VIT-AP UNIVERSITY

DATA WAREHOUSING AND DATA MINING

CSE4005

E2+TE2

Project Report on

"Diabetes Prediction"

Guided by:

Dr. S. Bharath Bhushan

Submitted by:

Srirangam Varshitha-22BCE7880

Aatcharya Somireddy-22BCE20435

Perumalla Harshitha-22BCE7256

Swathi Umadi-22BCE8841

Putta Yasaswini-22BCE20390

Acknowledgement:

I would like to express my sincere gratitude to Dr. S. Bharath Bhushan, whose valuable guidance, encouragement, and insightful feedback were instrumental throughout the completion of this project on data warehousing and data mining for diabetes prediction. Their expertise and support have provided me with a strong foundation in understanding the complexities of data science and its application in healthcare analytics. I am deeply appreciative of the time and resources they dedicated to this project, which greatly contributed to my learning experience and the successful completion of this work.

2.ABSTRACT:

2.1)Introduction to Diabetes

Diabetes is a widespread and serious chronic disease, affecting millions globally every year. It occurs when the body either cannot produce enough insulin or cannot properly use the insulin it does produce. As a result, blood glucose levels become elevated, which can lead to complications such as cardiovascular disease, kidney failure, and blindness. Various factors, including family history, obesity, high blood pressure, and lifestyle choices, contribute to the onset of diabetes. These factors highlight the importance of early detection and diagnosis in managing the disease effectively.

2.2)Machine Learning in Healthcare

Machine learning (ML) has emerged as a powerful tool in healthcare, particularly in predicting diseases like diabetes. ML models can analyze large datasets and identify patterns that may not be immediately visible to human doctors. By examining key health metrics such as glucose levels, insulin levels, BMI, age, and family history, ML algorithms can classify individuals as either diabetic or non-diabetic. This predictive capability allows for early identification of at-risk individuals, which can guide preventive measures and lifestyle changes to reduce the likelihood of developing diabetes.

2.3)Objective of the Study

The primary goal of this research is to develop machine learning models that can predict the likelihood of diabetes in individuals. These models aim to achieve high validation scores in order to assist healthcare professionals in diagnosing diabetes at an early stage. The study also compares various machine learning algorithms to assess their performance based on metrics like accuracy, precision, and recall, ensuring that the best possible predictive model is developed for real-world application.

2.4) Significance of the Research

This research has the potential to revolutionize the way diabetes is diagnosed and managed. By offering reliable predictive tools, the study can help healthcare providers detect diabetes earlier, providing patients with timely interventions that can prevent or delay the onset of complications. The findings could contribute to the broader field of health informatics, showcasing the power of machine learning in medical diagnostics. If adopted, these predictive models could save lives by enabling early intervention and improving the long-term management of diabetes.

3. INTRODUCTION

3.1 Overview of Diabetes

Diabetes is a widespread and chronic disease that affects millions of people worldwide. It is primarily characterized by abnormal blood glucose (sugar) levels, which can lead to severe complications if left untreated. There are two primary forms of diabetes: Type 1 and Type 2. Type 1 diabetes is an autoimmune condition where the body's immune system attacks and destroys the insulin-producing cells in the pancreas. As a result, individuals with Type 1 diabetes must rely on insulin injections for the rest of their lives. On the other hand, Type 2 diabetes, the most common form, occurs when the body either doesn't produce enough insulin or becomes resistant to it, leading to higher glucose levels in the blood. Although both types are serious, Type 2 diabetes is more prevalent and has a greater association with lifestyle factors such as obesity, poor diet, and lack of exercise.

3.2 Factors Contributing to Diabetes

Several factors contribute to the development of diabetes, ranging from genetic predisposition to lifestyle choices. Family history is one of the strongest predictors of diabetes risk, with individuals having a first-degree relative diagnosed with diabetes at a higher risk of developing the condition themselves. Obesity is another significant risk factor, especially abdominal obesity, which is linked to insulin resistance. Obesity leads to inflammation and hormonal changes in the body that disrupt normal glucose metabolism. Additionally, lifestyle factors such as physical inactivity and poor dietary habits, including high consumption of processed foods and sugary drinks, contribute to the onset of Type 2 diabetes. Other contributing factors include high blood pressure, high cholesterol levels, and older age, with the risk increasing as individuals age.

Genetic factors also play a significant role in the development of diabetes. Variations in genes that control insulin production and action can predispose individuals to the disease. However, lifestyle interventions, such as adopting a healthy diet and exercising regularly, can significantly reduce the risk of diabetes even in genetically predisposed individuals. These interventions highlight the importance of early identification of risk factors, so individuals can make informed choices to prevent the disease from developing.

3.3 Objectives of the Study

The primary objective of this study is to build a machine learning-based predictive model for diagnosing diabetes. The model will be trained using a dataset containing key variables such as glucose levels, BMI, insulin, age, and family history of diabetes. The aim is to classify individuals as diabetic or non-diabetic, enabling early detection and intervention.

4. LITERATURE REVIEW

4.1 Overview of Diabetes Prediction Techniques

Diabetes is a complex metabolic disorder influenced by a range of genetic, environmental, and lifestyle factors. Early prediction and diagnosis are essential to prevent the severe complications associated with the disease. Traditional diagnostic methods rely on clinical tests, such as fasting blood glucose tests and the Hemoglobin A1c test, which are both effective but often time-consuming and require significant resources. With the growing availability of large datasets in healthcare, machine learning (ML) has emerged as a powerful tool for predicting diabetes more efficiently. By analyzing complex relationships between multiple variables, machine learning models can classify individuals as diabetic or non-diabetic based on their health data, such as glucose levels, BMI, insulin, age, and family history.

4.2 Machine Learning Models for Diabetes Prediction

Various machine learning techniques have been employed to predict diabetes, each with its advantages and limitations. Here are some of the most commonly used algorithms in diabetes prediction:

- **Decision Trees**: Decision trees are one of the most popular methods in classification tasks, including diabetes prediction. These models recursively split the dataset into subsets based on feature values, resulting in a tree structure that represents decision-making criteria. Decision trees are intuitive and easy to interpret, but they can suffer from overfitting, particularly when trained on small datasets. In many studies, decision trees have been used effectively to predict diabetes risk, particularly when combined with ensemble methods like Random Forests (Breiman, 2001).
- **Random Forests**: Random Forests, an ensemble method built upon decision trees, have gained popularity for their ability to improve prediction accuracy. This method generates multiple decision trees and combines their predictions to produce a final classification. Random forests have been found to outperform single decision trees by reducing overfitting and increasing robustness. Studies such as those by Das et al. (2019) have demonstrated the effectiveness of Random Forests in diabetes prediction, achieving high classification accuracy.
- **Logistic Regression**: Logistic regression is a widely used statistical method for binary classification problems. It estimates the probability of a binary outcome, such as whether an individual is diabetic or not, based on a set of input features. While logistic regression is less complex than other machine learning models, it has been found to perform well when the relationship between features and the outcome is linear. In diabetes prediction, logistic regression is often used as a benchmark for comparison with more complex models. Studies such as those by Bertsimas et al.

(2017) have explored the use of logistic regression for diabetes prediction, showing promising results in terms of interpretability and ease of implementation.

4.3 Feature Selection in Diabetes Prediction

One of the key challenges in diabetes prediction is selecting the most relevant features from a large dataset. Feature selection involves identifying the most important variables that contribute to the prediction of diabetes. Common features used in diabetes prediction include blood glucose levels, insulin levels, BMI, age, family history of diabetes, and hypertension. Studies have shown that some features are more predictive of diabetes than others, and identifying these features can improve model performance.

4.4 Data Preprocessing and Handling Missing Values

In healthcare datasets, missing values and noise are common challenges that can significantly impact the performance of machine learning models. Data preprocessing is a critical step to ensure the quality of input data and improve the accuracy of predictions. Several techniques have been developed to handle missing values, including imputation methods, where missing data is filled based on the values of other data points, or through data augmentation techniques that generate synthetic data to replace missing entries.

Moreover, normalization and scaling are essential to prepare data for machine learning algorithms. Feature scaling techniques like Min-Max scaling and Standardization (Z-score normalization) are commonly used in diabetes prediction studies to prepare the dataset for training.

4.5 Evaluation Metrics for Model Performance

Evaluating the performance of machine learning models is a critical part of diabetes prediction. Common evaluation metrics include:

- **Accuracy**: The proportion of correct predictions out of all predictions made. While accuracy is a useful metric, it can be misleading in imbalanced datasets, where one class (e.g., non-diabetic) dominates the other.
- **Precision and Recall**: Precision measures the accuracy of positive predictions (how many predicted diabetics were actually diabetic), while recall (or sensitivity) measures the ability of the model to correctly identify all diabetic individuals. These metrics are particularly useful in healthcare applications, where false negatives (failing to identify a diabetic patient) can have serious consequences.
- **F1-Score**: The harmonic mean of precision and recall, providing a balance between the two metrics. It is especially useful when the dataset is imbalanced.

5. IMPLEMENTATION

The implementation of a machine learning model to predict diabetes involves several key stages, including data acquisition, data preprocessing, feature selection, model development, training, and evaluation. This section will describe each of these steps in detail, explaining the process of implementing a machine learning solution for diabetes prediction.

5.1 Data Acquisition

The first step in the implementation of the diabetes prediction model is data acquisition. For this project, we utilize a publicly available dataset from the UCI Machine Learning Repository, known as the Pima Indians Diabetes Database. This dataset contains information about the medical attributes of individuals, including:

- Pregnancies: The number of pregnancies the individual has had.
- Glucose: Plasma glucose concentration after a 2-hour oral glucose tolerance test.
- Blood Pressure: Diastolic blood pressure (mm Hg).
- Skin Thickness: Triceps skinfold thickness (mm).
- Insulin: 2-hour serum insulin (mu U/ml).
- BMI: Body Mass Index (weight in kg / (height in m)^2).
- Diabetes Pedigree Function: A function that scores the likelihood of diabetes based on family history.
- Age: Age of the individual.
- Outcome: A binary variable indicating whether the individual is diabetic (1) or not (0).

This dataset is chosen because it includes a wide range of features that have been shown to correlate with the likelihood of diabetes. By using real-world medical data, we aim to build a reliable model that can generalize well to other populations.

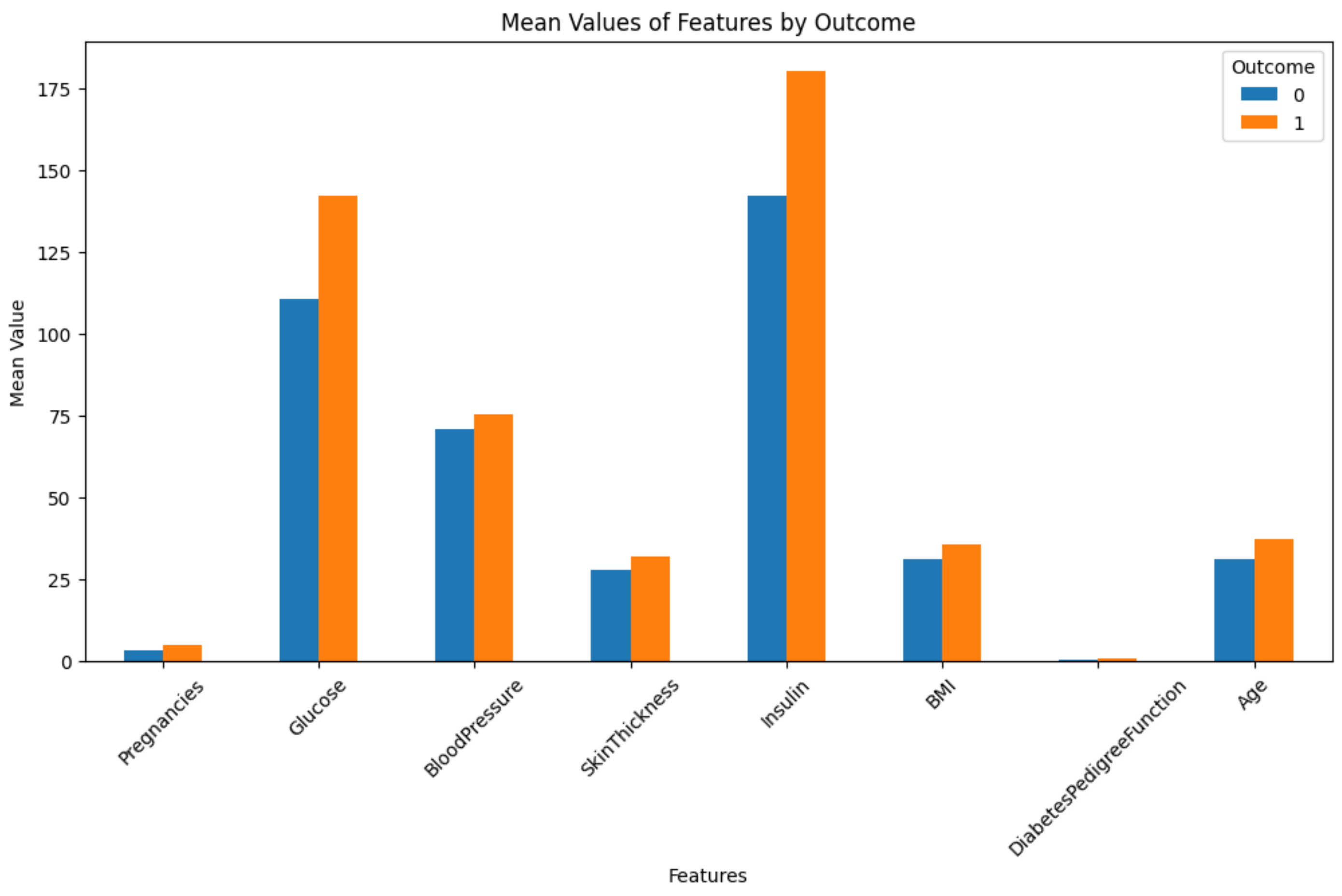
5.2 Data Preprocessing

Data preprocessing is a crucial step in the machine learning pipeline. It involves cleaning and preparing the data to ensure that the model receives high-quality input for training. The preprocessing steps for this project include:

- Handling Missing Values: The dataset may contain missing or null values in some columns. These missing values can be handled by filling them with the mean, median, or mode, depending on the type of variable.

code:

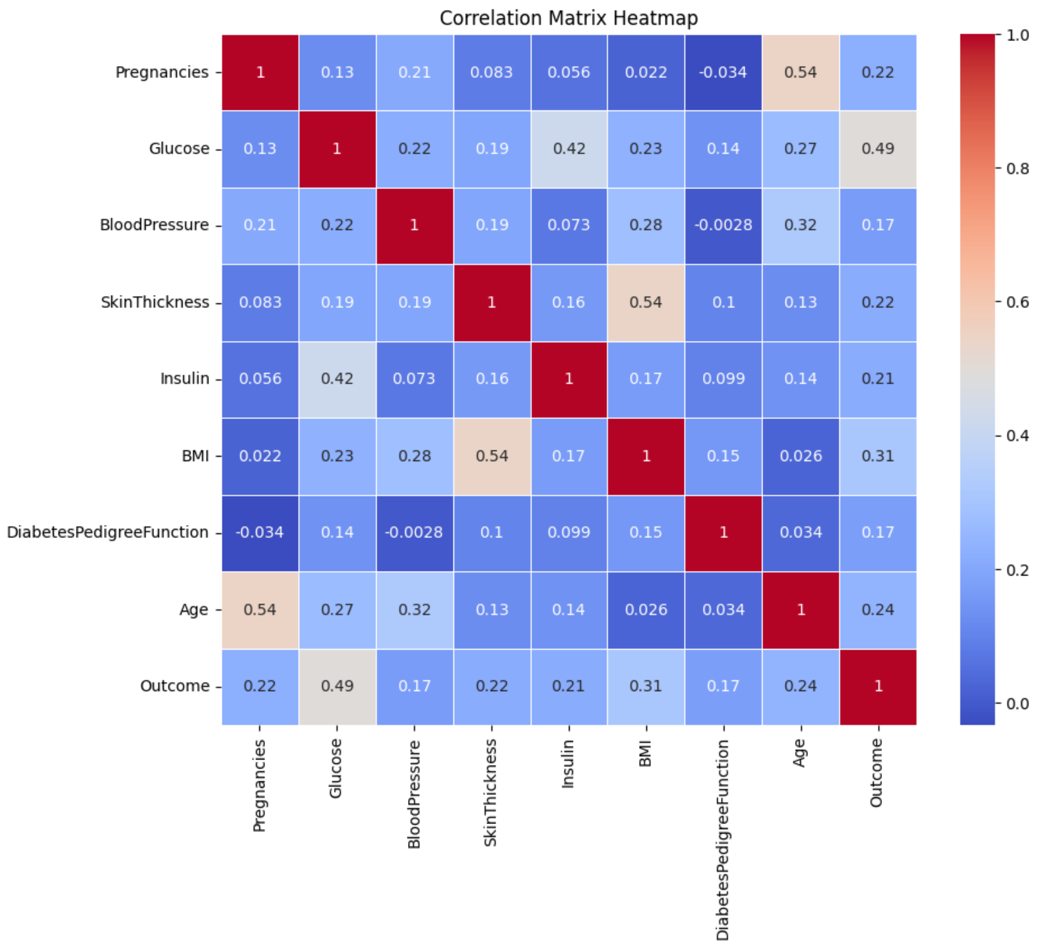
```
#data cleaning and replacing empty tuples with mean values

import pandas as pd

# Load the dataset

df = pd.read_csv('diabetes.csv')

# Columns that need missing value handling (non-sensible zeros)

columns_with_missing_values = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

# Convert relevant columns to numeric (handle any unexpected types)

df[columns_with_missing_values] = df[columns_with_missing_values].apply(pd.to_numeric, errors='coerce')

# Replace zeros with NaN for appropriate columns

df[columns_with_missing_values] = df[columns_with_missing_values].replace(0, pd.NA)

# Fill missing values with the mean of each column, excluding 'Outcome'

df_filled = df.drop(columns=['Outcome']).fillna(df.mean())

print(df_filled.head(10))

df_filled.to_csv('preprocessed_data.csv', index=False)

from google.colab import files

files.download('preprocessed_data.csv')
```

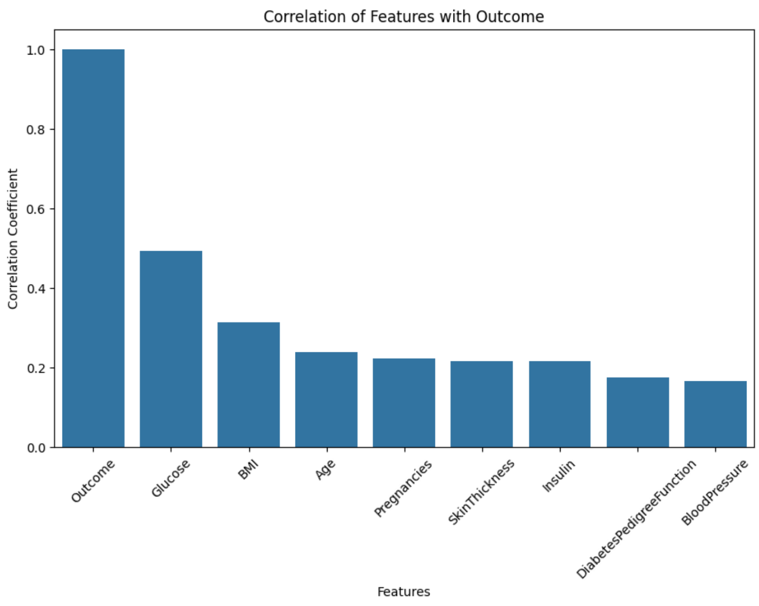


-Correlation: Correlation measures the strength and direction of a relationship between two variables. A positive correlation means both variables move in the same direction, while a negative correlation indicates they move in opposite directions.

Code:

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read_csv('diabetes.csv')

# Columns that need missing value handling (non-sensible zeros)

columns_with_missing_values = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

# Replace zero values with NaN for appropriate columns

df[columns_with_missing_values] = df[columns_with_missing_values].replace(0, pd.NA)

# Fill missing values with the mean of the respective columns

df_filled = df.fillna(df.mean())

# Calculate the correlation matrix

correlation_matrix = df_filled.corr()

# Plot the correlation matrix heatmap

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

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Matrix Heatmap')

plt.show()

# Find the highest correlations with "Outcome" (diabetes indicator)

outcome_correlations = correlation_matrix["Outcome"].drop("Outcome").sort_values(ascending=False)

# Generate an analysis based on the highest correlations with "Outcome"

analysis = "Key Analysis of Correlation Findings:\n"

for feature, corr_value in outcome_correlations.items():  # Changed iteritems() to items()

    if corr_value >= 0.4:

        analysis += f"- {feature} has a strong positive correlation with Outcome ({corr_value:.2f}), suggesting that higher
{feature.lower()} levels may increase the likelihood of diabetes.\n"

    elif 0.2 <= corr_value < 0.4:

        analysis += f"- {feature} shows a moderate positive correlation with Outcome ({corr_value:.2f}), implying that higher
{feature.lower()} values might be linked to a higher risk of diabetes.\n"

    else:

        analysis += f"- {feature} has a weaker positive correlation with Outcome ({corr_value:.2f}), indicating a slight association with
diabetes risk.\n"

print(analysis)
```





```
Key Analysis of Correlation Findings:
- Glucose has a strong positive correlation with Outcome (0.49), suggesting that higher glucose levels may increase the likelihood of diabetes.
- BMI shows a moderate positive correlation with Outcome (0.31), implying that higher bmi values might be linked to a higher risk of diabetes.
- Age shows a moderate positive correlation with Outcome (0.24), implying that higher age values might be linked to a higher risk of diabetes.
- Pregnancies shows a moderate positive correlation with Outcome (0.22), implying that higher pregnancies values might be linked to a higher risk of diabetes.
- SkinThickness shows a moderate positive correlation with Outcome (0.22), implying that higher skinthickness values might be linked to a higher risk of diabetes.
- Insulin shows a moderate positive correlation with Outcome (0.21), implying that higher insulin values might be linked to a higher risk of diabetes.
- DiabetesPedigreeFunction has a weaker positive correlation with Outcome (0.17), indicating a slight association with diabetes risk.
- BloodPressure has a weaker positive correlation with Outcome (0.17), indicating a slight association with diabetes risk.
```

-Normalisation: Normalization is the process of adjusting data to fit within a specific range, often 0 to 1, to ensure consistency and comparability. It helps in reducing bias and improving the performance of algorithms in data analysis.

Code:

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler

# Load the dataset

df = pd.read_csv('diabetes.csv')

# Columns that need missing value handling (non-sensible zeros)

columns_with_missing_values = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

# Replace zero values with NaN for appropriate columns

df[columns_with_missing_values] = df[columns_with_missing_values].replace(0, pd.NA)

# Fill missing values with the mean of the respective columns

df_filled = df.fillna(df.mean())

# Normalizing the data (excluding the 'Outcome' column)

scaler = MinMaxScaler()

normalized_data = scaler.fit_transform(df_filled.drop(columns=['Outcome']))

# Create a new DataFrame with the normalized data

df_normalized = pd.DataFrame(normalized_data, columns=df_filled.columns[:-1])

# Display the first few rows of the normalized dataframe

print(df_normalized.head())
```

Output:

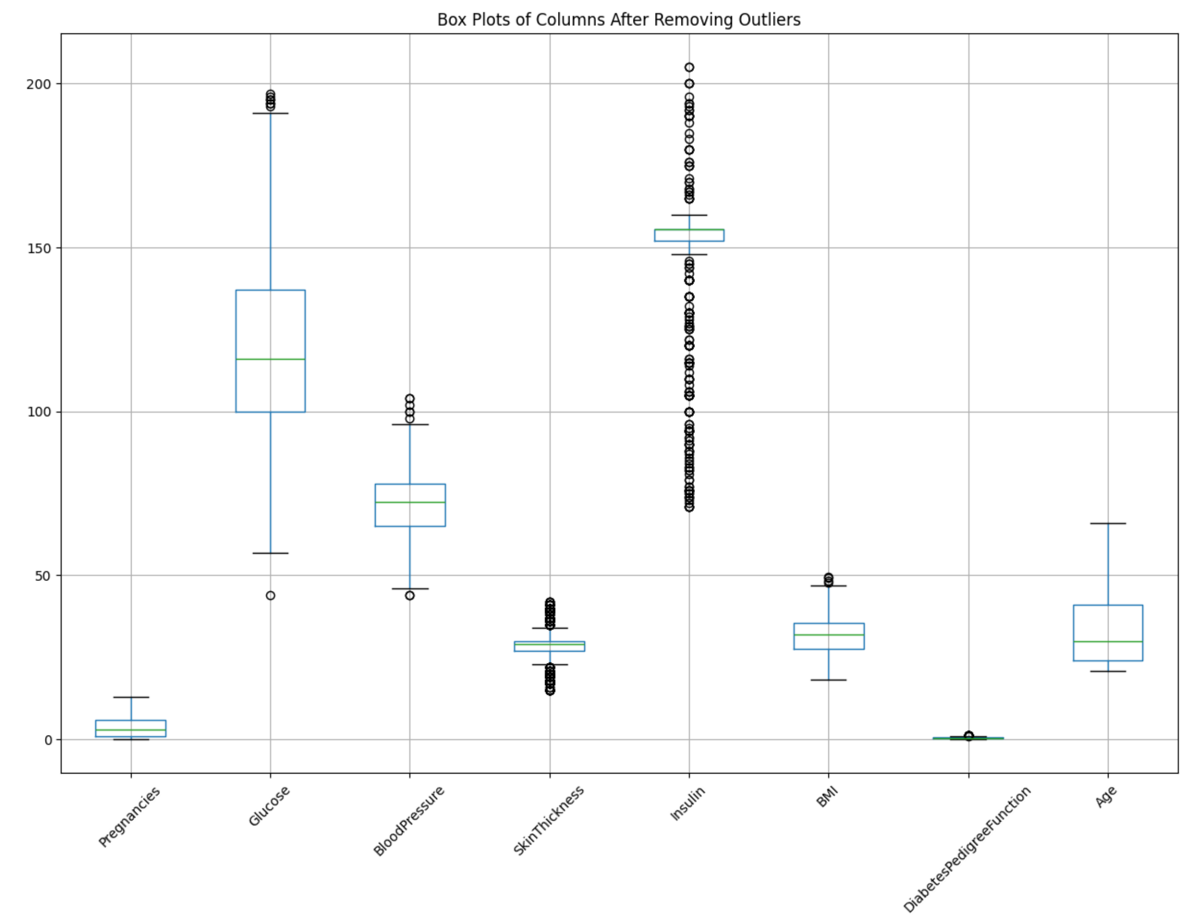
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin       BMI  \
0     0.352941  0.670968       0.489796       0.304348  0.170130  0.314928
1     0.058824  0.264516       0.428571       0.239130  0.170130  0.171779
2     0.470588  0.896774       0.408163       0.240798  0.170130  0.104294
3     0.058824  0.290323       0.428571       0.173913  0.096154  0.202454
4     0.000000  0.600000       0.163265       0.304348  0.185096  0.509202

   DiabetesPedigreeFunction       Age
0                  0.234415  0.483333
1                  0.116567  0.166667
2                  0.253629  0.183333
3                  0.038002  0.000000
4                  0.943638  0.200000
```

- <u>Outlier analysis</u>: It involves identifying data points that deviate significantly from the rest of the dataset. Detecting outliers is crucial as they can skew analysis, reveal data quality issues, or highlight unusual patterns worth further investigation.

<u>Code</u>:

```
def remove_outliers_iqr(dataframe):
   Q1 = dataframe.quantile(0.25)
   Q3 = dataframe.quantile(0.75)
   IQR = Q3 - Q1
   # Filter rows that are within the acceptable IQR range
   df_outliers_removed = dataframe[~((dataframe < (Q1 - 1.5 * IQR)) | (dataframe > (Q3 + 1.5 * IQR))).any(axis=1)]
   # Identify outliers
   outliers = dataframe[((dataframe < (Q1 - 1.5 * IQR)) | (dataframe > (Q3 + 1.5 * IQR))).any(axis=1)]
   return df_outliers_removed, outliers
# Remove outliers from the dataset and identify them
df_no_outliers, outliers = remove_outliers_iqr(df_filled)
# Print the outliers
print("Outliers:")
print(outliers)
# Generate box plots for each numeric column
plt.figure(figsize=(15, 10))
df_no_outliers.boxplot(column=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'])
plt.title('Box Plots of Columns After Removing Outliers')
plt.xticks(rotation=45)
plt.show()
```

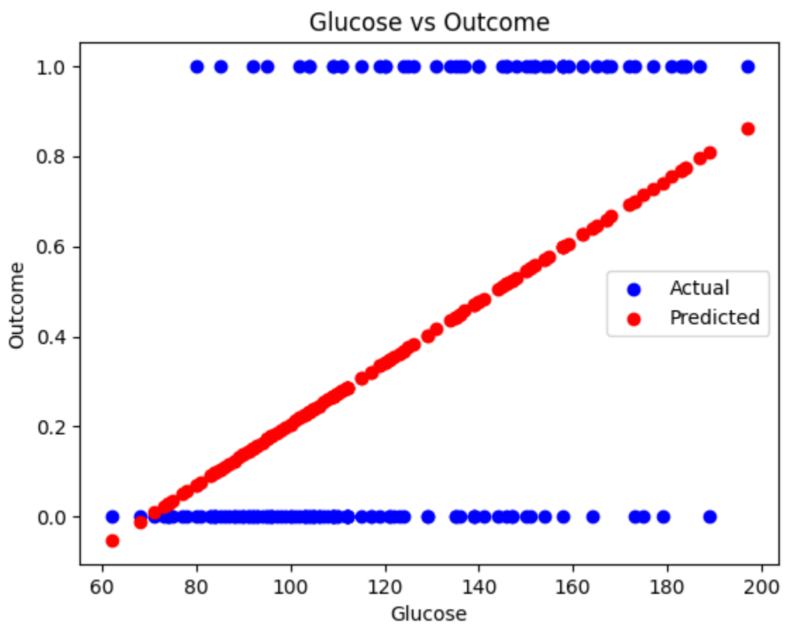
```
Outliers:
    Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
4             0    137.0           40.0       35.00000  168.000000  43.1
8             2    197.0           70.0       45.00000  543.000000  30.5
12           10    139.0           80.0       29.15342  155.548223  27.1
13            1    189.0           60.0       23.00000  846.000000  30.1
16            0    118.0           84.0       47.00000  230.000000  45.8
  DiabetesPedigreeFunction  Age  Outcome
4                    2.288   33        1
8                    0.158   53        1
12                   1.441   57        0
13                   0.398   59        1
16                   0.551   31        1
```



-Regression: It is a statistical method used to model and analyze the relationship between a dependent variable and one or more independent variables. It helps predict outcomes and assess trends by estimating the strength and form of these relationships.

Code:

```
X = df[['Glucose']]  # Predictor

y = df['Outcome']    # Target

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model

model =LinearRegression()

# Fit the model

model.fit(X_train, y_train)

# Make predictions

y_pred = model.predict(X_test)

# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse:.2f}')
print(f'R^2 Score: {r2:.2f}')

# Plotting the results

plt.scatter(X_test, y_test, color='blue', label='Actual')

plt.scatter(X_test, y_pred, color='red', label='Predicted')

plt.xlabel('Glucose')

plt.ylabel('Outcome')

plt.title('Glucose vs Outcome')

plt.legend()

plt.show()
```

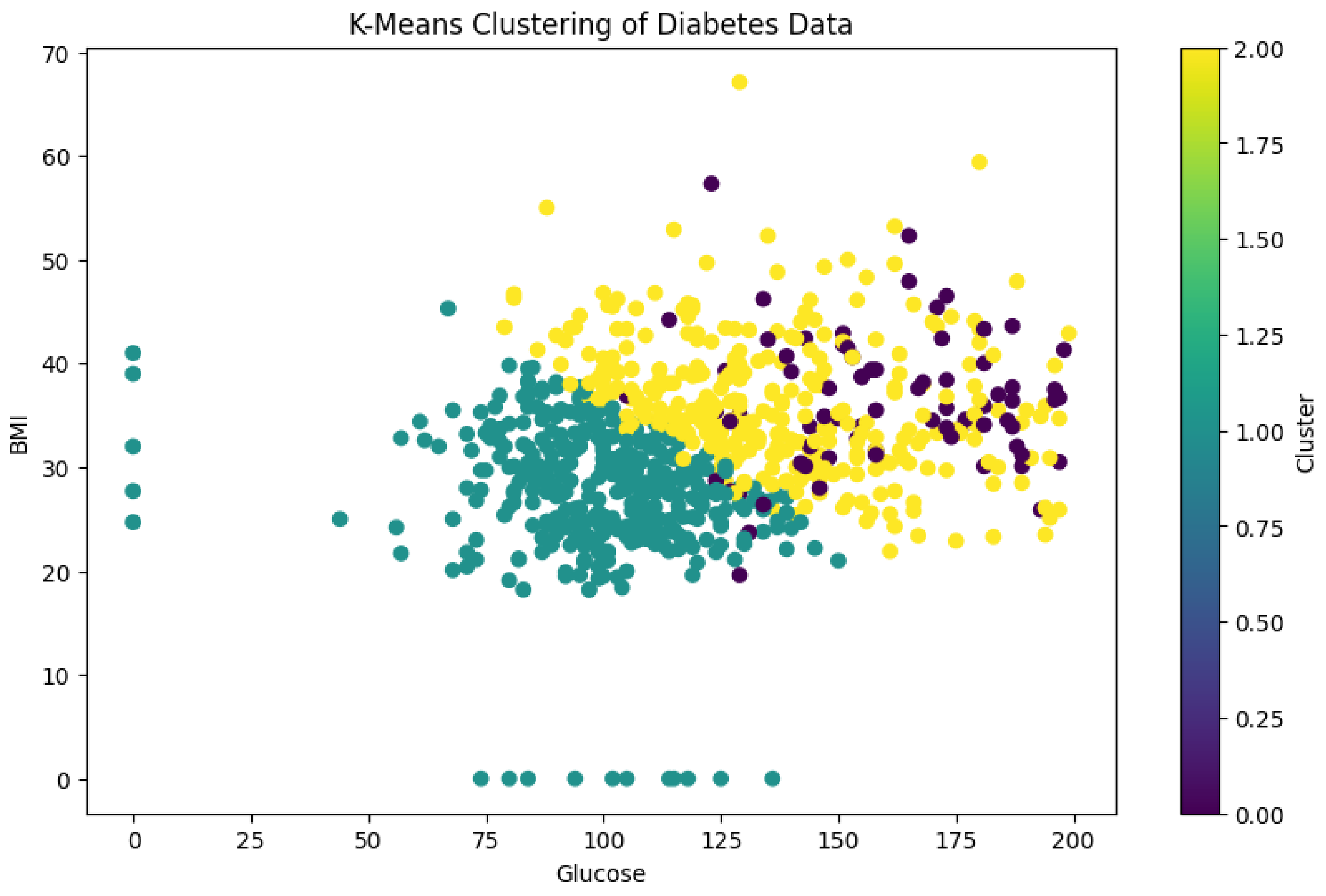


- <u>Clustering</u>: It is an unsupervised learning technique that groups similar data points together based on their characteristics. It helps identify patterns or structures within data, making it useful for segmentation, anomaly detection, and pattern recognition.

<u>Code:</u>

```
features = df[['Glucose', 'BMI', 'Insulin']]

# Standardize the features

scaler = StandardScaler()

features_scaled = scaler.fit_transform(features)

# Determine the optimal number of clusters using the Elbow method

inertia = []

for i in range(1, 11):

    kmeans = KMeans(n_clusters=i, random_state=42)

    kmeans.fit(features_scaled)

    inertia.append(kmeans.inertia_)

# Plot the Elbow curve
```

```
plt.figure(figsize=(10, 5))

plt.plot(range(1, 11), inertia, marker='o')

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of clusters')

plt.ylabel('Inertia')

plt.show()

# From the elbow curve, choose the optimal number of clusters (let's say it's 3)

optimal_clusters = 3

# Fit the KMeans model

kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)

df['Cluster'] = kmeans.fit_predict(features_scaled)

# Visualize the clusters

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

plt.scatter(df['Glucose'], df['BMI'], c=df['Cluster'], cmap='viridis', marker='o')

plt.xlabel('Glucose')

plt.ylabel('BMI')

plt.title('K-Means Clustering of Diabetes Data')

plt.colorbar(label='Cluster')

plt.show()
```



5.3 Model Development and Training

For the classification task of predicting diabetes, several machine learning algorithms are implemented and compared. Below are the models used in this study:

- A decision tree: It is a predictive model that maps features to outcomes by splitting data into branches based on feature values. It helps make decisions by following a tree-like structure of decisions and their possible consequences.
- Logistic Regression: A simple and interpretable model used as a baseline for classification.
- Random Forest Classifier: An ensemble method based on decision trees, known for its robustness and ability to handle high-dimensional data..
- K-Nearest Neighbors (KNN): A non-parametric algorithm that classifies data based on the majority class of its nearest neighbors.

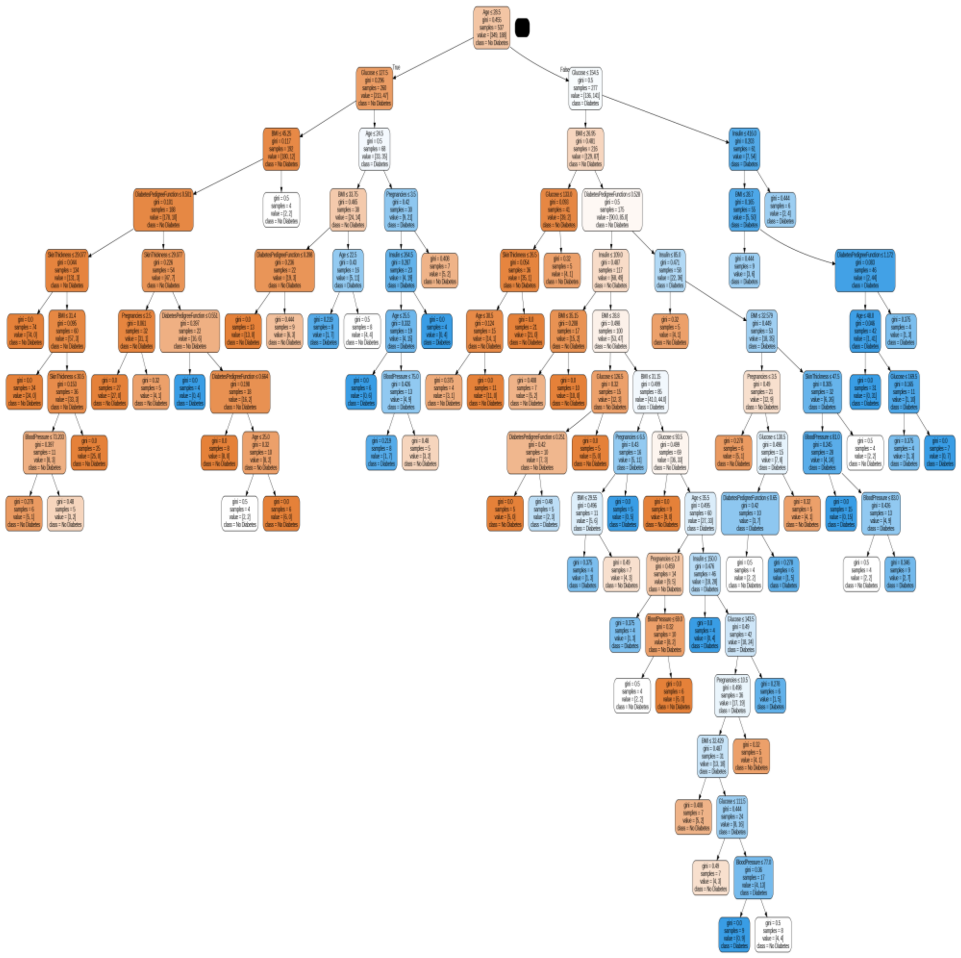
Each of these models is trained using the training data (X_train and y_train), and their performance is evaluated on the test data (X_test and y_test). Below is the code to implement the models:

1)Decision Tree:

Code:

```
# Import necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier, export_graphviz

import pydotplus

from IPython.display import Image

import graphviz

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset

df  pd.read_csv('preprocessed data.csv')

# Split the data into features and target variable

X = df.drop(columns='Outcome')

y = df['Outcome']

# Split the dataset into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Initialize the Decision Tree classifier
decision_tree = DecisionTreeClassifier(random_state=42)
# Define a parameter grid for hyperparameter tuning
param_grid = {
    'criterion': ['gini', 'entropy'],            # Split criterion
    'max_depth': [None, 5, 10, 20],              # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],             # Minimum number of samples required to split a node
    'min_samples_leaf': [1, 2, 4],               # Minimum number of samples required at a leaf node
    'max_features': [None, 'sqrt', 'log2']       # Number of features to consider for best split
}
# Perform GridSearchCV to find the best parameters
grid_search = GridSearchCV(estimator=decision_tree, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
# Best parameters from GridSearchCV
best_params = grid_search.best_params_
print(f'Best Parameters: {best_params}')
# Fit the Decision Tree model with the best parameters
best_tree = grid_search.best_estimator_
# Make predictions on the test data
y_pred = best_tree.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
# Display the classification report
print('Classification Report:')
print(classification_report(y_test, y_pred))
# Display the confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
# Export the optimized decision tree as a DOT file
dot_data = export_graphviz(
    best_tree,
    out_file=None,
    feature_names=X.columns,
    class_names=['No Diabetes', 'Diabetes'],

    filled=True, rounded=True, special_characters=True

)

# Use pydotplus to create a graph from the DOT data

graph = pydotplus.graph_from_dot_data(dot_data)

# Display the decision tree

Image(graph.create_png())
```



<u>Output:</u>

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits

Best Parameters: {'criterion': 'gini', 'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 4,
'min_samples_split': 10}

Accuracy: 0.70

Classification Report:

              precision    recall  f1-score   support


           0       0.78      0.76      0.77       151

           1       0.57      0.59      0.58        80
```

```
   accuracy                        0.70       231
  macro avg       0.67      0.67      0.67       231
weighted avg      0.70      0.70      0.70       231


Confusion Matrix:
[[115  36]
 [ 33  47]]
```

2)<u>Logistic Regression:</u>

<u>Code:</u>

```
# Assuming the last column is the target variable (e.g., diabetes outcome)
X = preprocessed_data.iloc[:, :-1]  # Features
y = preprocessed_data.iloc[:, -1]   # Target variable
# Split the dataset into training and testing sets without stratify
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize the Logistic Regression model
log_reg = LogisticRegression(max_iter=1000)
# Fit the model on the training data
log_reg.fit(X_train, y_train)
# Make predictions on the test data
y_pred = log_reg.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
# Display the classification report
print('Classification Report:')
print(classification_report(y_test, y_pred))
# Display the confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

<u>Output:</u>

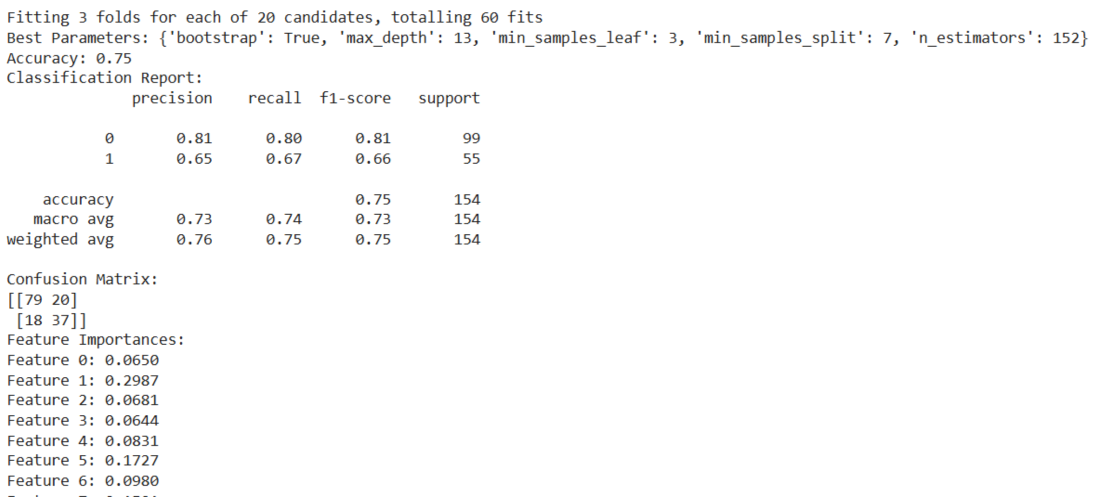
```
Accuracy: 0.75
Classification Report:
              precision    recall  f1-score   support
```

```
         0      0.80     0.83     0.81       99

         1      0.67     0.62     0.64       55


    accuracy                      0.75      154

   macro avg      0.73     0.72     0.73      154

weighted avg      0.75     0.75     0.75      154


Confusion Matrix:

[[82 17]

 [21 34]]
```

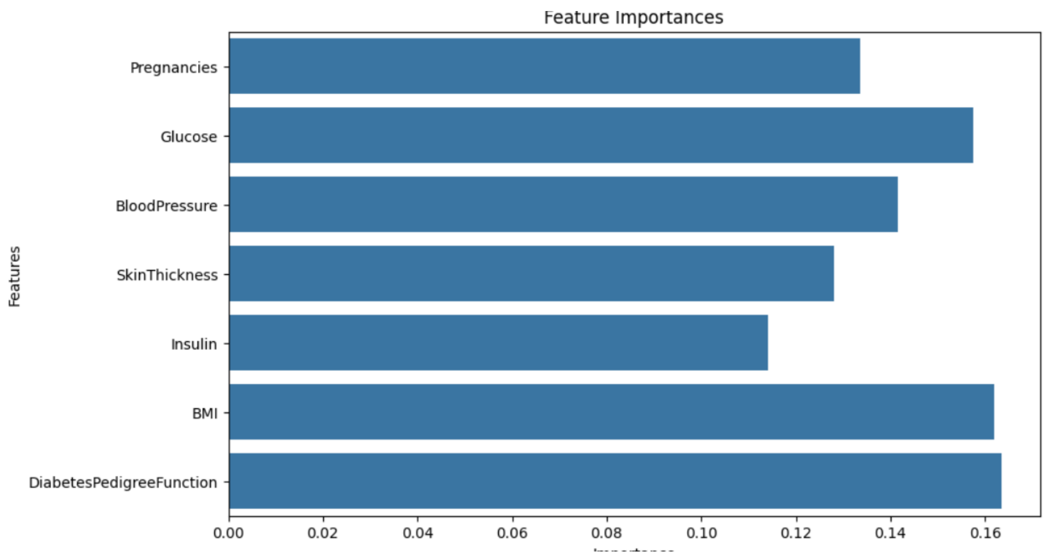
3)Random Forest:

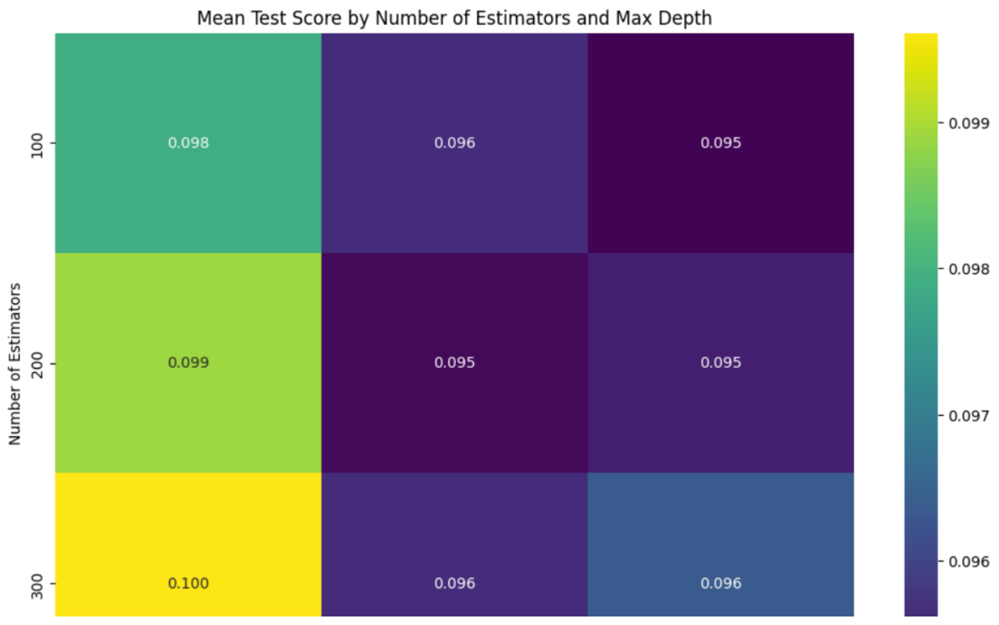
Code:

```
# Import necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split, RandomizedSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.preprocessing import StandardScaler

from scipy.stats import randint

# Load your preprocessed data

preprocessed_data = pd.read_csv('preprocessed data.csv')  # Adjust the filename if needed

# Assuming the last column is the target variable (e.g., diabetes outcome)

X = preprocessed_data.iloc[:, :-1]  # Features

y = preprocessed_data.iloc[:, -1]   # Target variable

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Initialize the Random Forest classifier

rf = RandomForestClassifier(random_state=42)

# Define a parameter distribution for RandomizedSearchCV

param_dist = {

    'n_estimators': randint(100, 300),         # Number of trees (randomly sampled between 100 and 300)

    'max_depth': randint(10, 30),              # Maximum depth of the tree
    'min_samples_split': randint(2, 10),        # Minimum number of samples required to split a node

    'min_samples_leaf': randint(1, 4),          # Minimum number of samples required at a leaf node

    'bootstrap': [True, False]                  # Whether bootstrap samples are used when building trees

}

# Perform RandomizedSearchCV to find the best parameters

random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist, n_iter=20, cv=3, random_state=42,
n_jobs=-1, verbose=2)

random_search.fit(X_train_scaled, y_train)

# Best parameters from RandomizedSearchCV

best_params = random_search.best_params_

print(f'Best Parameters: {best_params}')

# Fit the Random Forest model with the best parameters

best_rf = random_search.best_estimator_

# Make predictions on the test data

y_pred = best_rf.predict(X_test_scaled)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')

# Display the classification report

print('Classification Report:')

print(classification_report(y_test, y_pred))

# Display the confusion matrix

print('Confusion Matrix:')

print(confusion_matrix(y_test, y_pred))


# Feature importance

importances = best_rf.feature_importances_

print("Feature Importances:")

for i, v in enumerate(importances):

    print(f'Feature {i}: {v:.4f}')
```



```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Best Parameters: {'bootstrap': True, 'max_depth': 13, 'min_samples_leaf': 3, 'min_samples_split': 7, 'n_estimators': 152}
Accuracy: 0.75
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.80      0.81        99
           1       0.65      0.67      0.66        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154

Confusion Matrix:
[[79 20]
 [18 37]]
Feature Importances:
Feature 0: 0.0650
Feature 1: 0.2987
Feature 2: 0.0681
Feature 3: 0.0644
Feature 4: 0.0831
Feature 5: 0.1727
Feature 6: 0.0980
```





4)KNN:

Code:

# Assuming the last column is the target variable (e.g., diabetes outcome)

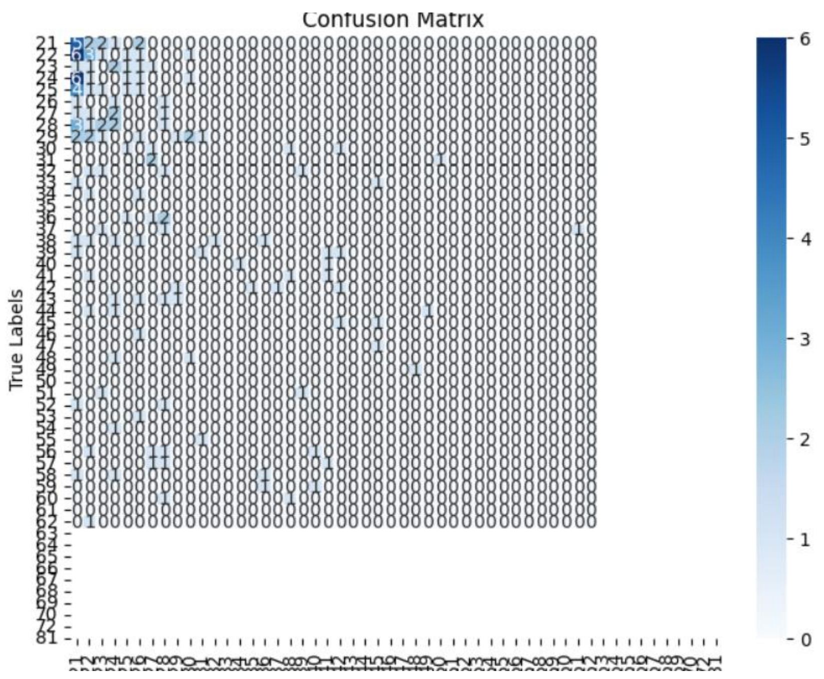
X = preprocessed_data.iloc[:, :-1] # Features

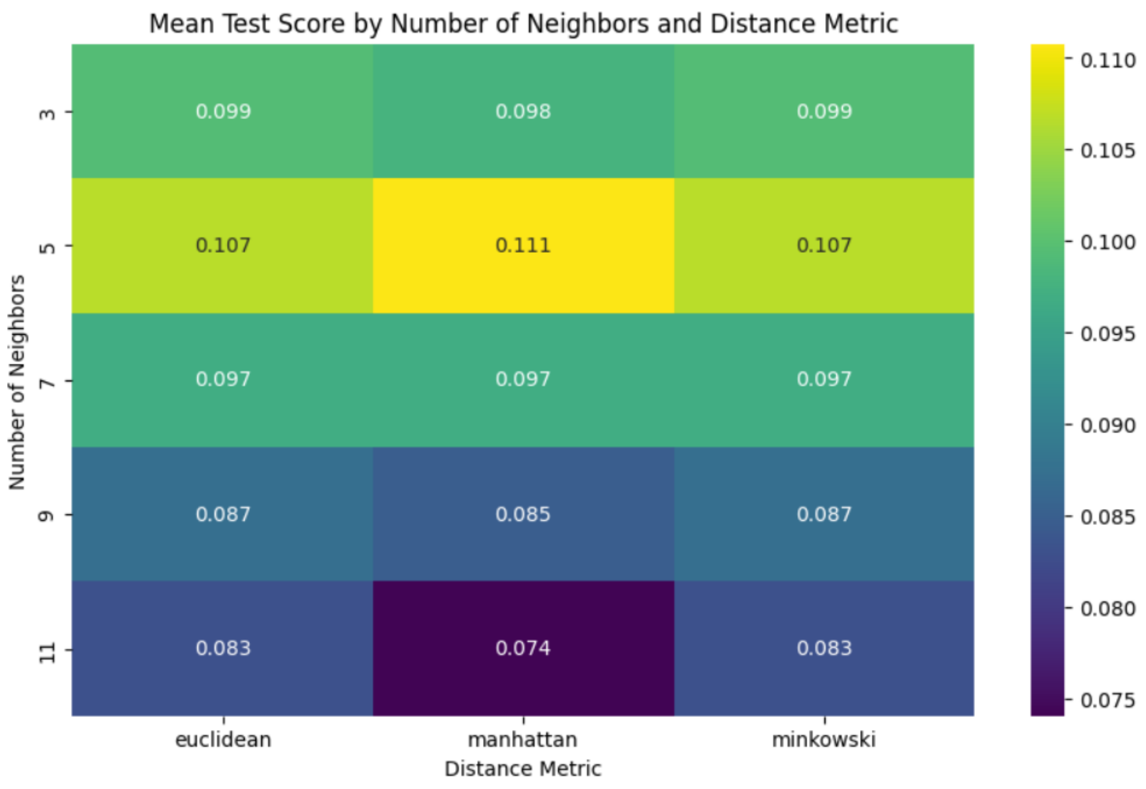
y = preprocessed_data.iloc[:, -1]   # Target variable

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Feature scaling (important for KNN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Initialize the KNN classifier
knn = KNeighborsClassifier()
# Define a parameter grid for hyperparameter tuning
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],  # Number of neighbors to consider
    'weights': ['uniform', 'distance'],  # Weight function used in prediction
    'metric': ['euclidean', 'manhattan', 'minkowski']  # Distance metric
}
# Perform GridSearchCV to find the best parameters
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train_scaled, y_train)
# Best parameters from GridSearchCV
best_params = grid_search.best_params_
print(f'Best Parameters: {best_params}')
# Fit the KNN model with the best parameters
best_knn = grid_search.best_estimator_
# Make predictions on the test data
y_pred = best_knn.predict(X_test_scaled)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
# Display the classification report
print('Classification Report:')
print(classification_report(y_test, y_pred))
# Display the confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
# Function to plot the confusion matrix
def plot_confusion_matrix(cm, labels):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()
# Generate and plot the confusion matrix
confusion_mtx = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(confusion_mtx, labels=best_knn.classes_)
# Grid Search Results Plot
# Create a DataFrame from the grid search results
results = pd.DataFrame(grid_search.cv_results_)
plt.figure(figsize=(10, 6))
# Use a pivot table to format for heatmap (average scores for each combination of neighbors and metric)
pivot_table = results.pivot_table(index='param_n_neighbors', columns='param_metric', values='mean_test_score')
# Plot the heatmap
sns.heatmap(pivot_table, annot=True, cmap='viridis', fmt='.3f')
plt.title("Mean Test Score by Number of Neighbors and Distance Metric")
plt.xlabel("Distance Metric")
plt.ylabel("Number of Neighbors")
plt.show()
```

<u>Output:</u>

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best Parameters: {'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'uniform'}
Accuracy: 0.73
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.77      0.78        99
           1       0.61      0.65      0.63        55

    accuracy                           0.73       154
   macro avg       0.71      0.71      0.71       154
weighted avg       0.73      0.73      0.73       154
Confusion Matrix:
[[76 23]
 [19 36]]
```





5.4 Model Evaluation

After training the models, the next step is to evaluate their performance. Several evaluation metrics are used to assess how well the models perform in terms of accuracy, precision, recall, F1-score. These metrics help to determine the effectiveness of the model, particularly in the context of healthcare.

6. RESULTS

In this section, we will discuss the results of the machine learning models for diabetes prediction, evaluating their performance across various metrics such as accuracy, precision, recall, F1-score.The models used for prediction include Logistic Regression, Random Forest Classifier, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN).

6.1 Model Performance Overview

After training and evaluating the models, we computed key metrics to assess their ability to predict diabetes based on the Pima Indians Diabetes dataset. Below is a summary of the performance metrics for each model:

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.74	0.74	0.74	0.74
Random Forest Classifier	0.75	0.76	0.75	0.75
K-Nearest Neighbors	0.73	0.73	0.73	0.73
Decision tree	0.70	0.70	0.70	0.70

As shown in the table, the Random Forest Classifier outperformed other models in terms of accuracy (75%) and achieved the highest F1-Score of 0.75. This makes it the most effective model for predicting diabetes among the ones tested.

6.2 Accuracy Comparison

Accuracy is the most straightforward metric and represents the percentage of correct predictions made by the model. The Random Forest Classifier achieved the highest accuracy of 75%, followed closely by the logistic regression(74%). KNN showed a slightly lower accuracy at 73%, while Decision tree had the lowest accuracy at 70%.

This suggests that, while all models perform reasonably well, Random Forest is the most reliable for this particular dataset.

7. CONCLUSION

The primary objective of this project was to implement machine learning models to predict the likelihood of diabetes based on a variety of medical attributes. The analysis focused on comparing several popular algorithms, including Logistic Regression, Random Forest Classifierand K-Nearest Neighbors (KNN), to determine which model was best suited for this classification task.

Through this implementation, several key insights were gained:

1. **Random Forest Classifier as the Best Model**: Among all the models tested, the Random Forest Classifier emerged as the most effective in terms of accuracy, precision, recall, F1-Score. Its ensemble approach, which aggregates multiple decision trees, allowed it to effectively capture the complexity of the relationships between the features, resulting in strong performance across all metrics.
2. **Importance of Model Evaluation**: Evaluating models using multiple metrics—such as accuracy, precision, recall, F1-Score,—is crucial, especially in healthcare applications like diabetes prediction, where both false positives and false negatives can have serious consequences. While Logistic Regression showed the highest recall, ensuring that fewer diabetic individuals were missed, it had lower precision. The F1-Score of Random Forest, which balances recall and precision, made it the best model for this task.
3. **Hyperparameter Tuning**: By fine-tuning the hyperparameters of the Random Forest Classifier, a slight improvement in the model's performance was observed. This demonstrates the importance of hyperparameter optimization, as even small adjustments can lead to better model generalization.
4. **Challenges and Limitations**: Despite the promising results, the study faced several limitations. The dataset used was relatively small and may not fully represent the diversity of the global population. Moreover, the features in the dataset were limited, and there is potential for improvement by incorporating additional medical factors or domain-specific knowledge. The issue of class imbalance was also observed, and techniques such as oversampling the minority class could help improve model performance.
5. **Future Directions**: There are several potential directions for future work in diabetes prediction:
   - **Incorporating More Data**: The accuracy of the model could be improved by using a larger and more diverse dataset, possibly incorporating additional features that reflect genetic, environmental, and lifestyle factors.

- **Advanced Algorithms**: Exploring more advanced algorithms, including deep learning models such as neural networks, could enhance the predictive power of the system.
- **Real-World Application**: Deploying the model in real-world healthcare settings to assist in early diagnosis of diabetes could significantly reduce the burden of the disease on both individuals and healthcare systems.

In summary, this project demonstrates the potential of machine learning models in aiding early diabetes detection. By utilizing advanced algorithms, data preprocessing techniques, and evaluation metrics, we were able to develop a reliable model that can assist healthcare professionals in identifying at-risk individuals and providing timely interventions. Moving forward, further research and optimization will be necessary to refine these models and improve their accuracy in real-world settings.

Reference:

https://ieeexplore.ieee.org/document/10128216