# GlucoSense- AI-Powered Diabetes Detection for Early Intervention

A Project Report

Submitted to


Under the supervision of

Ravi

Submitted By

Charitha Duvvuru

21731A3119

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING -AI**

**PBR VISVODAYA INSTITUTE OF TECHNOLOGY & SCIENCE**

**(AUTONOMOUS)**

**(Affiliated to J.N.T.U.A, Approved by AICTE and Accredited by NAAC)**

**KAVALI, SPSR NELLORE, AP-524201.**

# Table of Contents

# Abstract

The prevalence of diabetes has surged globally over the past fifteen years, driven largely by changes in lifestyle and dietary habits. This alarming trend necessitates proactive measures for early detection and intervention. GlucoSense: AI-Powered Diabetes Detection for Early Intervention is an innovative project designed to address this need by harnessing advanced machine learning techniques. The primary objective is to develop a predictive model capable of categorizing individuals as diabetic, pre-diabetic, or healthy based on healthcare statistics and lifestyle data. Early detection empowers healthcare providers and individuals to adopt timely interventions, potentially reversing or mitigating the impact of diabetes.

The project employs a systematic, milestone-based approach. It begins with comprehensive data collection, utilizing healthcare records and lifestyle survey data to build a robust dataset. This is followed by data exploration and preprocessing, which involves handling missing values, addressing class imbalances, encoding categorical data, and removing outliers. Advanced preprocessing techniques such as oversampling are employed to enhance model accuracy. Feature selection and dimensionality reduction techniques are then used to identify the most relevant attributes influencing diabetes prediction, optimizing the dataset for machine learning models.

The project's core involves building classification models using diverse algorithms, including ensemble techniques, to ensure a reliable and accurate predictive system. Each model is rigorously evaluated using metrics like precision, recall, F1 score, AUC, and overall classification accuracy. Comparative analysis of these metrics enables the selection of the most effective model.

Upon finalizing the model, a comprehensive evaluation and validation process is conducted to ensure its reliability across varying data thresholds. The culmination of the project includes a detailed presentation and documentation capturing every stage, from problem statement and data analysis to model development, results, and recommendations for implementation. The deliverables also include final code submissions on GitHub for reproducibility and scalability.

By integrating healthcare data with machine learning, GlucoSense aims to transform diabetes management, offering a predictive tool that supports both individuals and healthcare providers. This initiative not only underscores the critical role of technology in health analytics but also sets a benchmark for future projects targeting lifestyle-induced health issues. Ultimately, the project aspires to foster awareness and promote preventive healthcare measures in the fight against diabetes.

# Introduction

Diabetes is one of the most significant global health challenges of the 21st century, affecting millions of individuals across all age groups. Over the past fifteen years, the prevalence of diabetes has escalated at an alarming rate due to rapid urbanization, sedentary lifestyles, and unhealthy dietary patterns. As a metabolic disorder, diabetes disrupts the body's ability to regulate blood glucose levels effectively, leading to long-term complications such as cardiovascular diseases, kidney failure, neuropathy, and vision impairment. Despite advancements in medical science, the burden of diabetes continues to rise, highlighting the urgent need for innovative solutions that facilitate early detection and intervention.

Traditionally, diabetes is diagnosed through clinical tests such as fasting blood sugar levels, HbA1c tests, and oral glucose tolerance tests. While effective, these approaches are often reactive, identifying the disease only after it has progressed significantly. In contrast, predictive analytics powered by artificial intelligence (AI) offers a proactive solution. By analysing large-scale datasets that encompass lifestyle choices, dietary habits, and healthcare statistics, AI systems can identify individuals at risk of developing diabetes long before clinical symptoms manifest. This project, GlucoSense: AI-Powered Diabetes Detection for Early Intervention, embodies this forward-thinking approach, leveraging the power of machine learning to develop a robust diabetes prediction system.

The primary objective of this project is to analyse the intricate relationships between lifestyle factors and the likelihood of diabetes onset. The project follows a structured methodology beginning with data collection from healthcare records and lifestyle surveys. This data is then subjected to exploratory analysis and preprocessing to ensure accuracy, consistency, and relevance. Feature selection and dimensionality reduction techniques are employed to identify the most influential variables, optimizing the dataset for machine learning algorithms. The system will then implement various classification models, including ensemble techniques, to predict whether an individual is diabetic, pre-diabetic, or healthy.

A key aspect of the project is the rigorous evaluation of model performance. Metrics such as precision, recall, F1 score, and the area under the curve (AUC) will be used to compare the efficacy of different models. The final output will include a scalable and interpretable AI model, along with actionable insights and recommendations. This deliverable will be accompanied by detailed documentation and a presentation to effectively communicate the findings and their implications.

By focusing on prevention rather than cure, GlucoSense seeks to transform the landscape of diabetes management. It aims to empower individuals and healthcare providers with the tools needed to make informed decisions, ultimately reducing the global burden of diabetes. This project also serves as a testament to the transformative potential of AI in addressing complex health challenges, paving the way for future innovations in predictive healthcare analytics.

# Problem Statement

Diabetes has become a critical global health challenge, affecting millions of lives and imposing substantial burdens on healthcare systems. Over the past fifteen years, its prevalence has risen sharply due to unhealthy lifestyle habits, poor dietary choices, and sedentary behaviour. While traditional diagnostic methods, such as blood glucose tests, are reliable, they often identify the disease only after significant progression. This delay in detection not only increases the risk of severe complications, such as cardiovascular diseases, kidney failure, and neuropathy, but also limits the opportunity for early intervention and prevention.

Currently, there is a lack of predictive tools that leverage the vast troves of healthcare and lifestyle data to anticipate the risk of diabetes before symptoms manifest. The absence of such systems results in missed opportunities for timely lifestyle adjustments and medical interventions that could potentially reverse or delay the onset of the disease. With diabetes-related healthcare costs and mortality rates on the rise, the need for an innovative, data-driven approach is more urgent than ever.

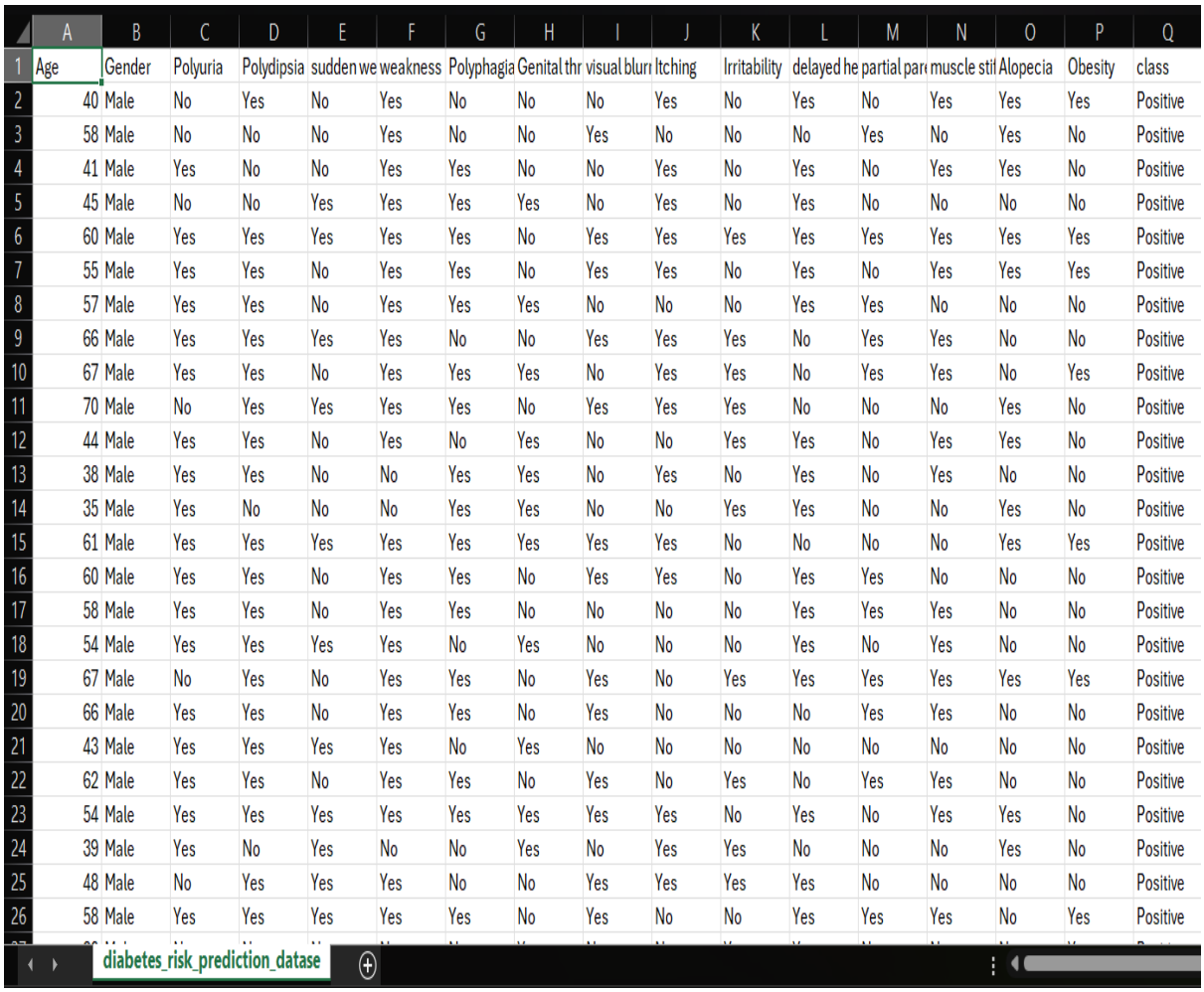
This project addresses the critical gap by developing GlucoSense: AI-Powered Diabetes Detection for Early Intervention. By integrating healthcare statistics, lifestyle factors, and advanced machine learning models, the project aims to provide an accurate, scalable, and proactive solution for early diabetes prediction. This system will empower individuals and healthcare providers to take preventive measures, reducing the global burden of diabetes and improving public health outcomes.

# Data Collection

**Data Sources:**

For the GlucoSense project, the data used for analysis and model development was sourced from Kaggle, utilizing the **"diabetes_risk_prediction_dataset.csv"**, which contains comprehensive healthcare and lifestyle statistics essential for predicting diabetes risk.

**Sample of 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

- **Age:**
  The age of the individual, which is a key factor in determining the risk of developing diabetes, with older individuals generally being at higher risk.
- **Gender:**
  Indicates the sex of the individual (Male/Female).
- **Polyuria:**
  Frequent urination, often a symptom of uncontrolled diabetes, as the kidneys attempt to excrete excess glucose.
- **Polydipsia:**
  Excessive thirst, typically occurring due to dehydration from frequent urination in individuals with high blood sugar levels.
- **Sudden Weight Loss:**
  Unexplained weight loss, which can occur when the body breaks down muscle.
- **Weakness:**
  A general feeling of fatigue or lack of energy, often seen in individuals with uncontrolled blood sugar levels due to the body's inability to properly use glucose.
- **Polyphagia:**
  Excessive hunger, a result of the body's inability to use glucose properly.
- **Genital Thrush:**
  A fungal infection in the genital area.
- **Visual Blurring:**
  Blurred vision caused by fluctuating blood sugar levels, which can affect the shape of the eye's lens and lead to temporary or permanent vision problems.
- **Itching:**
  Skin irritation or dryness, which occurs in individuals with diabetes.
- **Irritability:**
  Mood swings or irritability, which can be a result of fluctuating blood sugar levels affecting brain function**.**
- **Delayed Healing:**
  Slower recovery from cuts or infections, common in individuals with diabetes due to poor circulation and impaired immune response.
- **Partial Paresis:**
  Partial paralysis or muscle weakness resulting from nerve damage.
- **Muscle Stiffness:**
  Tightness or rigidity in muscles, often caused by diabetic neuropathy or poor circulation associated with high blood sugar levels.
- **Alopecia:**
  Hair loss, which can result from diabetes-related complications like poor circulation and nerve damage affecting hair follicles.
- **Obesity:**
  Excess body fat, particularly abdominal fat.
- **Class:**
  The target variable that indicates whether the individual is diabetic or not, based on the combination of all the features in the dataset.

# Data Exploration (EDA) and Data Preprocessing

## 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 libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from itertools import combinations
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

## Load the dataset:

```
#load the dataset
data = pd.read_csv("diabetes_risk_prediction_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.

```
#number of duplicate rows
print("Number of duplicate rows:")
print(data.duplicated().sum())
```

```
Number of duplicate rows:
269
```

```
#drop the duplicate rows
data=data.drop_duplicates()
```

After dropping the duplicates, the statistics is as follows:

```
print("Summary Statistics:")
print(data.describe())
```

```
Summary Statistics:
              Age
count  251.000000
mean    48.864542
std     12.526036
min     16.000000
25%     39.000000
50%     48.000000
75%     58.000000
max     90.000000
```

```
#check for missing values
print("Missing Values in the Dataset:")
print(data.isnull().sum())
```

```
Missing Values in the Dataset:
Age                 0
Gender              0
Polyuria            0
Polydipsia          0
sudden weight loss  0
weakness            0
Polyphagia          0
Genital thrush      0
visual blurring     0
Itching             0
Irritability        0
delayed healing     0
partial paresis     0
muscle stiffness    0
Alopecia            0
Obesity             0
class               0
```

**Outlier detection:**

Outlier detection using the Interquartile Range (IQR) method is a statistical technique used to identify data points that significantly differ from the rest of the data. The IQR is calculated by measuring the spread of the middle 50% of the data. Outliers are identified as any data points that fall below or above a certain range based on the IQR.

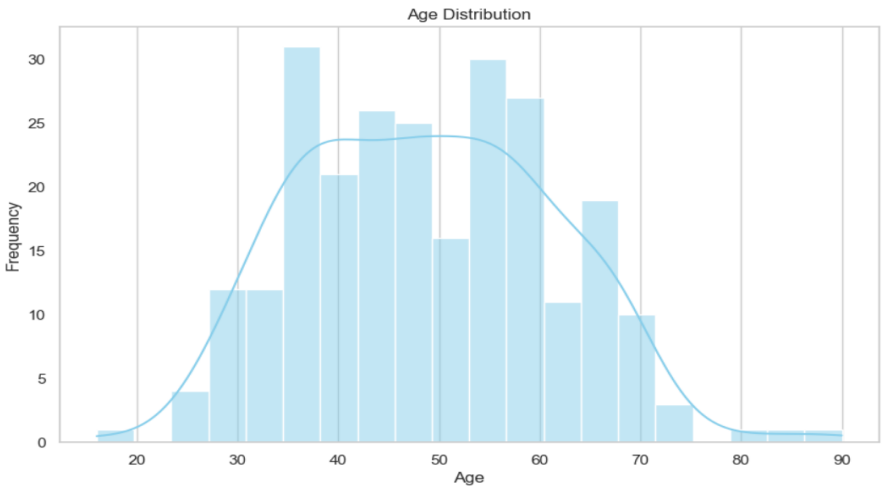
```python
# Function to find outliers in a dataset using the IQR method
def find_outliers_iqr(data):
    outliers_dict = {}
    for column in data.select_dtypes(include=[np.number]).columns:
        Q1 = data[column].quantile(0.25)  # 1st quartile (25th percentile)
        Q3 = data[column].quantile(0.75)  # 3rd quartile (75th percentile)
        IQR = Q3 - Q1  # Interquartile Range
        # Define the lower and upper bounds for outliers
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        # Identify outliers in the current column
        outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)]
        # Store outliers in the dictionary
        outliers_dict[column] = outliers
        # Print outliers for the current column
        print(f"\nOutliers in '{column}':")
        print(outliers)
    return outliers_dict  # Return the dictionary containing outliers for each column
# Call the function to find outliers in the dataset 'data'
outliers = find_outliers_iqr(data)
```

```
Outliers in 'Age':
     Age  Gender Polyuria Polydipsia sudden weight loss weakness Polyphagia  \
102   90  Female       No        Yes                Yes       No         No

    Genital thrush visual blurring Itching Irritability delayed healing  \
102            Yes             Yes     Yes           No              No

    partial paresis muscle stiffness Alopecia Obesity     class
102              No              Yes      Yes      No  Positive
```

## Univariate analysis:

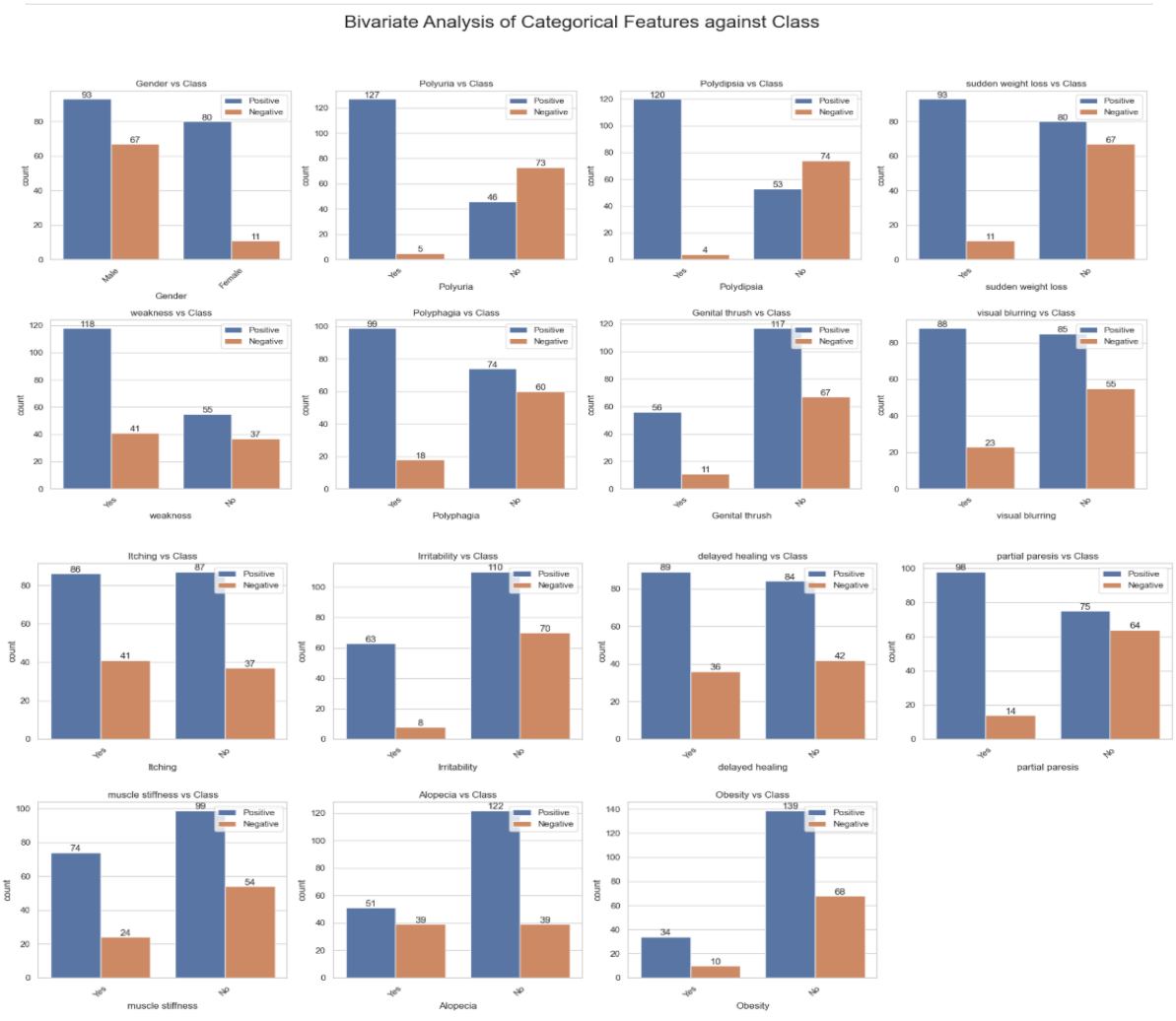
Univariate analysis involves examining a single variable to understand its distribution, central tendency (mean, median, mode), and spread (variance, standard deviation). It uses visualizations like histograms and box plots to identify patterns, outliers, and data quality issues, helping inform decisions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

```python
#histogram plot to show age distribution frequency
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.histplot(data['Age'], bins=20, kde=True, color='skyblue')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(axis='y')
plt.show()
```



## Bivariate analysis:

```
#Bivariate analysis of every feature with the target variable
important_features = ['Gender', 'Polyuria', 'Polydipsia', 'sudden weight loss', 'weakness',
                      'Polyphagia', 'Genital thrush', 'visual blurring', 'Itching', 'Irritability',
                      'delayed healing', 'partial paresis', 'muscle stiffness', 'Alopecia', 'Obesity']
#placing 4 plots in each row
n_cols = 4
n_rows = -(-len(important_features) // n_cols)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 20))
fig.suptitle('Bivariate Analysis of Categorical Features against Class', fontsize=24)
axes = axes.flatten()
for i, feature in enumerate(important_features):
    ax = axes[i]
    if i == 0:
        plot = sns.countplot(data=data, x=feature, hue='class', ax=ax)
    else:
        plot = sns.countplot(data=data, x=feature, hue='class', ax=ax, order=["Yes", "No"])
    ax.set_title(f'{feature} vs Class')
    ax.tick_params(axis='x', rotation=45)
    for p in plot.patches:
        height = p.get_height()
        if height > 0:
            plot.annotate(f'{int(height)}',
                          (p.get_x() + p.get_width() / 2., height),
                          ha='center', va='center',
                          xytext=(0, 5), textcoords='offset points')
    plot.legend(loc='upper right')
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



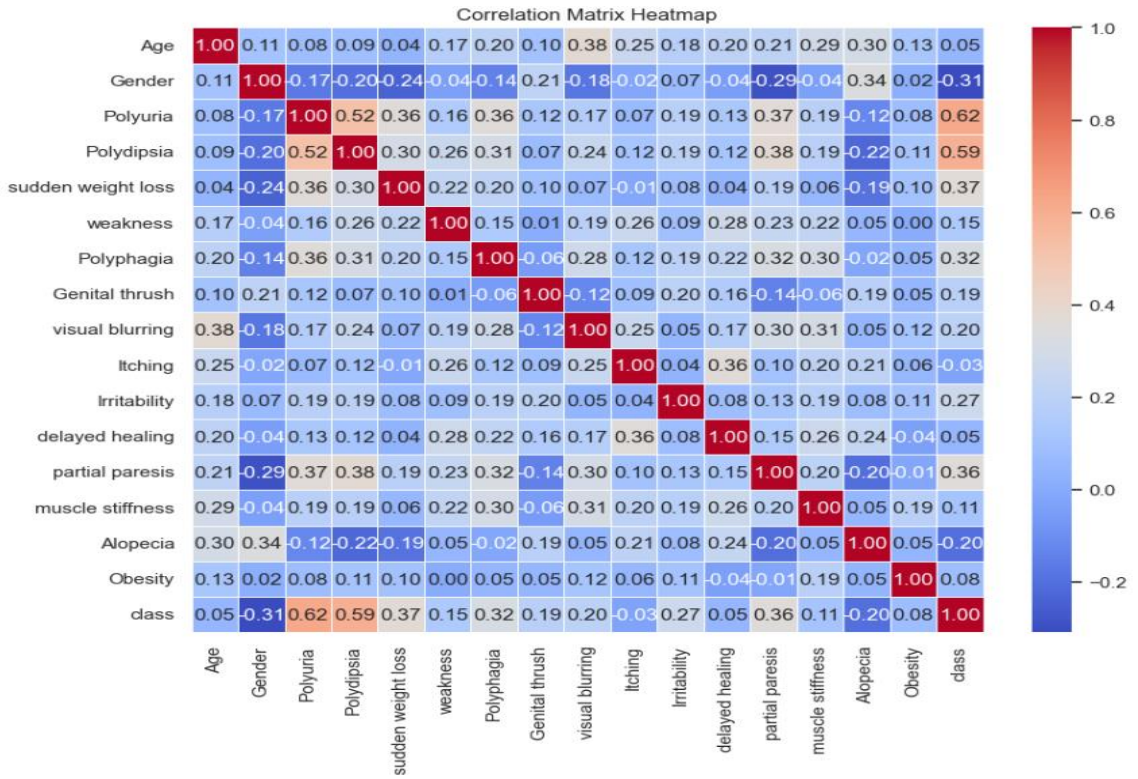
**Key Insights:**

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.

**Corelation 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.

```
# Create a copy of the original dataframe to avoid modifying it
df_encoded = data.copy()
# Dictionary to store label encoders for categorical columns
label_encoders = {}
# Loop through columns and encode categorical features
for column in df_encoded.columns:
    if df_encoded[column].dtype == 'object':  # Check for categorical columns
        label_encoders[column] = LabelEncoder()  # Initialize encoder
        df_encoded[column] = label_encoders[column].fit_transform(df_encoded[column])  # Apply encoding
# Compute the correlation matrix for the encoded dataframe
correlation_matrix = df_encoded.corr()
# Plot the heatmap for the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix Heatmap')  # Set the title
plt.show()  # Display the heatmap
```



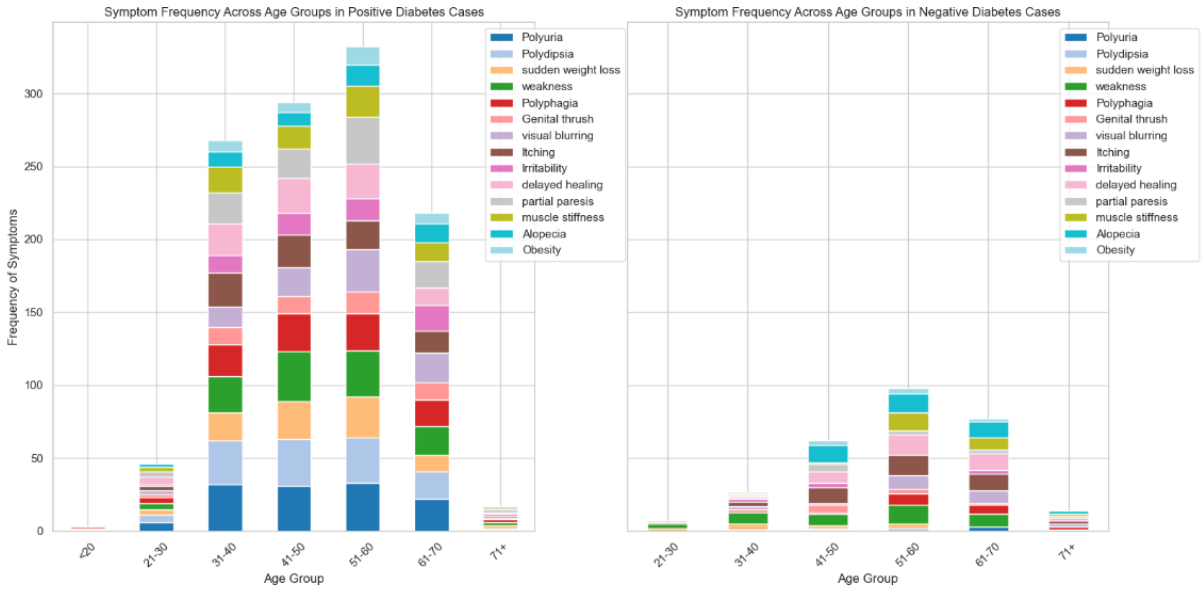
**High Positive Correlations:**

Polyuria and Polydipsia have a strong positive correlation, suggesting these symptoms often co-occur. Polyuria and class: Polyuria shows a strong positive correlation with diabetes presence. Polydipsia and class: Similarly, Polydipsia is also strongly correlated with the class, indicating that it may be an important feature for diabetes prediction.

**Top Correlated Features with the Target (class):**

Polyuria: 0.62 (strongest predictor of diabetes presence). Polydipsia: 0.59 (second strongest predictor). Sudden Weight Loss: 0.37 (moderate correlation with diabetes presence).

```
binary_features = ['Polyuria', 'Polydipsia', 'sudden weight loss', 'weakness',
                   'Polyphagia', 'Genital thrush', 'visual blurring', 'Itching',
                   'Irritability', 'delayed healing', 'partial paresis', 'muscle stiffness',
                   'Alopecia', 'Obesity']
positive_diabetes_copy = data[data['class'] == 'Positive'].copy()
negative_diabetes_copy = data[data['class'] == 'Negative'].copy()
age_bins = [0, 20, 30, 40, 50, 60, 70, 80]
age_labels = ['<20', '21-30', '31-40', '41-50', '51-60', '61-70', '71+']
# Create a new 'Age Group' column based on age ranges for both positive and negative cases
positive_diabetes_copy['Age Group'] = pd.cut(positive_diabetes_copy['Age'], bins=age_bins, labels=age_labels)
negative_diabetes_copy['Age Group'] = pd.cut(negative_diabetes_copy['Age'], bins=age_bins, labels=age_labels)
# Group by 'Age Group' and count the frequency of 'Yes' values for each symptom in positive cases
age_symptom_counts_positive = positive_diabetes_copy.groupby('Age Group', observed=True)[binary_features].apply(lambda x: (
# Group by 'Age Group' and count the frequency of 'Yes' values for each symptom in negative cases
age_symptom_counts_negative = negative_diabetes_copy.groupby('Age Group', observed=True)[binary_features].apply(lambda x: (
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8), sharey=True)
age_symptom_counts_positive.plot(kind='bar', stacked=True, ax=ax1, colormap='tab20')
ax1.set_title('Symptom Frequency Across Age Groups in Positive Diabetes Cases')
ax1.set_xlabel('Age Group')
ax1.set_ylabel('Frequency of Symptoms')
ax1.legend(loc='upper right', bbox_to_anchor=(1.2, 1))
ax1.tick_params(axis='x', rotation=45)
# Plot stacked bar chart for negative diabetes cases
age_symptom_counts_negative.plot(kind='bar', stacked=True, ax=ax2, colormap='tab20')
ax2.set_title('Symptom Frequency Across Age Groups in Negative Diabetes Cases')
ax2.set_xlabel('Age Group')
ax2.legend(loc='upper right', bbox_to_anchor=(1.2, 1))  # Move legend outside the plot
ax2.tick_params(axis='x', rotation=45)  # Rotate x-axis labels for better readability
# Adjust layout to prevent overlapping and ensure plots are well spaced
plt.tight_layout()
plt.show()
```



**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 incidence.

# 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 40.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| **1** | 58.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| **2** | 41.0 | 1.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| **3** | 45.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **4** | 60.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

### Scaling the data:

Min-Max Scaling (or normalization) is a data preprocessing technique that transforms data to a fixed range, typically [0, 1]. It ensures that all feature values are scaled proportionally without distorting their relationships. This is particularly useful for machine learning algorithms sensitive to feature magnitude.

```
#scaling the data
scaler = MinMaxScaler()
data['Age'] = scaler.fit_transform(data[['Age']])
```

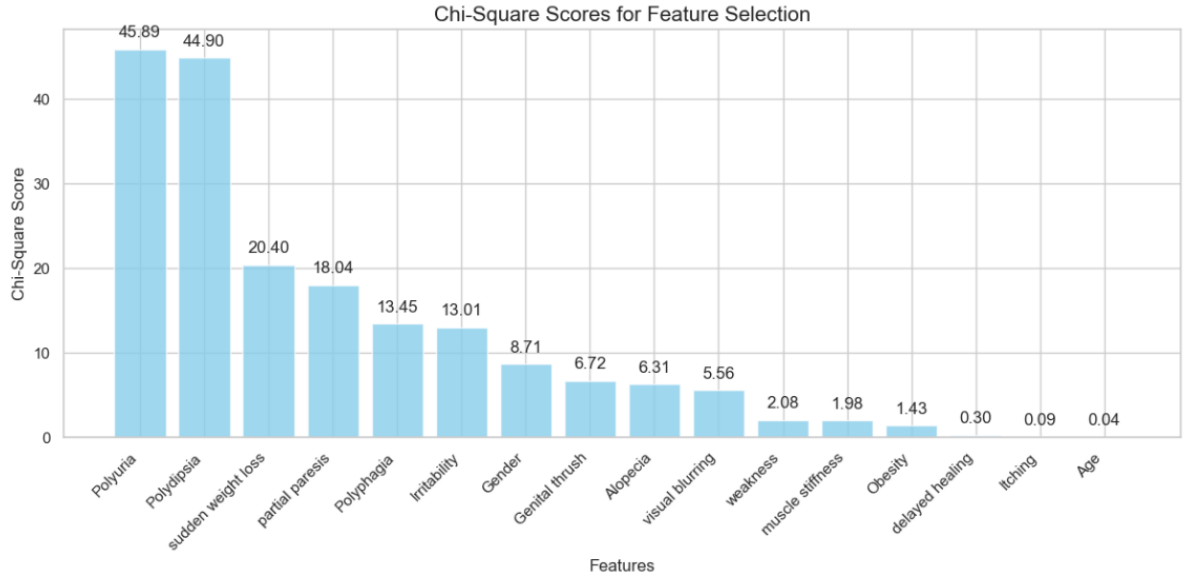
### Feature selection using chi square test:

Using the Chi-Square test for feature selection is a powerful approach, especially for datasets with categorical variables. It helps identify which features are significantly associated with the target variable, allowing for the construction of more effective predictive models by focusing on the most relevant features.

```
target_column = 'class'
X = data.drop(columns=[target_column])
y = data[target_column]
categorical_cols = X.select_dtypes(include=['object', 'category']).columns
for col in categorical_cols:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
X = X.astype(float)
# Initialize the SelectKBest method with the chi-square test as the score function
chi2_selector = SelectKBest(score_func=chi2, k='all')
chi2_selector.fit(X, y)
chi2_scores = pd.DataFrame({
    'Feature': X.columns,
    'Chi2 Score': chi2_selector.scores_
}).sort_values(by='Chi2 Score', ascending=False)
# Print the feature selection results using the chi-square test
print("Feature Selection using Chi-Square Test:")
print(chi2_scores)
```

```
Feature Selection using Chi-Square Test:
               Feature  Chi2 Score
2             Polyuria   45.890026
3           Polydipsia   44.903006
4   sudden weight loss   20.403086
12     partial paresis   18.043260
6           Polyphagia   13.449259
10        Irritability   13.006198
1               Gender    8.712006
7       Genital thrush    6.720759
14            Alopecia    6.313391
8       visual blurring    5.556841
5             weakness    2.077010
13    muscle stiffness    1.984555
15            Obesity    1.431754
11     delayed healing    0.302236
9             Itching    0.086492
0                 Age    0.044684
```

```
plt.figure(figsize=(12, 6))
# Create a bar chart to visualize the Chi-Square scores for each feature
bars = plt.bar(chi2_scores['Feature'], chi2_scores['Chi2 Score'], color='skyblue', alpha=0.8)
for bar, score in zip(bars, chi2_scores['Chi2 Score']):
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 1,
        f'{score:.2f}',
        ha='center', va='bottom', fontsize=12
    )
plt.xticks(rotation=45, ha='right')
plt.xlabel('Features')
plt.ylabel('Chi-Square Score')
plt.title('Chi-Square Scores for Feature Selection', fontsize=15)
plt.tight_layout()
plt.show()
```



**Top Features:**

Polyuria and Polydipsia have the highest Chi-square scores (~45.89 and ~44.90). These are the most strongly associated with the target variable, likely critical for predictions. Sudden weight loss and Partial paresis also show strong relevance (scores ~20.40 and ~18.04).

**Moderate Features:**

Polyphagia, Irritability, and Genital thrush have moderate scores (~8–13). They may be relevant but less impactful compared to the top features.

**Low-Impact Features:**

Weakness, Muscle stiffness, and Obesity have low scores (<2.5), suggesting weaker associations with the target. Delayed healing, Itching, and Age have negligible scores, likely not useful for prediction.

**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.

```
# Scale the features (X) using the specified scaler (e.g., StandardScaler)
X_scaled = scaler.fit_transform(X)
# Initialize the PCA (Principal Component Analysis) model
pca = PCA()
pca.fit(X_scaled)
cumulative_variance = pca.explained_variance_ratio_.cumsum()
# Find the number of components that explain at least 95% of the variance
n_components = (cumulative_variance < 0.95).sum() + 1
print(f"Number of components to retain 95% variance: {n_components}")
```

```
Number of components to retain 95% variance: 14
```

**Actionable Steps:**

But in this case, we only have 17 features in which one is target feature and we remain with 16 features after performing PCA we get to know that 14 features are essential to retain 95% data. So, there is no need to perform dimensionality reduction in this case.

**Divide target columns from data:**

```
X=data.drop(columns=['class'])
y=data['class']
```

**Train test split:**

```
#divide train data and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print("Shape of X_train:",X_train.shape)
print("Shape of X_test:",X_test.shape)
print("Shape of y_train:",y_train.shape)
print("Shape of y_test:",y_test.shape)
```

```
Shape of X_train: (200, 16)
Shape of X_test: (51, 16)
Shape of y_train: (200,)
Shape of y_test: (51,)
```

# Build a classification model

### 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.

```
results=[]
```

```
# Define the hyperparameter grid for Logistic Regression
param_grid_lr = {
    'C': [0.1, 1, 10],
    'solver': ['liblinear', 'lbfgs']
}
# Initialize the Logistic Regression model with max iterations set to 500 and random state for reproducibility
lr = LogisticRegression(max_iter=500, random_state=42)
# Perform GridSearchCV to search for the best hyperparameters using 5-fold cross-validation and 'f1' scoring
grid_search_lr = GridSearchCV(lr, param_grid_lr, cv=5, scoring='f1', n_jobs=-1)
# Fit the GridSearchCV model to the training data
grid_search_lr.fit(X_train, y_train)
# Get the best model (Logistic Regression) from GridSearchCV
best_lr = grid_search_lr.best_estimator_
y_pred_lr = best_lr.predict(X_test)
y_prob_lr = best_lr.predict_proba(X_test)[:, 1]
print(f"Best Parameters for Logistic Regression: {grid_search_lr.best_params_}")
```

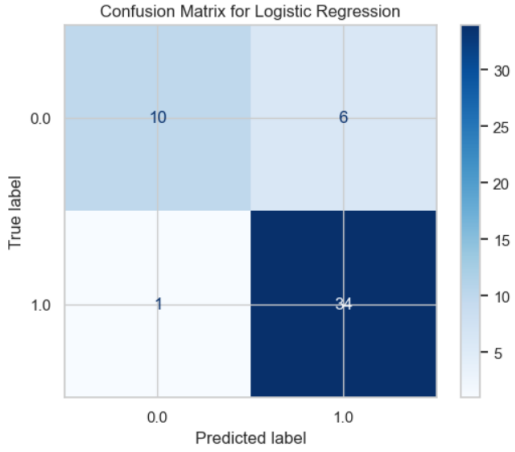
```
Best Parameters for Logistic Regression: {'C': 0.1, 'solver': 'lbfgs'}
```

```
# Calculate and display various performance metrics for the Logistic Regression model
accuracy = accuracy_score(y_test, y_pred_lr)
precision = precision_score(y_test, y_pred_lr)
recall = recall_score(y_test, y_pred_lr)
f1 = f1_score(y_test, y_pred_lr)
auc = roc_auc_score(y_test, y_prob_lr)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC: {auc:.4f}")
```

```
# Calculate and display various performance metrics for the Logistic Regression model
accuracy = accuracy_score(y_test, y_pred_lr)
precision = precision_score(y_test, y_pred_lr)
recall = recall_score(y_test, y_pred_lr)
f1 = f1_score(y_test, y_pred_lr)
auc = roc_auc_score(y_test, y_prob_lr)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC: {auc:.4f}")
```

```
Accuracy: 0.8627
Precision: 0.8500
Recall: 0.9714
F1 Score: 0.9067
AUC: 0.9500
```

```
cm = confusion_matrix(y_test, y_pred_lr)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Logistic Regression")
plt.show()
```



**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.

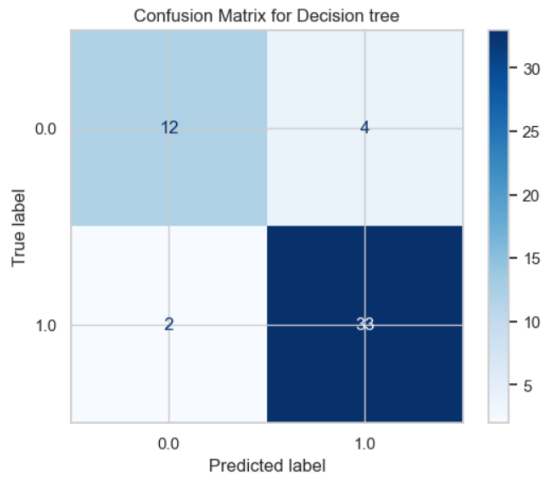
```
param_grid_dt = {
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}
# Initialize the Decision Tree classifier with a fixed random state for reproducibility
dt = DecisionTreeClassifier(random_state=42)
# Perform GridSearchCV to search for the best hyperparameters using 5-fold cross-validation and 'f1' scoring
grid_search_dt = GridSearchCV(dt, param_grid_dt, cv=5, scoring='f1', n_jobs=-1)
grid_search_dt.fit(X_train, y_train)
# Retrieve the best Decision Tree model found by GridSearchCV
best_dt = grid_search_dt.best_estimator_
# Make predictions on the test data using the best Decision Tree model
y_pred_dt = best_dt.predict(X_test)
print(f"Best Parameters for Decision Tree: {grid_search_dt.best_params_}")
```

```
Best Parameters for Decision Tree: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 2}
```

```
# Calculate and display various performance metrics for the decision tree model
accuracy = accuracy_score(y_test, y_pred_dt)
precision = precision_score(y_test, y_pred_dt)
recall = recall_score(y_test, y_pred_dt)
f1 = f1_score(y_test, y_pred_dt)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.8824
Precision: 0.8919
Recall: 0.9429
F1 Score: 0.9167
```

```
cm = confusion_matrix(y_test, y_pred_dt)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Decision tree")
plt.show()
```




**Support Vector Machine:**

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that best separates data points into distinct classes while maximizing the margin between them.

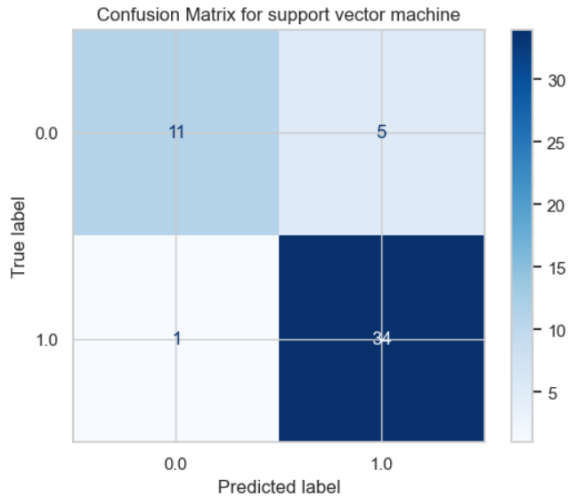
```
# Define the hyperparameter grid for Support Vector Classifier (SVC)
param_grid_svc = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
svc = SVC(random_state=42, probability=True)
# Perform GridSearchCV to find the best hyperparameters using 5-fold cross-validation and 'f1' scoring
grid_search_svc = GridSearchCV(svc, param_grid_svc, cv=5, scoring='f1', n_jobs=-1)
grid_search_svc.fit(X_train, y_train)
best_svc = grid_search_svc.best_estimator_
y_pred_svc = best_svc.predict(X_test)
y_prob_svc = best_svc.predict_proba(X_test)[:, 1]
print(f"Best Parameters for SVM: {grid_search_svc.best_params_}")
```

```
Best Parameters for SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

```
# Calculate and display various performance metrics for the support vector machine model
accuracy = accuracy_score(y_test, y_pred_svc)
precision = precision_score(y_test, y_pred_svc)
recall = recall_score(y_test, y_pred_svc)
f1 = f1_score(y_test, y_pred_svc)
auc = roc_auc_score(y_test, y_prob_svc)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC: {auc:.4f}")
```

```
Accuracy: 0.8824
Precision: 0.8718
Recall: 0.9714
F1 Score: 0.9189
AUC: 0.9679
```

```
cm = confusion_matrix(y_test, y_pred_svc)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for support vector machine")
plt.show()
```




**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.

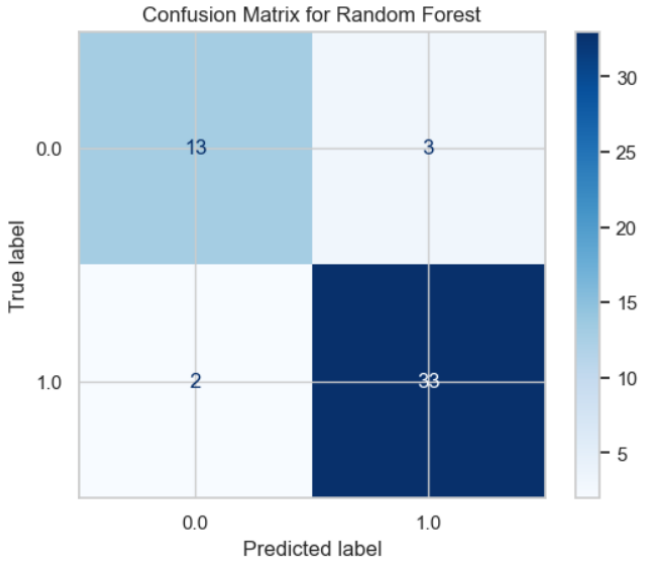
```
# Define the hyperparameter grid for Random Forest Classifier
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 20],
    'min_samples_split': [2, 5, 10]
}
# Initialize the Random Forest Classifier with a fixed random state for reproducibility
rf = RandomForestClassifier(random_state=42)
# Perform GridSearchCV to search for the best hyperparameters using 5-fold cross-validation and 'f1' scoring
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='f1', n_jobs=-1)
grid_search_rf.fit(X_train, y_train)
best_rf = grid_search_rf.best_estimator_
y_pred_rf = best_rf.predict(X_test)
y_prob_rf = best_rf.predict_proba(X_test)[:, 1]
print(f"Best Parameters for Random Forest: {grid_search_rf.best_params_}")
```

```
Best Parameters for Random Forest: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 50}
```

```
# Calculate and display various performance metrics for the random forest model
accuracy = accuracy_score(y_test, y_pred_rf)
precision = precision_score(y_test, y_pred_rf)
recall = recall_score(y_test, y_pred_rf)
f1 = f1_score(y_test, y_pred_rf)
auc = roc_auc_score(y_test, y_prob_rf)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC: {auc:.4f}")
```

```
Accuracy: 0.9020
Precision: 0.9167
Recall: 0.9429
F1 Score: 0.9296
AUC: 0.9777
```

```
cm = confusion_matrix(y_test, y_pred_rf)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Random Forest")
plt.show()
```




## Gradient Boosting:

Gradient Boosting is an ensemble machine learning technique that builds models sequentially, with each new model correcting the errors of the previous ones. It combines weak learners, typically decision trees, into a strong model by optimizing a loss function to improve accuracy and reduce bias.

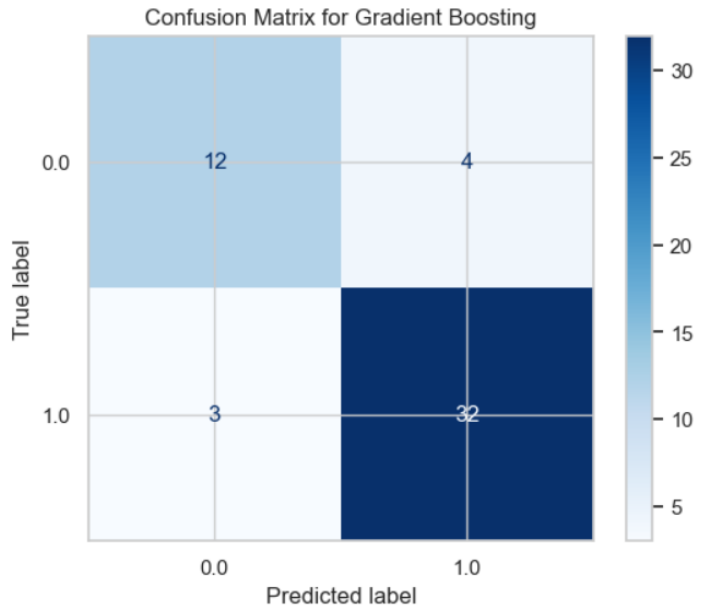
```
# Define the hyperparameter grid for Gradient Boosting Classifier
param_grid_gb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10]
}
# Initialize the Gradient Boosting Classifier with a fixed random state for reproducibility
gb = GradientBoostingClassifier(random_state=42)
# Perform GridSearchCV to find the best hyperparameters using 5-fold cross-validation and 'f1' scoring
grid_search_gb = GridSearchCV(gb, param_grid_gb, cv=5, scoring='f1', n_jobs=-1)
grid_search_gb.fit(X_train, y_train)
best_gb = grid_search_gb.best_estimator_
y_pred_gb = best_gb.predict(X_test)
y_prob_gb = best_gb.predict_proba(X_test)[:, 1]
print(f"Best Parameters for Gradient Boosting: {grid_search_gb.best_params_}")
```

```
Best Parameters for Gradient Boosting: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200}
```

```
# Calculate and display various performance metrics for the gradient boosting model
accuracy = accuracy_score(y_test, y_pred_gb)
precision = precision_score(y_test, y_pred_gb)
recall = recall_score(y_test, y_pred_gb)
f1 = f1_score(y_test, y_pred_gb)
auc = roc_auc_score(y_test, y_prob_gb)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC: {auc:.4f}")
```

```
Accuracy: 0.8627
Precision: 0.8889
Recall: 0.9143
F1 Score: 0.9014
AUC: 0.9500
```

```
cm = confusion_matrix(y_test, y_pred_gb)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Gradient Boosting")
plt.show()
```




**Extra trees:**

Extra Trees (Extremely Randomized Trees) is an ensemble learning method similar to Random Forest but differs by using random splits for feature thresholds. This increases diversity among trees, reduces overfitting, and improves computational efficiency for both classification and regression tasks.

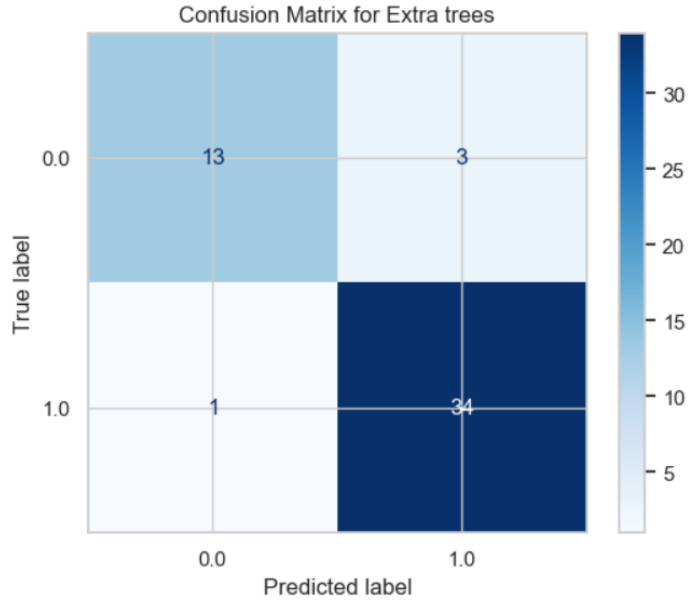
```
param_grid_et = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 20],
    'min_samples_split': [2, 5, 10]
}
et = ExtraTreesClassifier(random_state=42)
grid_search_et = GridSearchCV(et, param_grid_et, cv=5, scoring='f1', n_jobs=-1)
grid_search_et.fit(X_train, y_train)
best_et = grid_search_et.best_estimator_
y_pred_et = best_et.predict(X_test)
y_prob_et = best_et.predict_proba(X_test)[:, 1]
print(f"Best Parameters for Extra Trees: {grid_search_et.best_params_}")
```

```
Best Parameters for Extra Trees: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}
```

```
# Calculate and display various performance metrics for the extra trees model
accuracy = accuracy_score(y_test, y_pred_et)
precision = precision_score(y_test, y_pred_et)
recall = recall_score(y_test, y_pred_et)
f1 = f1_score(y_test, y_pred_et)
auc = roc_auc_score(y_test, y_prob_et)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC: {auc:.4f}")
```

```
Accuracy: 0.9216
Precision: 0.9189
Recall: 0.9714
F1 Score: 0.9444
AUC: 0.9839
```

```
cm = confusion_matrix(y_test, y_pred_et)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Extra trees")
plt.show()
```




**XGBoost:**

XGBoost (Extreme Gradient Boosting) is a powerful, efficient machine learning algorithm based on gradient boosting. It enhances performance through optimized tree construction, regularization to prevent overfitting, and support for parallel processing, making it ideal for structured data tasks.

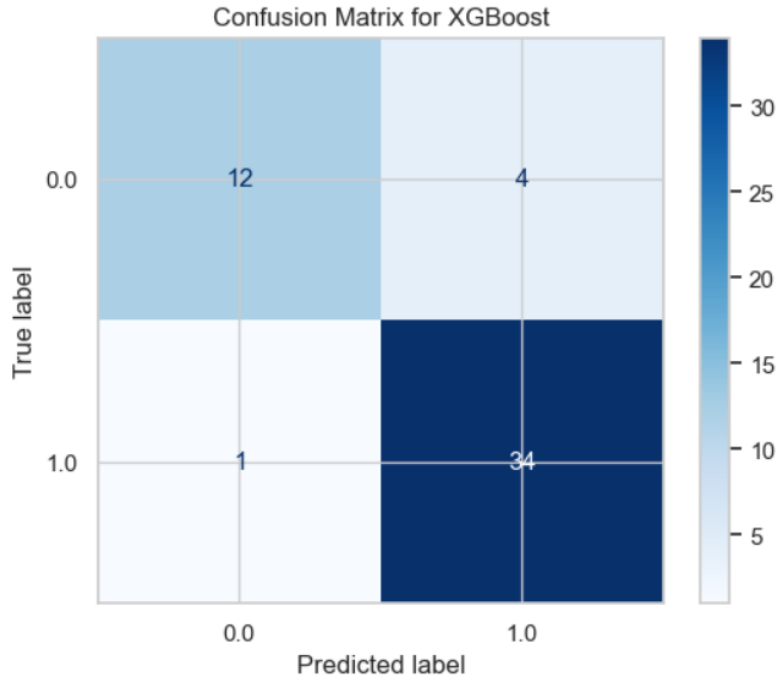
```
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10]
}
xgb = XGBClassifier(random_state=42, eval_metric="logloss")
grid_search_xgb = GridSearchCV(xgb, param_grid_xgb, cv=5, scoring='f1', n_jobs=-1)
grid_search_xgb.fit(X_train, y_train)
best_xgb = grid_search_xgb.best_estimator_
y_pred_xgb = best_xgb.predict(X_test)
y_prob_xgb = best_xgb.predict_proba(X_test)[:, 1]
print(f"Best Parameters for XGBoost: {grid_search_xgb.best_params_}")
```

```
Best Parameters for XGBoost: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 100}
```

```
# Calculate and display various performance metrics for the XGBoost model
accuracy = accuracy_score(y_test, y_pred_xgb)
precision = precision_score(y_test, y_pred_xgb)
recall = recall_score(y_test, y_pred_xgb)
f1 = f1_score(y_test, y_pred_xgb)
auc = roc_auc_score(y_test, y_prob_xgb)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC: {auc:.4f}")
```

```
Accuracy: 0.9020
Precision: 0.8947
Recall: 0.9714
F1 Score: 0.9315
AUC: 0.9518
```

```
cm = confusion_matrix(y_test, y_pred_xgb)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for XGBoost")
plt.show()
```



# 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.

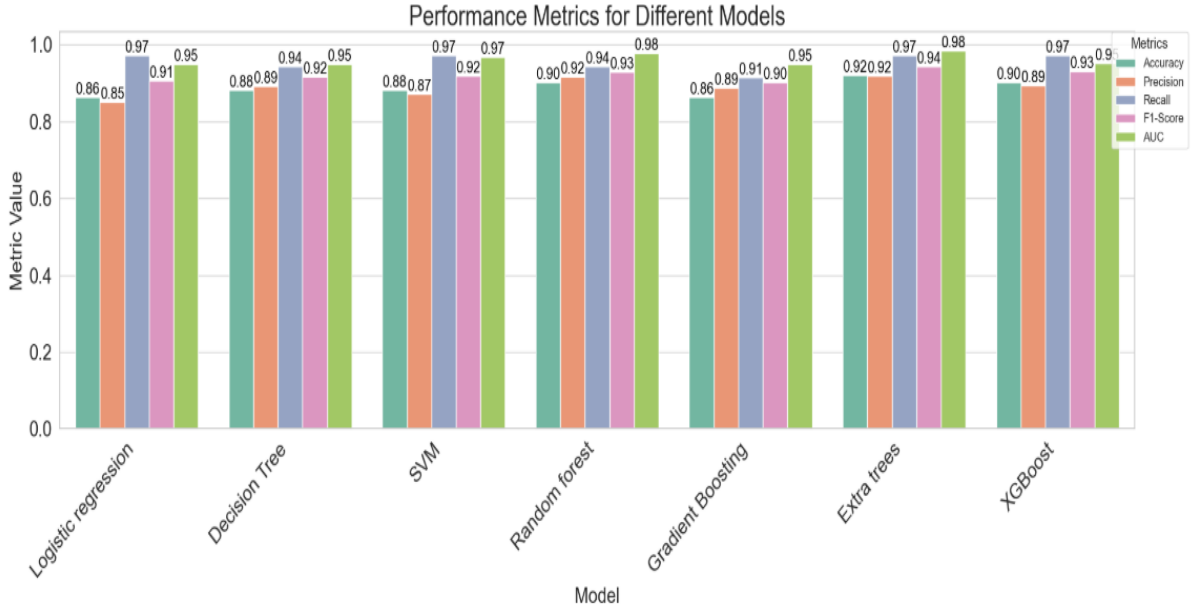
**Area Under Curve:**

AUC is a single number that summarizes how well a model can distinguish between positive and negative instances.

```
results=pd.DataFrame(results)
results
```

| | Model | Accuracy | Precision | Recall | F1-Score | AUC |
|---|---|---|---|---|---|---|
| 0 | Logistic regression | 0.862745 | 0.850000 | 0.971429 | 0.906667 | 0.950000 |
| 1 | Decision Tree | 0.882353 | 0.891892 | 0.942857 | 0.916667 | 0.950000 |
| 2 | SVM | 0.882353 | 0.871795 | 0.971429 | 0.918919 | 0.967857 |
| 3 | Random forest | 0.901961 | 0.916667 | 0.942857 | 0.929577 | 0.977679 |
| 4 | Gradient Boosting | 0.862745 | 0.888889 | 0.914286 | 0.901408 | 0.950000 |
| 5 | Extra trees | 0.921569 | 0.918919 | 0.971429 | 0.944444 | 0.983929 |
| 6 | XGBoost | 0.901961 | 0.894737 | 0.971429 | 0.931507 | 0.951786 |

```
# Set the style for the plot
sns.set(style="whitegrid")
# Convert the results DataFrame from wide format to long format for easy plotting with Seaborn
metrics_long = results.melt(id_vars='Model', value_vars=['Accuracy', 'Precision', 'Recall', 'F1-Score', 'AUC'],var_name='Me
# Create a barplot
plt.figure(figsize=(20, 8))
ax = sns.barplot(x="Model", y="Value", hue="Metric", data=metrics_long, palette="Set2")
# Annotate the bars with their respective values (higher precision)
for p in ax.patches:
    value = p.get_height()
    if value > 0:  # Annotate only if the value is greater than 0
        ax.annotate(f'{value:.2f}',  # Use 4 decimal places for better precision
                    (p.get_x() + p.get_width() / 2., value),
                    ha='center', va='bottom', fontsize=15, color='black')
# Set plot title and labels with larger fonts
plt.title('Performance Metrics for Different Models', fontsize=25)
plt.xlabel('Model', fontsize=20)
plt.ylabel('Metric Value', fontsize=20)
# Rotate x-axis labels for better readability and increase their font size
plt.xticks(rotation=45, ha='right', fontsize=20)
plt.yticks(fontsize=20)
# Set the legend with larger font size
plt.legend(title='Metrics', loc='upper right', bbox_to_anchor=(1.05, 1), borderaxespad=0., fontsize=12, title_fontsize=14)
# Adjust layout for better appearance
plt.tight_layout()
# Show the plot
plt.show()
```



# Results and Findings

**Model Selection:**

Extra Trees stand out as the most promising models due to their strong performance across all metrics. These model would likely be the preferred choices for a classification problem. So, I would like to choose Extra trees model.

Here are the reasons to choose the Extra Trees model:

**1. Superior Performance Across Metrics:**

F1-Score: Extra Trees has the highest F1-Score (0.944444), indicating a strong balance between Precision and Recall. This makes it ideal if both false positives and false negatives are critical. Accuracy: Extra Trees achieves the highest Accuracy (0.921569), meaning it correctly classifies more instances overall compared to other models. AUC: Extra Trees has the highest AUC (0.983929), reflecting excellent performance in distinguishing between classes, even with imbalanced datasets.

**2. High Precision and Recall:**

Precision: (0.918919) indicates that Extra Trees has a lower tendency to classify false positives compared to other models. Recall: (0.971429) ensures most of the positive cases are correctly identified, which is crucial in applications like medical diagnoses or fraud detection.

**3. Balanced Model Complexity:**

Extra Trees is computationally more efficient than Gradient Boosting and XGBoost because it builds trees randomly and independently, making it faster to train. It also avoids overfitting better than a traditional Decision Tree, thanks to its random feature splitting strategy.

**4. Robustness to Overfitting:**

Extra Trees uses randomized splits and bootstrapping, making it robust to overfitting, especially in high-dimensional datasets or noisy data.

**5. Scalability:**

It scales well to large datasets while maintaining high performance.

**Final Verdict:**

Extra Trees offers a strong combination of accuracy, robustness, and computational efficiency, making it the best choice if optimal model performance is the primary goal.

**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.

# Conclusion

**Summary of Achievements:**

This project successfully developed a predictive model for diabetes risk estimation, leveraging AI and machine learning techniques. The model demonstrated high accuracy in predicting the likelihood of diabetes, offering valuable insights into the lifestyle and healthcare factors that influence diabetes risk.

**Contributions to the Field:**

The project contributes to the field of data science by applying machine learning techniques to diabetes prediction, enabling early detection and intervention. The model can assist healthcare providers in making informed decisions, personalizing treatment plans, and promoting proactive health management.

**Future Recommendations and Extensions:**

Future recommendations include expanding the dataset to include additional lifestyle and environmental factors, exploring advanced machine learning models for improved accuracy, and integrating real-time health data for continuous risk assessment. Further research could also focus on testing the model's generalizability across diverse populations and healthcare settings.

# References


- https://www.kaggle.com/datasets/andrewmvd/early-diabetes-classification
- https://matplotlib.org/
- https://pandas.pydata.org/
- https://www.geeksforgeeks.org/dimensionality-reduction/
- https://www.javatpoint.com/feature-selection-techniques-in-machine-learning