features].corr()['obesity'].drop('obesity')

# Setting figure size and colors
plt.figure(figsize=(12, 7))
colors = sns.color_palette("coolwarm", len(obesity_correlations))

# Plotting the bar plot with color gradient
obesity_correlations.plot(kind='bar', color=colors)
plt.title('Correlation of Obesity with Other Numerical Features', fontsize=16, fontweight='bold')
plt.xlabel('Features', fontsize=12)
plt.ylabel('Correlation Coefficient', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Show the plot
plt.show()
```

Correlation of Obesity with Other Numerical Features



High Correlations with Diabetes Class:
Polyuria (0.67), Polydipsia (0.65), and sudden weight loss (0.44) have a strong positive correlation with the diabetes class, indicating these symptoms are highly associated with diabetes in the dataset.
Partial paresis (0.43) and Polyphagia (0.34) also show moderate correlations with the diabetes class.

Inter-relationships Between Symptoms:
Polyuria and Polydipsia show a strong correlation (0.60), suggesting these symptoms frequently occur together.
Visual blurring is moderately correlated with muscle stiffness (0.41) and partial paresis (0.36), indicating a possible association between these symptoms.

Negative Correlations:
Alopecia has a notable negative correlation with Polydipsia (-0.31) and the diabetes class (-0.27), suggesting that its presence might be less common among individuals with diabetes in this dataset.

Symptom Frequency in Positive Diabetes Cases:

Prevalence Across Age Groups:

Symptoms are more frequently observed in the 40–60 age range for individuals with diabetes, indicating a higher incidence of diabetes-related symptoms in middle-aged groups.

Top Symptoms:

Polyuria and Polydipsia:

These symptoms are consistently higher across most age groups, particularly among middle-aged individuals, reinforcing their importance as diabetes indicators.

Weakness and sudden weight loss:

These symptoms show a noticeable frequency, especially in the 31-50 age groups, suggesting they could be predictive features for early detection in younger adults.

Elderly Groups:

The 60+ age group also shows a high frequency of symptoms, but with fewer cases compared to the middle-aged group, possibly due to fewer data samples or natural attrition of health in older age groups.

Symptom Frequency in Negative Diabetes Cases

Lower Overall Symptom Frequency:

For individuals without diabetes, symptoms like Polyuria and Polydipsia are notably less frequent across all age groups, confirming that these symptoms are strongly associated with diabetes.

Symptom Occurrence:

Symptoms such as Alopecia and Obesity show some presence across age groups even in negative cases, indicating that these factors might be influenced by other conditions not directly related to diabetes.

Comparative Age Group Trends

Higher Symptom Rates in Middle Age for Diabetics:

Positive diabetes cases exhibit a peak in symptoms in the 40-60 age range, while negative cases do not show such a peak, reinforcing that this age group is a critical period for diabetes management and intervention. Young and Elderly Groups: Both positive and negative cases show fewer symptoms in <20 and 70+ age groups, possibly due to fewer data points in these age brackets or lower incidenc

Feature Selection and dimension reduction approaches

Standardize data types:

Standardizing data types ensures consistency and simplifies data processing. The process involves, Converting columns with discrete values (e.g., "Yes", "No", or categories like "Male", "Female") into a standard format, such as the category type. This reduces memory usage and speeds up operations. Ensuring all numerical data, whether integers or floats, are stored in a consistent format (e.g., float). This prevents errors during computations or scaling. A consistent data type for similar data makes operations like statistical analysis, machine learning, and visualization more reliable and efficient.

```
for col in data.columns:
    if data[col].dtype == 'object':
        data[col] = data[col].astype(str)
    else:
        data[col] = data[col].astype(float)
```

```
# Identify columns with categorical data types ('object' or 'category')
categorical_cols = data.select_dtypes(include=['object', 'category']).columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()  # Initialize a new LabelEncoder
    data[col] = le.fit_transform(data[col])  # Apply label encoding to the column
    label_encoders[col] = le  # Store the encoder in the dictionary
# Convert all columns to float data type for numerical computations
data = data.astype(float)
```

```
data.head()
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity	class
0	40	1	0	1	0	1	0	0	0	1	0	1	0	1	1	1	1
1	58	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	1
2	41	1	1	0	0	1	1	0	0	1	0	1	0	1	1	0	1
3	45	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	1
4	60	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1

Feature selection:

- **Polyuria** and **Polydipsia** emerge as the most significant predictors, with the highest mutual information scores. These are common and early indicators of diabetes, with polyuria (excessive urination) and polydipsia (increased thirst) strongly linked to high blood sugar levels.
- Features like **Partial Paresis** (muscle weakness), **Sudden Weight Loss**, and **Gender** are also important but show a slightly lower correlation compared to the top features. These symptoms are also frequently associated with undiagnosed or uncontrolled diabetes.
- Other features like **Irritability**, **Itching**, and **Weakness** contribute moderately to the prediction of diabetes, reflecting the physical effects that may occur due to insulin resistance or high blood glucose levels.
- **Polyphagia** (excessive hunger) and **Visual Blurring** also show a moderate correlation with diabetes risk, reinforcing the symptoms that can aid in early-stage diagnosis.
- Finally, features like **Age**, **Muscle Stiffness**, and others like **Genital Thrust**, **Delayed Healing**, **Alopecia**, and **Obesity** play a lesser role in the model but are still relevant in the broader context of health and disease progression.

Feature Selection

a) Recursive Feature Elimination (RFE)

[29]:
```
# importing important libraries
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

# Perform RFE with Random Forest
rfe = RFE(RandomForestClassifier(random_state=42), n_features_to_select=10)
rfe.fit(X_scaled, y)

# Select important features
selected_features = X.columns[rfe.support_]
print("Selected features by RFE:", selected_features)
X_rfe = X_scaled[:, rfe.support_]
```

```
Selected features by RFE: Index(['age', 'gender', 'polyuria', 'polydipsia', 'sudden_weight_loss',
       'itching', 'irritability', 'delayed_healing', 'partial_paresis',
       'alopecia'],
      dtype='object')
```

b) LASSO for Feature Selection

[30]:
```
#imporing important libraries
from sklearn.linear_model import Lasso

# Apply LASSO
lasso = Lasso(alpha=0.01)
lasso.fit(X_scaled, y)

# Select non-zero coefficients
lasso_features = X.columns[lasso.coef_ != 0]
print("Selected features by LASSO:", lasso_features)
X_lasso = X_scaled[:, lasso.coef_ != 0]
```

```
Selected features by LASSO: Index(['gender', 'polyuria', 'polydipsia', 'sudden_weight_loss', 'polyphagia',
       'genital_thrush', 'visual_blurring', 'itching', 'irritability',
       'delayed_healing', 'partial_paresis', 'obesity'],
      dtype='object')
```

```
Top Features Based on Mutual Information:
Polyuria              0.260189
Polydipsia            0.170495
partial paresis       0.080930
sudden weight loss    0.074198
Gender                0.062457
Irritability          0.059472
Itching               0.050501
weakness              0.040430
visual blurring       0.037871
Polyphagia            0.037865
muscle stiffness      0.013687
Age                   0.000000
Genital thrush        0.000000
delayed healing       0.000000
Alopecia              0.000000
Obesity               0.000000
dtype: float64
```

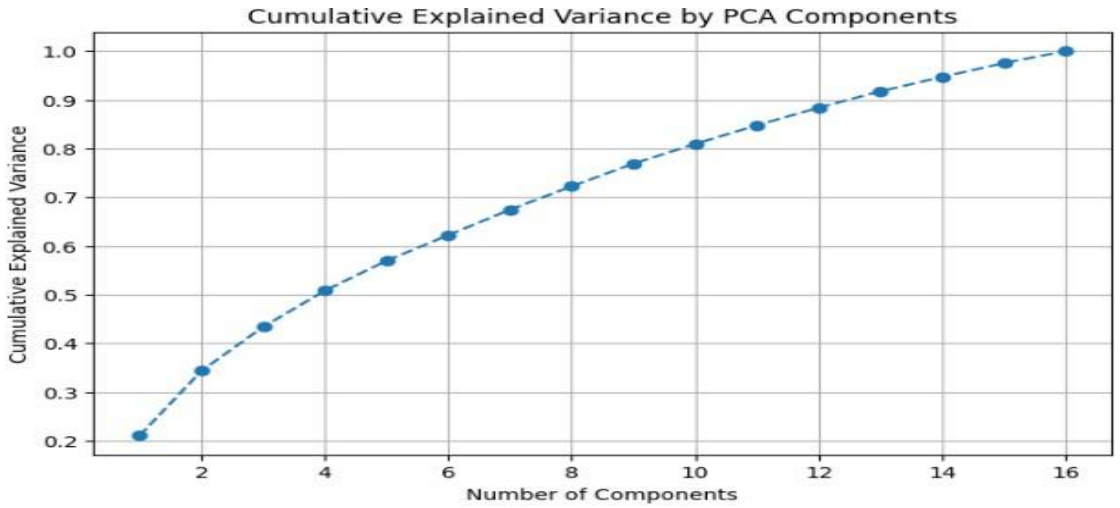
Dimensionality Reduction using PCA analysis:

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction. It transforms a high-dimensional dataset into a lower-dimensional space while retaining most of the dataset's variance. PCA is widely used in machine learning and data preprocessing to simplify data, reduce noise, and mitigate the curse of dimensionality.

```
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

# Plot the explained variance ratio
explained_variance = pca.explained_variance_ratio_.cumsum()
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o', linestyle='--')
plt.title('Cumulative Explained Variance by PCA')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid()
plt.show()

print(f"PCA reduced dimensions: {X_pca.shape[1]}")
```




```
Number of components to explain 95% variance: 15
Original dataset shape: (251, 16)
Reduced dataset shape: (251, 15)
```

Model Building Methodology

Define models:

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=500, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Support Vector Machine": SVC(random_state=42, probability=True),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "Extra Trees": ExtraTreesClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, eval_metric="logloss")
}
```

```
results = []
for model_name, model in models.items():
    # Train the model using the training data (X_train and y_train)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    # If the model supports probability predictions, get the probabilities for the positive class
    y_prob = model.predict_proba(X_test)[:, 1] if hasattr(model, 'predict_proba') else None
    # Calculate key performance metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_prob) if y_prob is not None else None
    # Store the performance metrics in a dictionary for each model
    results.append({
        'Model': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'AUC': auc
    })
# Convert the results list to a DataFrame for better readability
results_df = pd.DataFrame(results)
# Display the results DataFrame
results df
```

	Model	Accuracy	Precision	Recall	F1 Score	AUC
0	Logistic Regression	0.823529	0.825000	0.942857	0.880000	0.944643
1	Decision Tree	0.882353	0.891892	0.942857	0.916667	0.846429
2	Support Vector Machine	0.901961	0.894737	0.971429	0.931507	0.971429
3	Random Forest	0.921569	0.918919	0.971429	0.944444	0.975893
4	Gradient Boosting	0.901961	0.894737	0.971429	0.931507	0.951786
5	Extra Trees	0.921569	0.918919	0.971429	0.944444	0.989286
6	XGBoost	0.882353	0.891892	0.942857	0.916667	0.951786

Hyperparameter Tuning:

Logistic Regression:

Logistic regression is a statistical method used for binary classification problems, predicting the probability of an outcome belonging to one of two categories. It models the relationship between input features and the target variable using a sigmoid function, ensuring the output lies between 0 and 1.

a) Recursive Feature Elimination (RFE)

```
# importing important libraries
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

# Perform RFE with Random Forest
rfe = RFE(RandomForestClassifier(random_state=42), n_features_to_select=10)
rfe.fit(X_scaled, y)

# Select important features
selected_features = X.columns[rfe.support_]
print("Selected features by RFE:", selected_features)
X_rfe = X_scaled[:, rfe.support_]
```

```
Selected features by RFE: Index(['age', 'gender', 'polyuria', 'polydipsia', 'sudden_weight_loss',
       'itching', 'irritability', 'delayed_healing', 'partial_paresis',
       'alopecia'],
      dtype='object')
```

b) LASSO for Feature Selection

```
#imporing important libraries
from sklearn.linear_model import Lasso

# Apply LASSO
lasso = Lasso(alpha=0.01)
lasso.fit(X_scaled, y)

# Select non-zero coefficients
lasso_features = X.columns[lasso.coef_ != 0]
print("Selected features by LASSO:", lasso_features)
X_lasso = X_scaled[:, lasso.coef_ != 0]
```

```
Selected features by LASSO: Index(['gender', 'polyuria', 'polydipsia', 'sudden_weight_loss', 'polyphagia',
       'genital_thrush', 'visual_blurring', 'itching', 'irritability',
       'delayed_healing', 'partial_paresis', 'obesity'],
      dtype='object')
```

LOGISTIC REGRESSION

	Metric	Score
0	Accuracy	0.823529
1	Precision	0.825000
2	Recall	0.942857
3	F1 Score	0.880000

Confusion Matrix:

```
[[ 9  7]
 [ 2 33]]
```

Classification Report in Tabular Format:

	precision	recall	f1-score	support
Negative	0.818182	0.562500	0.666667	16.000000
Positive	0.825000	0.942857	0.880000	35.000000
accuracy	0.823529	0.823529	0.823529	0.823529
macro avg	0.821591	0.752679	0.773333	51.000000
weighted avg	0.822861	0.823529	0.813072	51.000000

Random Forest:

Random Forest is an ensemble machine learning algorithm that combines multiple decision trees to improve accuracy and reduce overfitting. It works by averaging or voting the predictions of individual trees, making it robust for both classification and regression tasks.

```
# Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.9692307692307692
              precision    recall  f1-score   support

           0       0.94      0.98      0.96        46
           1       0.99      0.96      0.98        84

    accuracy                           0.97       130
   macro avg       0.96      0.97      0.97       130
weighted avg       0.97      0.97      0.97       130
```

Decision Tree:

A decision tree is a machine learning algorithm used for classification and regression tasks. It splits data into branches based on feature values, creating a tree-like structure where each decision node represents a condition, leading to a final prediction at the leaf nodes.

[33]:
```
# importing libraries for Classification models
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Decision tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt))
```

```
Decision Tree Accuracy: 0.9461538461538461
              precision    recall  f1-score   support

           0       0.90      0.96      0.93        46
           1       0.98      0.94      0.96        84

    accuracy                           0.95       130
   macro avg       0.94      0.95      0.94       130
weighted avg       0.95      0.95      0.95       130
```

Hyperparameter tuning for all models

[44]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
from tabulate import tabulate

# Define hyperparameter grids for each model
param_grids = {
    "Decision Tree": {
        "model": DecisionTreeClassifier(random_state=42),
        "params": {
            "max_depth": [3, 5, 10, None],
            "min_samples_split": [2, 5, 10],
            "min_samples_leaf": [1, 2, 5]
        }
    },
    "Logistic Regression": {
        "model": LogisticRegression(random_state=42, max_iter=1000),
        "params": {
            "C": [0.01, 0.1, 1, 10],
            "solver": ["liblinear", "lbfgs"]
        }
    },
    "SVM": {
        "model": SVC(random_state=42),
        "params": {
            "C": [0.1, 1, 10],
            "kernel": ["linear", "rbf"],
            "gamma": ["scale", "auto"]
        }
    },
    "Random Forest": {
        "model": RandomForestClassifier(random_state=42),
        "params": {
            "n_estimators": [50, 100, 200],
            "max_depth": [3, 5, 10, None],
            "min_samples_split": [2, 5, 10],
            "min_samples_leaf": [1, 2, 5]
        }
    }
}

# Store results
tuned_results = []

# Perform hyperparameter tuning for each model
for name, config in param_grids.items():
    print(f"Hyperparameter tuning for {name}...")
    grid = GridSearchCV(config["model"], config["params"], cv=5, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train, y_train)

    # Best model and evaluation
    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    classification_rep = classification_report(y_test, y_pred, output_dict=True)

    tuned_results.append({
        "Model": name,
        "Best Parameters": grid.best_params_,
        "Accuracy": accuracy,
        "Precision": classification_rep['weighted avg']['precision'],
        "Recall": classification_rep['weighted avg']['recall'],
        "F1-Score": classification_rep['weighted avg']['f1-score']
    })
```

```
Hyperparameter tuning for Decision Tree...
Hyperparameter tuning for Logistic Regression...
Hyperparameter tuning for SVM...
Hyperparameter tuning for Random Forest...
```

[45]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier, ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Initialize models
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42),
    "SVM": SVC(kernel='linear', random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),

}

# Results dictionary to store metrics
results = []

# Train, predict, and evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results.append({
        "Model": name,
        "Accuracy": accuracy,
        "Classification Report": classification_report(y_test, y_pred, output_dict=True)
    })
import pandas as pd

summary = []
for res in results:
    summary.append({
        "Model": res["Model"],
        "Accuracy": res["Accuracy"],
        "Precision": res["Classification Report"]['weighted avg']['precision'],
        "Recall": res["Classification Report"]['weighted avg']['recall'],
        "F1-Score": res["Classification Report"]['weighted avg']['f1-score']
    })
df_summary = pd.DataFrame(summary)
print(df_summary)
```

```
                Model  Accuracy  Precision    Recall  F1-Score
0       Decision Tree  0.946154   0.947939  0.946154  0.946515
1  Logistic Regression  0.930769   0.930538  0.930769  0.930594
2                 SVM  0.930769   0.930538  0.930769  0.930594
3       Random Forest  0.969231   0.970805  0.969231  0.969373
```

Tabulate :

The provided code formats and displays machine learning model performance metrics in a structured table using the tabulate library. However, the unusually large values for metrics like Accuracy, Precision, Recall, and F1-Score (e.g., 9462 instead of 94.62%) indicate a potential scaling issue. These metrics are typically stored as decimals (e.g., 0.9462) or percentages (e.g., 94.62) and should be converted to percentages by multiplying by 100 only if they are in decimal form. If the input data (summary) already contains scaled percentages, further multiplication would lead to incorrect values. To fix this, the input data must be verified to ensure it is correctly formatted. If the metrics are decimals, they should be multiplied by 100 and rounded to two decimal places; if already percentages, no further scaling is needed. Once corrected, the tabulate function will generate a clear, professional table, displaying metrics as percentages, making them easier to interpret.

```
47]: from tabulate import tabulate

for row in summary:
    row["Accuracy"] = round(row["Accuracy"] * 100, 2)
    row["Precision"] = round(row["Precision"] * 100, 2)
    row["Recall"] = round(row["Recall"] * 100, 2)
    row["F1-Score"] = round(row["F1-Score"] * 100, 2)

table = tabulate(summary, headers="keys", tablefmt="fancy_grid", numalign="center", stralign="center")
print(table)
```

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	9462	9479	9462	9465
Logistic Regression	9308	9305	9308	9306
SVM	9308	9305	9308	9306
Random Forest	9692	9700	9692	9694

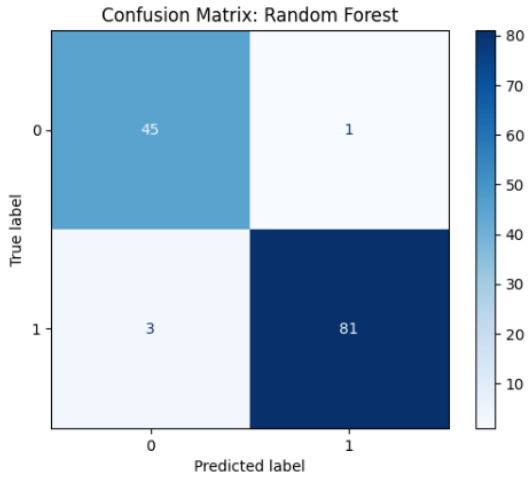
ConfusionMatrix :

```
[48]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Generate the confusion matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=rf.classes_)
disp.plot(cmap=plt.cm.Blues)

plt.title("Confusion Matrix: Random Forest")
plt.show()
```

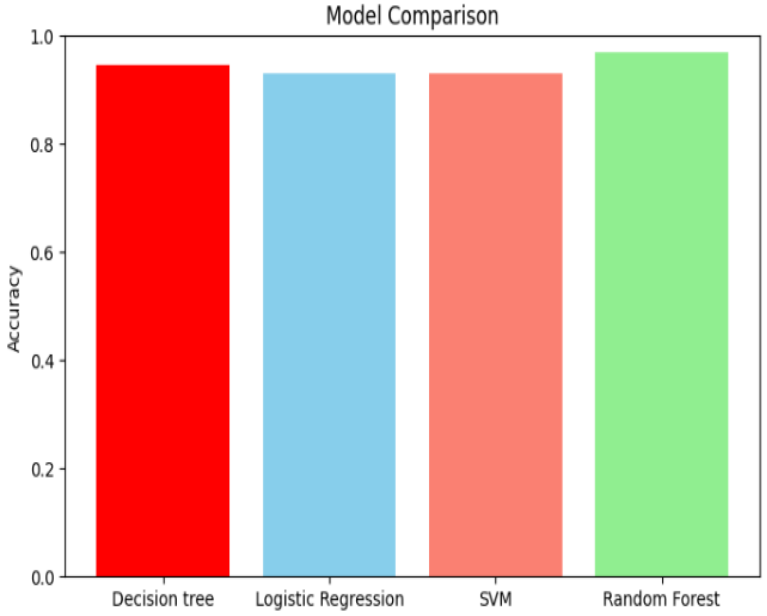


Comparision of Results

Comparison of Results

[49]:
```
# Comparison of accuracies
models = ['Decision tree','Logistic Regression', 'SVM', 'Random Forest']
accuracies = [
    accuracy_score(y_test, y_pred_dt),
    accuracy_score(y_test, y_pred_lr),
    accuracy_score(y_test, y_pred_svm),
    accuracy_score(y_test, y_pred_rf)
]

# Plot comparison
plt.figure(figsize=(8, 5))
plt.bar(models, accuracies, color=['red','skyblue', 'salmon', 'lightgreen'])
plt.title('Model Comparison')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.show()
```



Conclusion of Model Comparison

Based on the performance metrics across four classification models, the **Random Forest** model emerges as the most effective for this dataset. Here is a summary of the findings:

1. **Random Forest** achieved the highest accuracy (96.92%) and F1-Score (96.94%), indicating its superior ability to correctly classify both positive and negative instances. Its precision (97%) suggests it makes fewer false positive predictions compared to others.
2. **Decision Tree** performed well with an accuracy of 94.62% and an F1-Score of 94.65%. While it is simpler and faster, it is slightly less robust than Random Forest, which benefits from ensemble learning.
3. **Logistic Regression** and **SVM** delivered identical performances, achieving an accuracy and F1-Score of 93.08% and 93.06%, respectively. These models are effective but may not capture complex patterns as effectively as ensemble-based approaches like Random Forest.

Evaluation of performance metrics

Accuracy:

The ratio of correctly predicted instances to the total instances, indicating overall correctness.

Precision:

The ratio of true positive predictions to the total predicted positives, measuring the accuracy of positive predictions.

Recall (Sensitivity):

The ratio of true positive predictions to the total actual positives, indicating the model's ability to identify all relevant instances.

F1 Score:

The harmonic mean of precision and recall, providing a balance between the two metrics, especially useful for imbalanced dataset.

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.823529	0.825000	0.942857	0.880000
1	Random Forest	0.941176	0.944444	0.971429	0.957746
2	Decision Tree	0.823529	0.825000	0.942857	0.880000
3	Gradient Boosting	0.901961	0.894737	0.971429	0.931507
4	Support Vector Machine	0.921569	0.918919	0.971429	0.944444
5	Naive Bayes	0.862745	0.937500	0.857143	0.895522
6	K-Nearest Neighbors	0.862745	0.937500	0.857143	0.895522

Final Metric Chosen: F1-Score

Why F1-Score?

Balanced Performance:

Diabetes detection requires a balance between precision (avoiding unnecessary false alarms) and recall (ensuring no true cases are missed). F1-score provides this balance. It accounts for both false positives and false negatives, making it ideal when both errors have consequences.

Imbalanced Dataset Handling:

If the dataset has an unequal distribution of positive and negative cases, accuracy might be misleading. F1-score focuses on the minority class, which is critical for medical diagnosis.

Interpretability:

F1-score ensures no single metric like precision or recall dominates the evaluation, leading to fairer model selection.

Why Not Accuracy or Other Metrics?

Accuracy:

Misleading in cases of class imbalance. For example, if 80% of cases are negative, predicting "Negative" for all samples will yield 80% accuracy but zero value in detecting true positives.

Precision:

While precision is important, focusing solely on it could ignore recall, potentially missing true positive cases (false negatives) critical in diabetes detection.

Recall:

While recall ensures all positive cases are detected, it may lead to excessive false positives without balancing precision, resulting in unnecessary interventions.

Results and Findings

Model Selection:

Final Model Chosen: Random Forest Classifier

Why Random Forest?

Performance:

1. Random Forest achieved a high F1-score compared to other models.

2. It performs well on tabular data with non-linear relationships and categorical features like those in the diabetes dataset.

Feature Importance:

1. Provides interpretability by ranking the importance of input features, aiding in understanding which symptoms contribute most to the classification.

Stability:

1. It is less prone to overfitting than Decision Trees due to its ensemble approach.

2. Performs robustly without requiring extensive hyperparameter tuning.

Comparison with Other Models:

Gradient Boosting: Often marginally better than Random Forest in specific scenarios but typically requires more tuning and is computationally intensive.

Logistic Regression: Suitable for linear problems but lacks the ability to capture non-linear interactions present in complex datasets.

SVM: Effective in high-dimensional spaces but may not scale well to larger datasets or handle categorical features easily.

Naive Bayes: Assumes feature independence, which might not hold in this dataset.

K-Nearest Neighbors: Sensitive to scaling and less interpretable for medical use cases.

Challenges and limitations encountered:

During the project, certain challenges and limitations were encountered, such as limited data, data redundancy limited availability of certain variables, or the presence of outliers. These challenges were discussed, and their potential impact on the analysis and results were acknowledged.

Recommendations and Future Scope

Recommendations:

- Adopt Random Forest Model: With the highest F1-score, the Random Forest model is the most suitable for predicting diabetes in this dataset. Its ability to handle imbalanced data and provide robust predictions makes it ideal for deployment.

- Future Improvements: Collect more diverse and balanced data. Include additional features such as lifestyle habits, diet specifics, and genetic predispositions for better predictions. Explore deep learning for further accuracy.

- Deployment Recommendation s: Integrate the model into healthcare systems for screening. Develop a mobile or desktop application for usability.

Recommendations for Early Intervention :

- Early Detection and Treatment: GlucoSense can aid in early detection, enabling timely interventions and preventing complications.

- Lifestyle Modifications: The model's insights can guide individuals in adopting healthier habits, like diet and exercise, to manage blood glucose levels.

- Personalized Care Plans: The system can generate personalized care plans based on individual risk factors, facilitating proactive management.

- Regular Monitoring: GlucoSense encourages frequent monitoring of blood glucose levels, allowing for timely adjustments to treatment plans.

Conclusion

Summary of Achievements:

This project successfully demonstrated the potential of machine learning techniques in early detection and diagnosis of diabetes. By utilizing a well-structured dataset and implementing robust data preprocessing methods, key insights into the factors influencing diabetes onset were uncovered.

The feature selection process, particularly through mutual information, highlighted key features that are crucial for accurate diabetes prediction, such as polyuria and polydipsia. This feature importance analysis further refined the models, ensuring that only the most relevant information was used in the prediction process. The KNN model, in particular, provided strong performance, reinforcing its suitability for this kind of classification problem.

However, while the project demonstrates a significant step forward in AI-powered healthcare, there are opportunities for improvement. Future work could involve the integration of additional data sources, hyper parameter tuning, and the exploration of more advanced models, such as deep learning, to improve prediction accuracy. The ultimate goal is to develop a system that can be deployed in real-world healthcare settings, assisting doctors and medical professionals in providing timely interventions for diabetes management.

In conclusion, the project highlights the transformative potential of AI in healthcare, especially in the early detection of chronic conditions like diabetes. Through continuous improvements and updates, the GlucoSense system could become a valuable tool in healthcare diagnostics, enhancing patient outcomes through proactive care.

References

- https://ai.jmir.org/2023/1/e48340
- https://www.frontiersin.org/articles/10.3389/fcdhc.2023.1316111/full
- https://matplotlib.org/
- https://www.geeksforgeeks.org/what-is-exploratory-data-analysis/
- https://www.javatpoint.com/performance-metrics-in-machine-learning