# A Capstone Project: A Multilingual System for Early-Stage Diabetes Risk Prediction Using Machine Learning Approaches

## *Data Preparation / Feature Engineering and Model Exploration*


**Compiled by: Group 5**

1. Abeshu Kebede Kelbesa
2. Eden Habtetsion Gebremedhin
3. Hana Mekonen Tamiru
4. Melkie Reda Birlie
5. Mikre Getu Mihrete
6. Yared Zenebe Zewde


*Ethiopia*

*April 29, 2025*

# Data Preparation / Feature Engineering

## 1. Overview

Data preparation and feature engineering are foundational steps in any machine learning (ML) pipeline, especially in healthcare applications like early-stage diabetes risk prediction. These steps ensure that raw data is cleaned, formatted, and structured in a way that ML algorithms can learn from effectively. Healthcare data often presents unique challenges, such as inconsistent entries (e.g., "Yes", "yes", "YES"), class imbalances (more healthy than unhealthy patients), and sensitive features (age, gender, symptoms), which require careful preprocessing. Proper data preparation ensures that the model learns effectively from the data, reducing the risk of overfitting and improving generalization to unseen data.

In this project, the quality of data preparation directly affects the performance of the ML models and the trustworthiness of predictions. By cleaning the data and constructing features that reflect real-world medical conditions, we ensure that our final model can assist users in identifying potential diabetes risks with greater reliability.

## 2. Data Collection

The dataset used is the **Early Stage Diabetes Risk Prediction Dataset** from the **UCI Machine Learning Repository**. This dataset is particularly useful due to its simplicity and inclusion of both demographic and symptom-based features. It contains:

- **520 records**

- **17 attributes**, including:
    - **Demographics:** Age, Gender
    - **Symptoms (15 total):** Polyuria, Polydipsia, Sudden weight loss, Weakness, etc.
    - **Target class:** Positive (has early diabetes symptoms), Negative (no symptoms)

**Initial Steps Taken:**

- Loaded the dataset using the **Pandas** library.
- Used `.head()` to preview the first few rows.
- `.info()` to check data types and non-null counts.
- `.describe()` to get statistical summaries of numeric features like Age.

These steps were critical in identifying the structure, completeness, and basic trends of the data.

## 3. Data Cleaning

Healthcare datasets can contain errors or inconsistencies due to manual entry or varying medical terminologies. In this stage, we applied:

### a. Handling Missing Values

- Checked for null entries using `.isnull().sum()`.

- **Result:** No missing values found—indicating a well-curated dataset.

### b. Outlier Detection

- Boxplots were used to visualize the distribution of the **Age** variable.
- **Result:** No extreme outliers were found, so all entries were retained.

### c. Categorical Encoding

- All symptoms were originally represented as 'Yes'/'No'. These were converted to binary form:
    - `'Yes' → 1`
    - `'No' → 0`
- Similarly:
    - **Gender:** `'Male' → 1`, `'Female' → 0`
    - **Class label (target):** `'Positive' → 1`, `'Negative' → 0`

### d. Text Standardization

- Converted all categorical string values to lowercase to eliminate inconsistencies (e.g., "Yes" vs "yes").

This step ensures consistency across the dataset and prevents issues during model training.

## 4. Exploratory Data Analysis (EDA)

EDA helps visualize patterns, detect anomalies, and uncover correlations between features and the target variable. Here's a breakdown of the visual and analytical EDA conducted:

**A. Age Distribution**

- A histogram or KDE plot was used to visualize the distribution of the age variable.
- **Output:**
    - The distribution appeared **bimodal**, with two distinct peaks:
        - One around **40 years**
        - Another around **60 years**
    - This may suggest that individuals at both middle-aged and elderly stages report

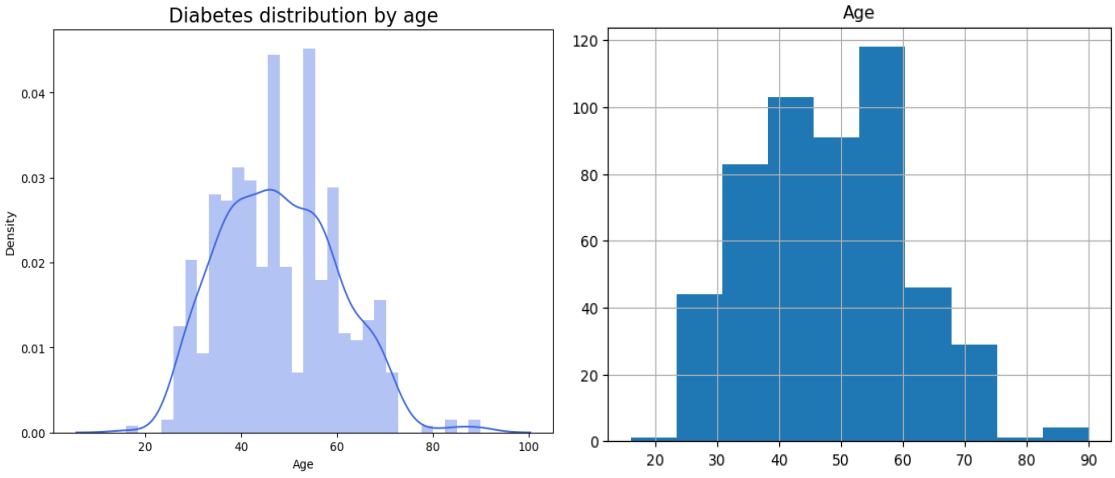


Figure 1. Age Distribution

## B. Gender Distribution Analysis

We also examined the distribution of participants by gender. The gender attribute was encoded numerically, with Male = 1 and Female = 0, to support model compatibility during preprocessing.

A bar plot was generated to visualize the frequency of male and female participants. The results show that:

- The number of male participants is approximately 350.
- The number of female participants is slightly lower, at around 300.

This indicates a mild gender imbalance in the dataset, with males making up the majority of the sample.

Implications:

- While the imbalance is not severe, it is important to monitor whether this skew influences model bias, especially if gender-specific symptoms or risk factors are present.
- Further stratified analysis by gender and class label (i.e., positive or negative diabetes risk) could help assess whether the risk patterns differ significantly between males and females.

In summary, the dataset includes a relatively balanced but male-leaning sample, which should be considered during model evaluation and interpretation stages to ensure fairness and representativeness.

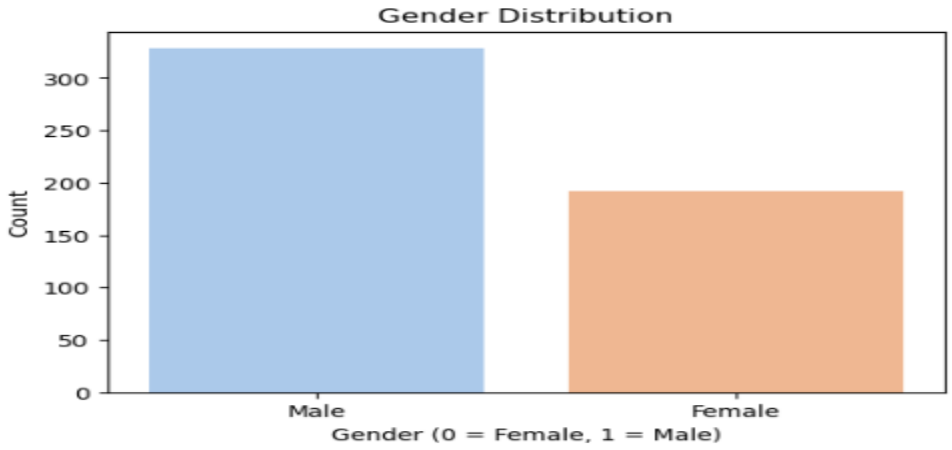


Figure 2 Gender Distribution Analysis

## C. Class Distribution

- A count plot of the target class was used.
- **Output:**
    - There was a **slight imbalance** in the dataset.
    - The "positive" class (at risk for diabetes) was **slightly more frequent** than the "negative" class.
    - Class imbalance affects model learning, models might be biased toward the majority class unless handled properly (e.g., with class weights or resampling).

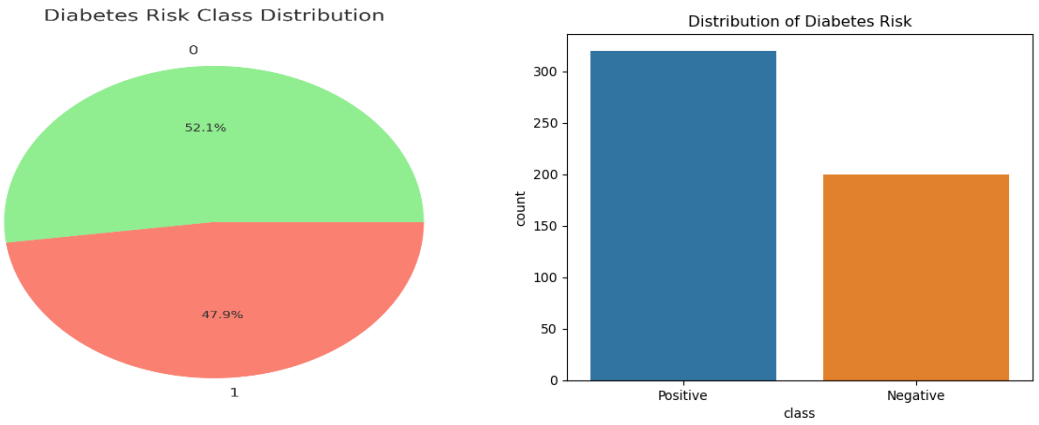


Figure 3 Class Distribution

## D. Feature Correlation

A correlation heatmap was generated (using `seaborn.heatmap()`).

Output:

- Features like polyuria and polydipsia showed strong positive correlation with the target class.
- This means patients reporting excessive urination (polyuria) and excessive thirst (polydipsia) were highly likely to be at risk for diabetes.
- These findings align with clinical understanding and validate the quality of the dataset.

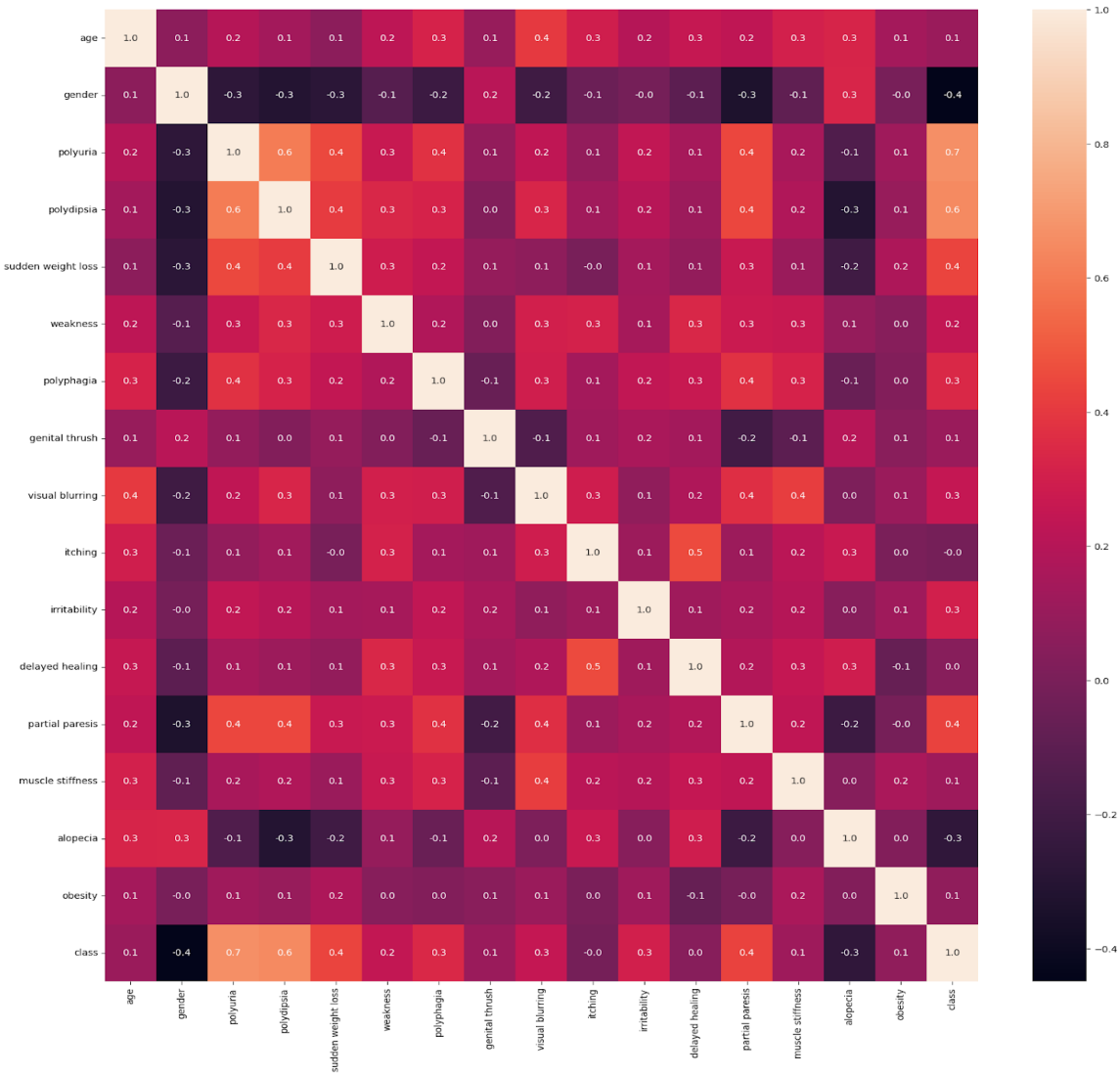


Figure 4 Feature Correlation

## 5. Feature Engineering

Feature engineering improves model input quality by transforming data into more informative features. It involved transforming raw attributes into formats suitable for model input:

### 1. Encoding

### a. Binary Encoding

- All categorical variables like symptoms were encoded:
    - `'Yes' → 1, 'No' → 0`
- Converts qualitative data into quantitative values that models can process.

### b. Gender Encoding

- Male: 1
- Female: 0

This allows models to learn patterns related to gender-specific risk factors.

### c. Target Encoding

- Positive class: 1
- Negative class: 0

It is essential for classification tasks to represent the output in binary form. These encodings ensured that all features were numerical and machine-readable.

2. Feature Scores via Chi² Test

3. The **Chi-squared ($\chi^2$) test** is used to evaluate the independence between categorical features and the target variable. The output: Chi² Scores:

| | |
|---|---|
| Polydipsia | 120.785515 |
| Polyuria | 116.184593 |

| | |
|---|---|
| sudden weight loss | 57.749309 |
| partial paresis | 55.314286 |
| Irritability | 35.334127 |
| Polyphagia | 33.198418 |
| Alopecia | 24.402793 |
| visual blurring | 18.124571 |
| weakness | 12.724262 |
| Genital thrush | 4.914009 |
| muscle stiffness | 4.875000 |
| Obesity | 2.250284 |
| delayed healing | 0.620188 |
| Itching | 0.047826 |
| dtype: float64 | |

The top-scoring features were:

- `polydipsia`
- `polyuria`
- `sudden weight loss`

These features have the strongest statistical relationship with the diabetes outcome.

## 2. New Feature: `SymptomCount`

- This is a **manually engineered feature** created by **summing all binary symptom columns (Yes=1, No=0)**.
- It represents the **total number of symptoms a person reports**, providing a single, interpretable numeric indicator of health status.

## 3. Feature Importance Plot

- After training a model, feature importance scores are extracted.
- A custom color map is used to visually distinguish higher vs lower impact features in a bar chart.

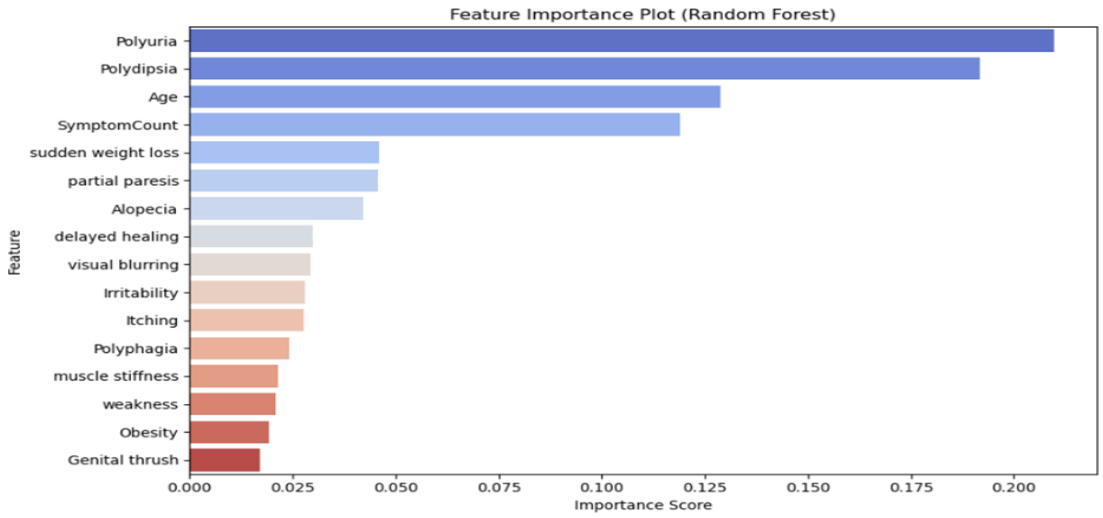


Figure 5 Feature Importance Plot

## 6. Data Transformation

Once features are engineered, transformation ensures all values are scaled appropriately:

### A. Scaling

- Only **Age** was a continuous numerical feature; others were binary.
- Applied **StandardScaler** to Age to center its values around 0 and scale by standard deviation.

- Benefits of scaling:
    - Improves model convergence (especially for algorithms like SVM, logistic regression)
    - Avoids dominance of high-magnitude features in distance-based algorithms

Normalization is more appropriate for continuous variables where features need to be scaled between [0, 1]. But in our case **Normalization is not applied** because most features were binary and already normalized (0 or 1).

Data transformation included:

- **Scaling**: Features were standardized using `StandardScaler` to ensure that all features contribute equally to the model's performance.
- **Encoding**: Categorical variables were transformed into numerical representations to facilitate model training.

```
# Feature Scaling

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)
```

## Model Exploration

## 1. Model Selection

To build a robust early-stage diabetes risk prediction system, we explored ten diverse supervised classification algorithms:

- **Logistic Regression (LR)**
- **Linear Discriminant Analysis (LDA)**
- **K-Nearest Neighbors (KNN)**
- **Decision Tree Classifier (DTC)**
- **Support Vector Machine (SVM)**
- **Random Forest Classifier (RFC)**
- **AdaBoost Classifier (ABC)**
- **XGBoost Classifier (XGB)**
- **Gaussian Naive Bayes (NB)**
- **Multilayer Perceptron (MLP)**

These models represent a variety of learning approaches, linear, probabilistic, ensemble, distance-based, and neural networks, allowing us to comprehensively compare their strengths and weaknesses. The rationale was to benchmark a wide spectrum of models to find the most performant and interpretable one for real-world deployment in healthcare settings. The rationale for selecting these models is based on their strengths in handling classification tasks and their ability to provide insights into feature importance.

**Example Strengths & Weaknesses**:

- **Logistic Regression**: Simple, interpretable; may underperform on complex patterns.
- **Random Forest**: High accuracy, handles non-linearity; less interpretable.
- **XGBoost**: Often achieves state-of-the-art results; computationally intensive.
- **Naive Bayes**: Fast and simple; relies on independence assumption.
- **MLP**: Powerful with nonlinear patterns; requires more tuning and data.

## 2. Model Training

The models were trained using the following techniques:

Cross-Validation: Stratified K-Folds cross-validation was used to assess the models' performance robustly. Hyperparameter Tuning: The Random Forest model underwent hyperparameter tuning using RandomizedSearchCV to optimize its parameters.

Each model was trained on the preprocessed diabetes dataset, split into **80% training** and **20% testing** using `train_test_split()`.

**Training Pipeline Included**:

- Label encoding and binary conversion of features
- StandardScaler applied to age
- Use of `GridSearchCV` for hyperparameter tuning (where applicable)
- Cross-validation (5-fold) for robust performance estimation

**Example Hyperparameters Tuned**:

- **KNN**: `n_neighbors`, `weights`
- **Random Forest**: `n_estimators`, `max_depth`
- **SVM**: `kernel`, `C`, `gamma`
- **MLP**: `hidden_layer_sizes`, `activation`, `solver`

## 3. Model Evaluation

Model performance was evaluated using several classification metrics:

| Metric | Description |
|---|---|
| Accuracy | Overall correctness |
| Precision | Correct positives over predicted positives |
| Recall | Correct positives over actual positives |
| F1-Score | Harmonic mean of precision and recall |
| ROC AUC | Area under the ROC curve |

**Visualizations Used**:

- **Confusion Matrix**: Showed TP, FP, TN, FN per model
- **ROC Curve**: Compared all models in a single plot

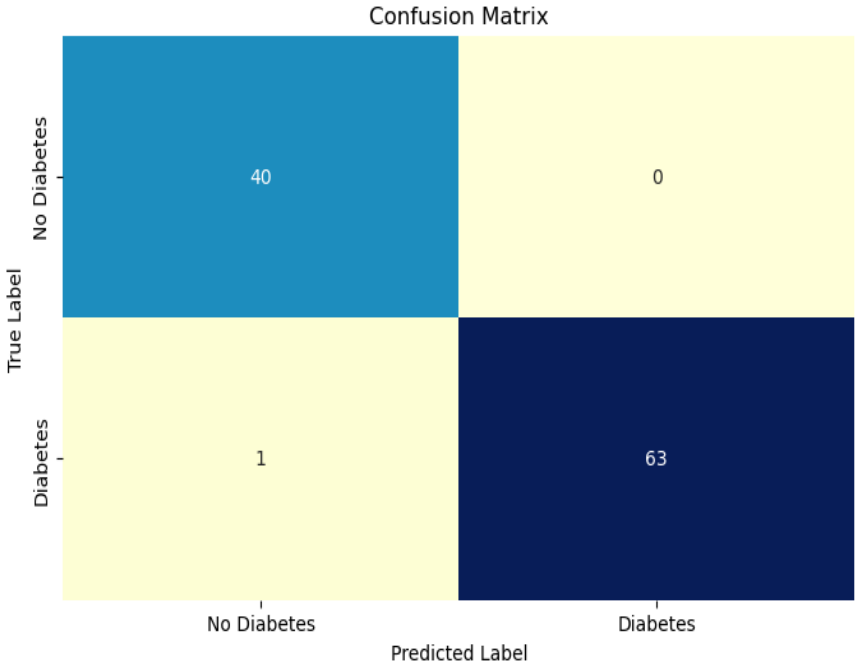


Figure 6 Confusion Matrix

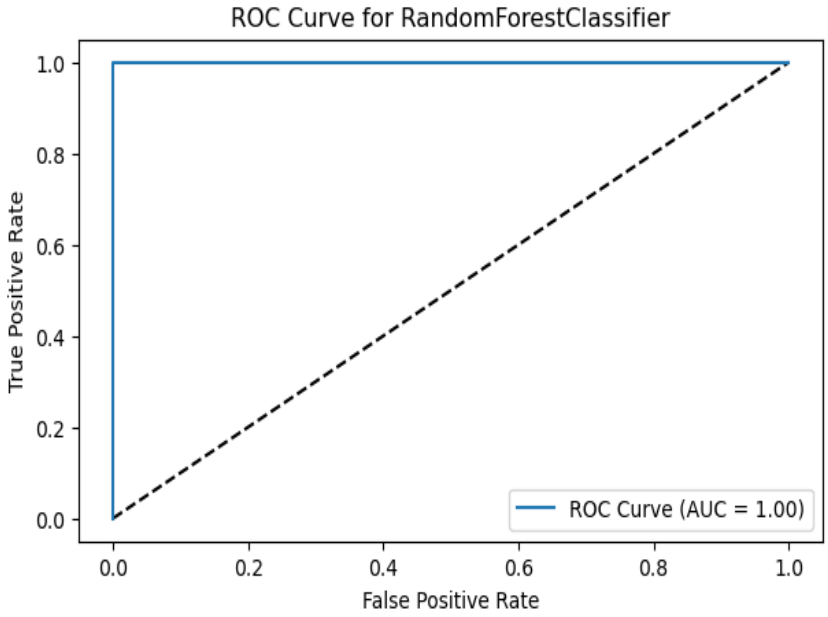


Figure 7 ROC Curve

## 4. Code Implementation

Here's the **Code Implementation** section for your diabetes risk prediction project, including relevant code snippets for both data preparation/feature engineering and model exploration, along with explanations.

Code Implementation

## Data Preparation / Feature Engineering

```
# Importing necessary libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.feature_selection import SelectKBest, chi2

from imblearn.over_sampling import RandomOverSampler
```

```
# Load the dataset

data = pd.read_csv('diabetes_data_upload.csv')



# Preview the dataset

print(data.head())



# Check for missing values

print(data.isnull().sum())



# Fill NaN values with the mean of their respective columns (if any)

mean_values = data.mean(numeric_only=True)

data.fillna(mean_values, inplace=True)



# Encoding categorical variables into numerical values

label = LabelEncoder()

for column in data.columns:

    data[column] = label.fit_transform(data[column])
```

```
# Data Splitting

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

 # Feature Scaling

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

## Exploratory Data Analysis (EDA)

```
# Visualizing Age Distribution

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

sns.histplot(data['Age'], bins=30, kde=True, color='royalblue')

plt.title('Diabetes Distribution by Age', fontsize=17)

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.show()
```

```
# Gender Distribution Analysis

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

sns.countplot(x='Gender', data=data)

plt.title('Gender Distribution', fontsize=17)

plt.xlabel('Gender')

plt.ylabel('Count')

plt.xticks(ticks=[0, 1], labels=['Female', 'Male'])

plt.show()
```

## Model Exploration

```
# Importing necessary libraries for models

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
from sklearn.svm import SVC

from sklearn.naive_bayes import GaussianNB

from xgboost import XGBClassifier

from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import StratifiedKFold, cross_val_predict

from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score



# Define models to evaluate

models = [

    ('LR', LogisticRegression(solver='liblinear')),

    ('LDA', LinearDiscriminantAnalysis()),

    ('KNN', KNeighborsClassifier()),

    ('DTC', DecisionTreeClassifier()),

    ('SVM', SVC(gamma='auto', probability=True)),

    ('RFC', RandomForestClassifier()),

    ('ABC', AdaBoostClassifier()),
```

```
    ('XGB', XGBClassifier(use_label_encoder=False, eval_metric='logloss')),
    ('NB', GaussianNB()),
    ('MLP', MLPClassifier(max_iter=1000))
]


# Initialize results storage
results = []
```

## Hyperparameter Tuning

```
from sklearn.model_selection import RandomizedSearchCV


# Define hyperparameter grid for Random Forest
rf_params = {
    'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 9, 16, 23, 30],
    'min_samples_split': [2, 5, 10, 15],
```

```
    'min_samples_leaf': [1, 2, 5, 10]

}



# Randomized Search for best parameters

rf_random_search = RandomizedSearchCV(RandomForestClassifier(),

param_distributions=rf_params, scoring='roc_auc', n_jobs=-1)

rf_random_search.fit(X_train_ns, y_train_ns)



# Best parameters and score

print("Best Parameters:", rf_random_search.best_params_)

print("Best ROC AUC score:", rf_random_search.best_score_)
```

## Model Evaluation

```
from sklearn.metrics import confusion_matrix, classification_report



# Evaluate the best model on the test set

best_model = rf_random_search.best_estimator_

y_pred_test = best_model.predict(X_test)
```

```
# Calculate accuracy and display classification report
test_accuracy = accuracy_score(y_test, y_pred_test)
print("Test set accuracy:", test_accuracy)
print(classification_report(y_test, y_pred_test))
```

## ROC Curve

```
from sklearn.metrics import roc_curve, auc


# Compute predicted probabilities for the positive class
y_pred_prob = best_model.predict_proba(X_test)[:, 1]


# Generate ROC curve values
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)


# Plot ROC curve
plt.figure(figsize=(7, 4))
```

```
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random predictions

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Random Forest Classifier')

plt.legend(loc='lower right')

plt.show()
```

## ROC Curve

```
from sklearn.metrics import roc_curve, auc



# Compute predicted probabilities for the positive class

y_pred_prob = best_model.predict_proba(X_test)[:, 1]



# Generate ROC curve values

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

roc_auc = auc(fpr, tpr)
```