

MACHINE LEARNING

Evaluation of Machine Learning Models for Predicting Diabetes Onset Using Patient Medical Records

PROJECT REPORT

MAHNDSE23.2F-007 - J.G. Sachini Gayanika

MADHNSE23.2F-008 - W.G. Ashen Wishwa Geeth Jayarathna

Higher Diploma in Software Engineering 23.2 Full Time
National Institute of Business Management

Declaration

I certify that this project does not incorporate without acknowledgment any material previously submitted for a Diploma in any institution and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where a due reference is made in the text. I also hereby give consent for my project report, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organizations.

MAHNDSE23.2F-007 J.G. Sachini Gayanika

.....

MADHNSE23.2F-008 W.G. Ashen Wishwa Geeth Jayarathna

.....

.....

.....

Date

Ms. Kaushalya
Dissanayake,
Supervisor, Consultant/lecturer,
National Institute of Business Management.

Acknowledgment

I would like to express my sincere appreciation to everyone who contributed to the successful completion of this assignment. First and foremost, I extend my gratitude to our lecturer, MS. Kaushalya Dissanayake, for providing me with the opportunity to explore and deepen my knowledge of the entire course. I would also like to thank my fellow batch members for their commitment to the project. Their insights and hard work greatly enriched the final deliveries. Additionally, I am thankful for the support and resources provided by our institution, which made this assignment possible. Finally, I appreciate the patience and understanding of our friends and family during the intense work on this project. Their encouragement and support were invaluable throughout this journey.

Abstraction

The objective of this work is to compare the performance of different types of machine learning algorithms in diagnosing diabetes based on patient's clinical history. In this study, variables/attributes with input data include glucose level, blood pressure, insulin level, BMI, age and other attributes. We explore and preprocess the data, followed by training and evaluating four machine learning models: The methodologies analyzed in this Article are Decision Tree, k-Nearest Neighbors or KNN, Support Vector Machine or SVM and Random Forest. The measure of the performance of each model includes accuracy, precision, recall, F1 score, and ROC-AUC. The purpose is to determine which model can contribute to early diabetes detection and enable healthcare workers to intervene effectively to enhance patients' experience.

Table of Content

Table of Contents

Chapter 1	1
Introduction.....	1
Background	1
Problem Statement	1
Aims and Objectives	2
Chapter 2.....	3
Data Exploration	3
Chapter 3.....	4
Data Preprocessing.....	4
Missing Values	4
Outlier Detection and Removal.....	5
Chapter 4.....	6
Diagrams	6
Chapter 5.....	15
Decision Tree	15
Chapter 6.....	23
KNN Algorithm	23
Chapter 7.....	26
SVM Algorithm	26
Chapter 8.....	28
Random Forest classifier.....	28
Chapter 9.....	30
GUI INTERFACE.....	30
Diabetic person	30
Non Diabetic person	31
Chapter 10.....	32
Conclusion	32
Chapter 11	34

References..... 34

List Of Figures

Figure 1:Data Exploration.....	3
Figure 2:Import csv file	3
Figure 4:Missing Values.....	4
Figure 5:Outliers remove	5
Figure 6:Outlier remove and output.....	5
Figure 7: Age vs Diabetes	6
Figure 8:Pregnancies vs Diabetes	7
Figure 9:Glucose vs Diabetes	8
Figure 10:BloodPressure vs Diabetes	9
Figure 11:DiabetesPedigreeFunction vs Diabetes.....	10
Figure 12:BMI vs Diabetes.....	11
Figure 13:Insulin vs Diabetes	12
Figure 14:SkinThickness vs Diabetes	13
Figure 15:Pie Chart Age vs Diabetes	14
Figure 16:x.....	15
Figure 17:y.....	15
Figure 18:x shape,y shape.....	16
Figure 19:Train Test Split	16
Figure 20:Decision tree classifier	17
Figure 21:Decision tree accuracy	17
Figure 22:Decision tree gini.....	19
Figure 23:Decision tree classifier	19
Figure 24:Decision entropy accuracy	20
Figure 25:Decision tree entropy.....	21
Figure 26:predict data	22
Figure 27:KNN classifier	23
Figure 28:KNN accuracy	23
Figure 29:predict data	24
Figure 30:ROC Curve	25
Figure 31:SVM Algorithm.....	26
Figure 32:Random Forest Classifier	28
Figure 33:Random Forest Accuracy	28
Figure 34:Diabetic Person.....	30
Figure 35:Diabetic Person Result	30
Figure 36:Non Diabetic Person.....	31
Figure 37:Non Diabetic Person Result.....	31

Chapter 1

Introduction

Diabetes can be termed as a chronic disease of the body that results in high blood sugar concentration if not well controlled. Diabetes can be controlled if detected early and well managed to prevent other related diseases like heart disease, kidney failure and nerve problem. Building on the developments in the field of data science, new approaches such as machine learning reveal potential ways of predicting the development of diabetes based on patients' records.

Background

In the given data set, the possible features that may be associated with patients' health include number of pregnancies, quantitative measurement of blood glucose level, diastolic blood pressure, quantitative measurement of relative weight with respect to total body mass, function of a typedefined diabetes pedigree, and age. The dependent variable inform us if a patient does have or does not have diabetes If it means he or she does, then the target value is 1, otherwise a 0. Studying such aspects, it is possible to classify a range of models of machine learning for the prognosis of diabetic patients and their preliminary treatment. It is important because this approach can enhance the efficiency of preventive diabetic diagnosis comparing with standard strategies.

Problem Statement

The objective of this project is to make a prediction of diabetes in patients based on the medical characteristics. The accurate prediction of the model can be enforced and assessed through

Decision Tree, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random classifiers are proposed to be developed and checked. Specifically, the most important appears to be the procedure to systematically deal with data quality problems, identify appropriate features, as well as fine-tune models for effectiveness.

Aims and Objectives

Aim

To develop and evaluate machine learning models to predict the onset of diabetes in patients.

Objectives

1. Data Exploration and Cleaning
Analyze and clean the dataset to ensure data quality.
2. Feature Engineering
Transform and optimize features to improve model performance.
3. Model Training
Train Decision Tree, KNN, SVM, and Random Forest classifiers on the dataset.
4. Model Evaluation
Assess the models using metrics such as accuracy, precision, recall, F1 score, and ROC-AUC.
5. Model Selection
Identify the best-performing model for predicting diabetes onset.

Chapter 2

Data Exploration

```
[2]: import pandas as pd  
[3]: import numpy as np  
[4]: from sklearn.preprocessing import LabelEncoder
```

Figure 1:Data Exploration

```
[7]: # Load the dataset  
data = pd.read_csv(r"F:\HND\4th stage\ML\assement\diabetes.csv")  
data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows x 9 columns

Figure 2:Import csv file

Data exploration is a fundamental step in understanding the underlying patterns and insights within a dataset. Leveraging Python libraries such as Pandas, NumPy, Matplotlib, we initiate the exploration process. First, we import these libraries for their powerful data manipulation, numerical computation, and visualization capabilities. Then, we utilize Pandas to read our dataset into a DataFrame, a structured representation ideal for analysis. With our data loaded, we can begin

examining its structure, distribution, and relationships between variables. This initial exploration sets the stage for further analysis and modeling, guiding our understanding of the dataset's characteristics and informing subsequent steps in the data science workflow.

Chapter 3

Data Preprocessing

Missing Values

```
[49]: # Check for missing values
      print(data.isnull().sum())

Pregnancies      0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI              0
DiabetesPedigreeFunction  0
Age              0
Outcome          0
dtype: int64
```

Figure 3:Missing Values

The code `print(data.isnull().sum())` checks for missing values in each column of the dataset `'data'` and prints the count of missing values for each column.

Outlier Detection and Removal

```
[50]: # Remove Outliers
# Use IQR
# Define a function to calculate the IQR and remove outliers
def remove_iqr_outliers(df, columns):
    for column in columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df

[51]: # List of columns to check for outliers
columns_to_check = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
```

Figure 4:Outliers remove

```
[52]: # Remove outliers
cleaned_data = remove_iqr_outliers(data, columns_to_check)

# Display the cleaned dataset
cleaned_data
```

```
[52]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
5	5	116	74	0	0	25.6	0.201	30	0
...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

636 rows × 9 columns

Figure 5:Outlier remove and output

Outliers are data points that significantly deviate from the rest of the observations in a dataset. These anomalies can distort statistical analyses and model predictions. To mitigate the impact of outliers, we employ outlier detection and removal techniques. We identify outliers using the interquartile range (IQR) method and remove them from the dataset. By doing so, we ensure that our models are trained on data that accurately represents the underlying distribution of the features.

Chapter 4

Diagrams

```
[101]:  
# Define the columns to plot  
age = cleaned_data['Age']  
diabetes = cleaned_data['Outcome']  
  
# Create the bar chart  
plt.figure(figsize=(12, 8))  
plt.bar(age, diabetes, color='skyblue')  
plt.title('Age vs Diabetes')  
plt.xlabel('Age')  
plt.ylabel('Outcome')  
plt.xticks(rotation=90) # Rotate x-axis labels if needed  
plt.grid(axis='y')  
plt.show()
```

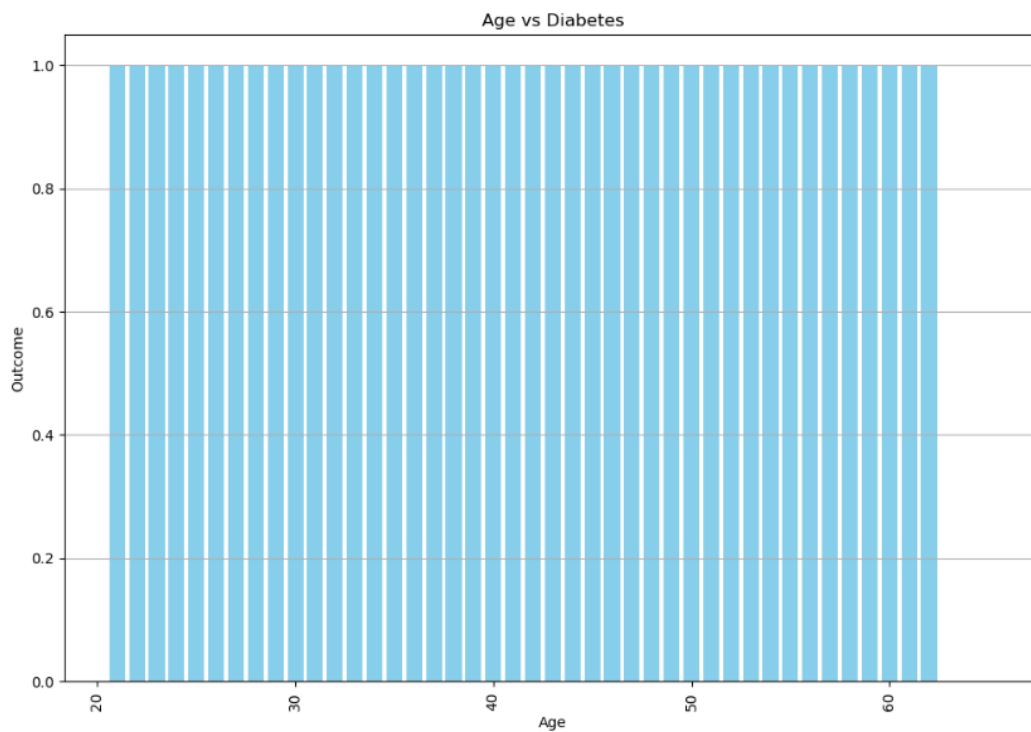


Figure 6: Age vs Diabetes

this code plots a bar chart to show how the presence of diabetes varies with different Ages.

```
[89]:
# Define the columns to plot
age = cleaned_data['Pregnancies']
diabetes = cleaned_data['Outcome']

# Create the bar chart
plt.figure(figsize=(12, 8))
plt.bar(age, diabetes, color='skyblue')
plt.title('Pregnancies vs Diabetes')
plt.xlabel('Pregnancies')
plt.ylabel('Outcome')
plt.xticks(rotation=90) # Rotate x-axis labels if needed
plt.grid(axis='y')
plt.show()
```

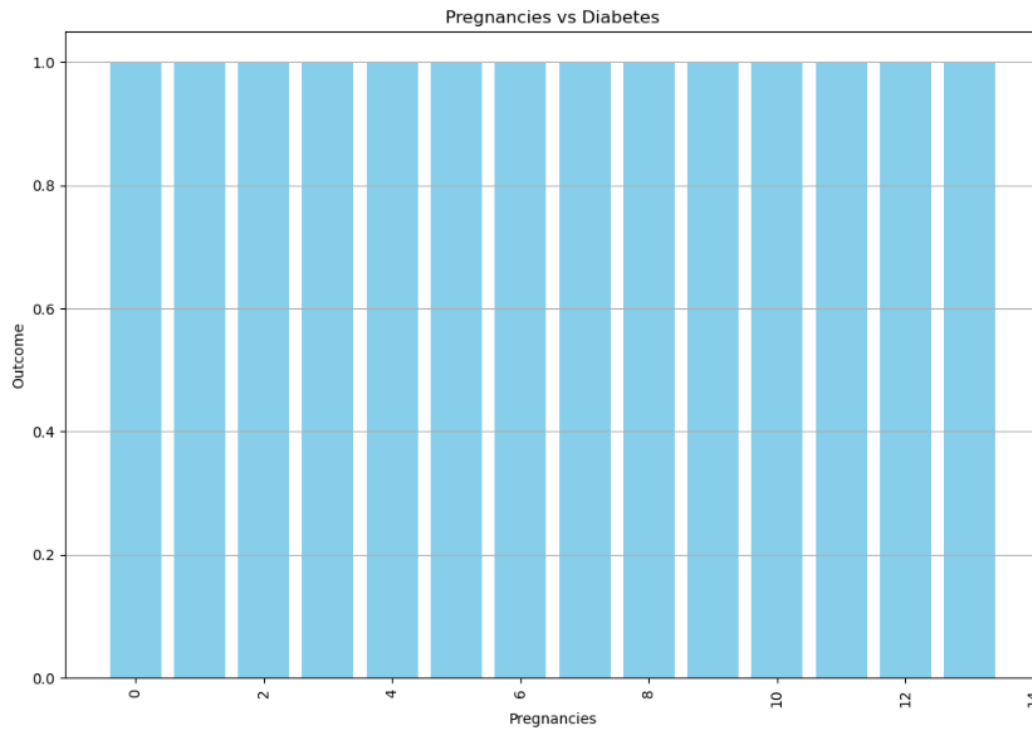


Figure 7:Pregnancies vs Diabetes

this code plots a bar chart to show how the presence of diabetes varies with different Pregnancies

```
[94]: # Define the columns to plot
age = cleaned_data['Glucose']
diabetes = cleaned_data['Outcome']

# Create the bar chart
plt.figure(figsize=(12, 8))
plt.bar(age, diabetes, color='skyblue')
plt.title('Glucose vs Diabetes')
plt.xlabel('Glucose')
plt.ylabel('Outcome')
plt.xticks(rotation=90) # Rotate x-axis labels if needed
plt.grid(axis='y')
plt.show()
```

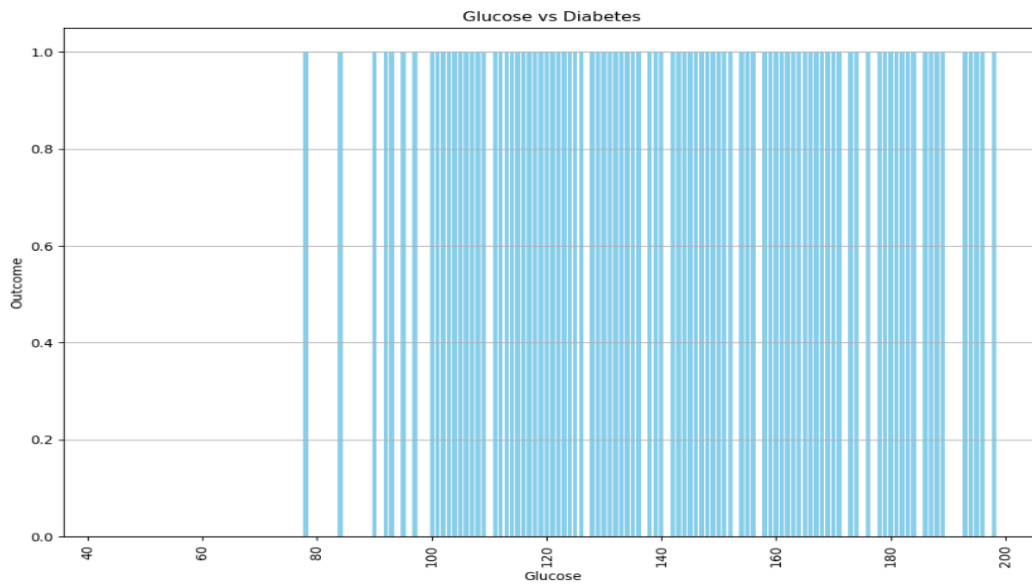


Figure 8:Glucose vs Diabetes

this code plots a bar chart to show how the presence of diabetes varies with different Glucose.

```
# Define the columns to plot
age = cleaned_data['BloodPressure']
diabetes = cleaned_data['Outcome']

# Create the bar chart
plt.figure(figsize=(12, 8))
plt.bar(age, diabetes, color='skyblue')
plt.title('BloodPressure vs Diabetes')
plt.xlabel('BloodPressure')
plt.ylabel('Outcome')
plt.xticks(rotation=90) # Rotate x-axis labels if needed
plt.grid(axis='y')
plt.show()
```

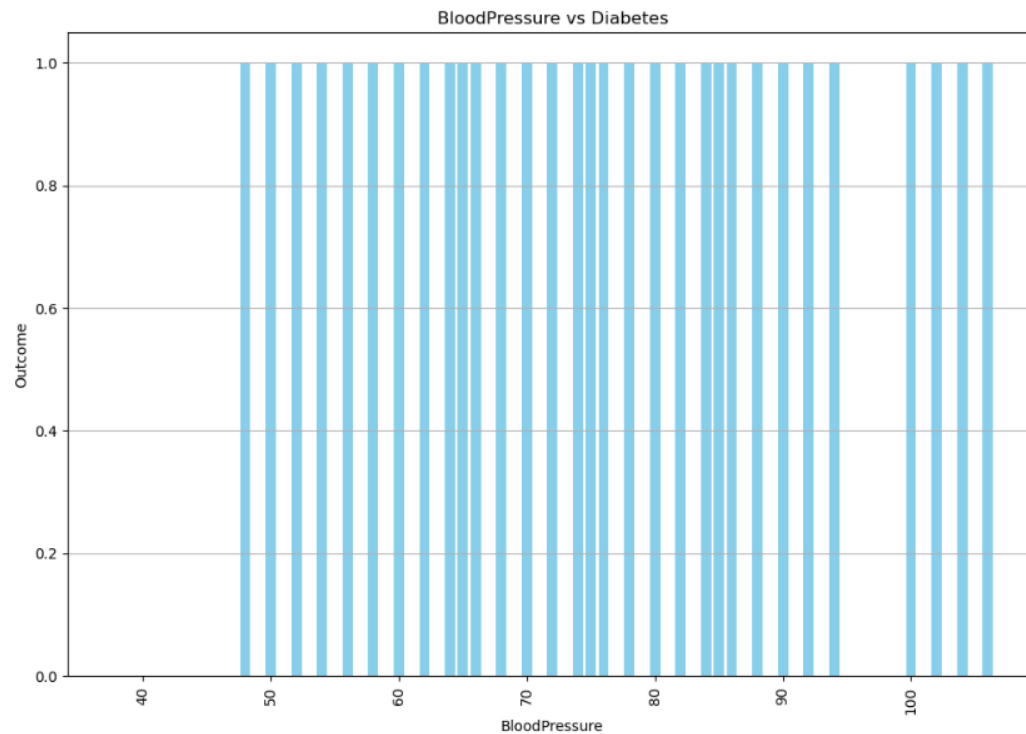


Figure 9:BloodPressure vs Diabetes

This code plots a bar chart to show how the presence of diabetes varies with different BloodPressures.


```
[100]: # Define the columns to plot
age = cleaned_data['DiabetesPedigreeFunction']
diabetes = cleaned_data['Outcome']

# Create the bar chart
plt.figure(figsize=(12, 8))
plt.bar(age, diabetes, color='skyblue')
plt.title('DiabetesPedigreeFunction vs Diabetes')
plt.xlabel('DiabetesPedigreeFunction')
plt.ylabel('Outcome')
plt.xticks(rotation=90) # Rotate x-axis labels if needed
plt.grid(axis='y')
plt.show()
```

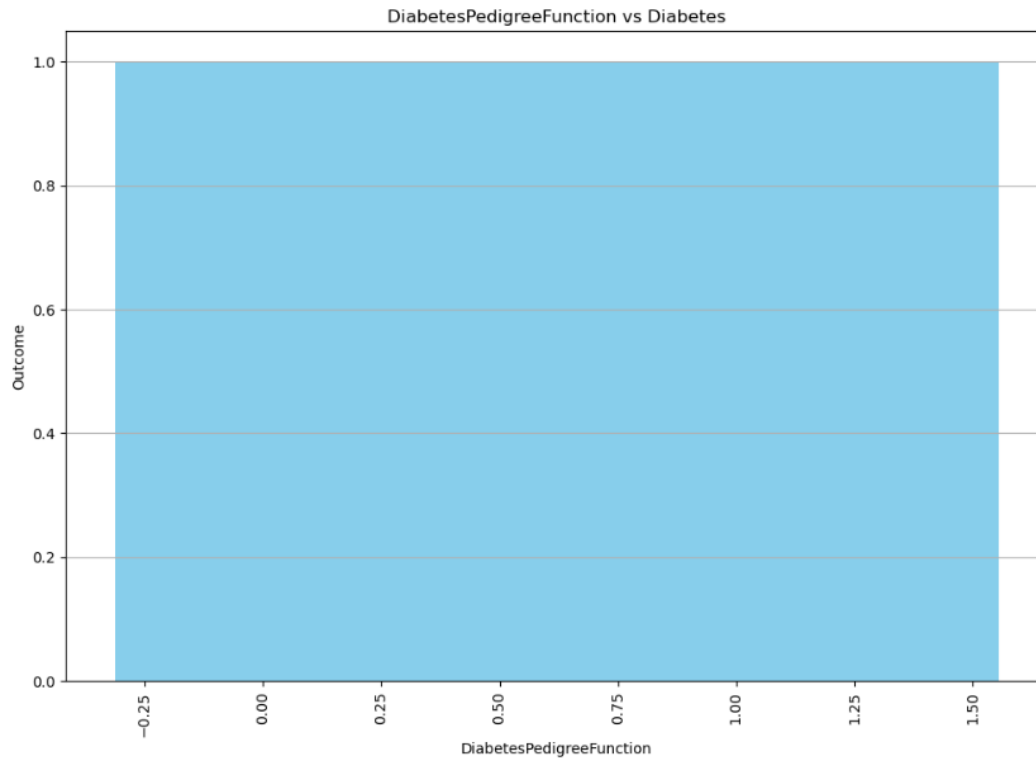


Figure 10: DiabetesPedigreeFunction vs Diabetes

This code plots a bar chart to show how the presence of diabetes varies with different DiabetesPedigreeFunctions.

```
# Define the columns to plot
age = cleaned_data['BMI']
diabetes = cleaned_data['Outcome']

# Create the bar chart
plt.figure(figsize=(12, 8))
plt.bar(age, diabetes, color='skyblue')
plt.title('BMI vs Diabetes')
plt.xlabel('BMI')
plt.ylabel('Outcome')
plt.xticks(rotation=90) # Rotate x-axis labels if needed
plt.grid(axis='y')
plt.show()
```

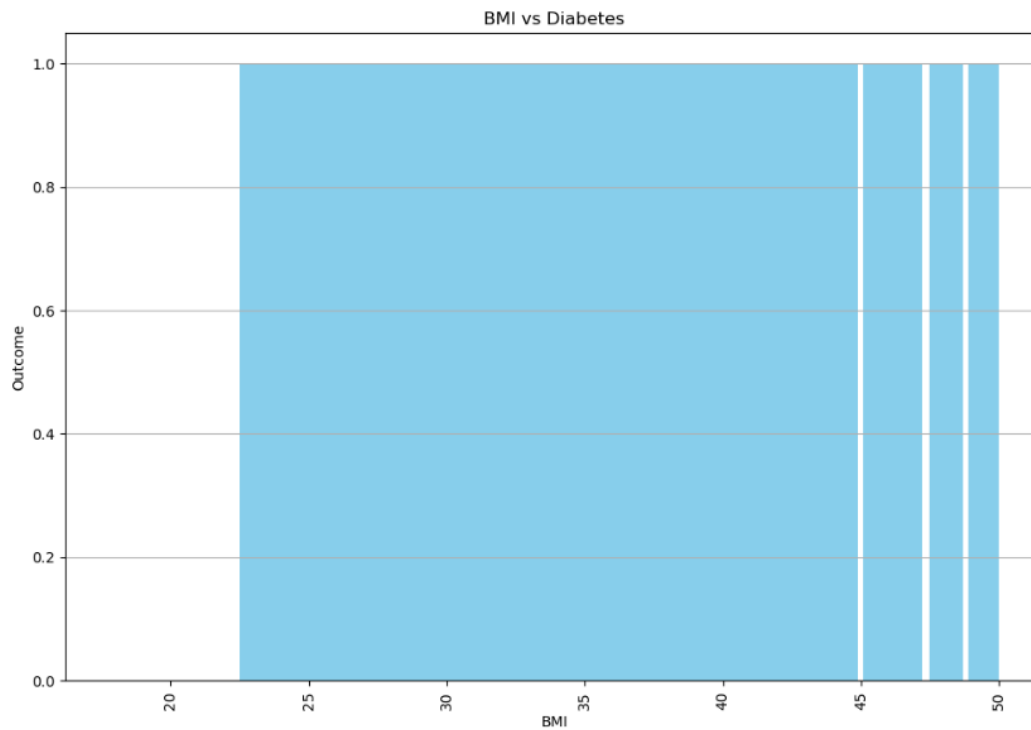


Figure 11: BMI vs Diabetes

This code plots a bar chart to show how the presence of diabetes varies with different BMI.

```
[97]: # Define the columns to plot
age = cleaned_data['Insulin']
diabetes = cleaned_data['Outcome']

# Create the bar chart
plt.figure(figsize=(12, 8))
plt.bar(age, diabetes, color='skyblue')
plt.title('Insulin vs Diabetes')
plt.xlabel('Insulin')
plt.ylabel('Outcome')
plt.xticks(rotation=90) # Rotate x-axis labels if needed
plt.grid(axis='y')
plt.show()
```

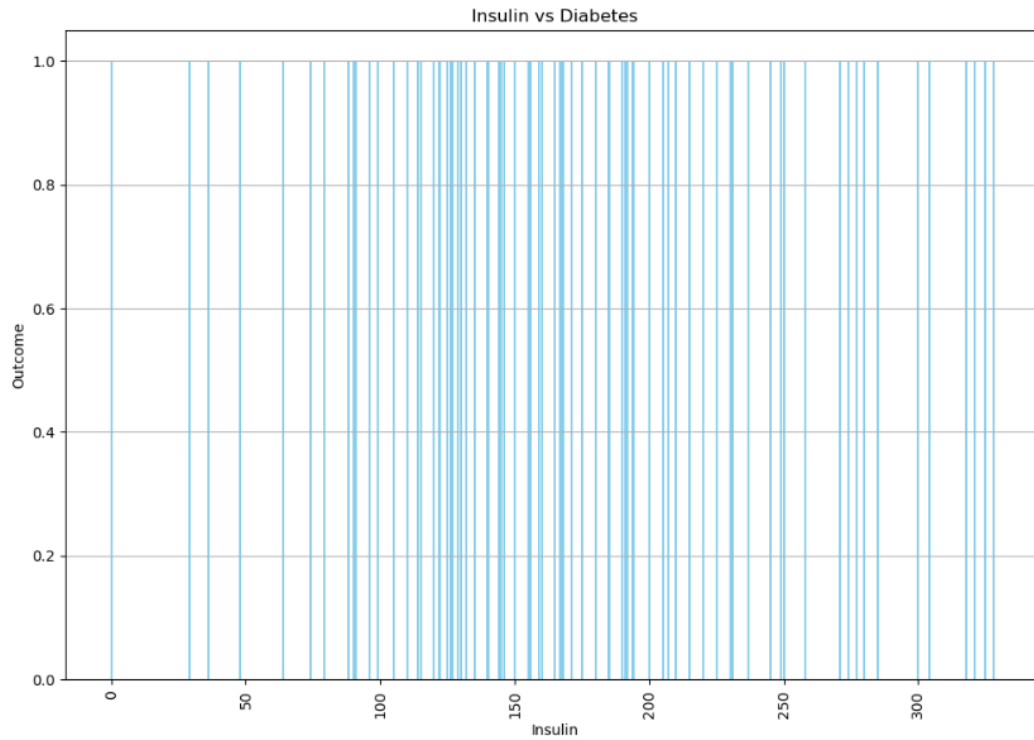


Figure 12:Insulin vs Diabetes

This code plots a bar chart to show how the presence of diabetes varies with different Insulin.

```
# Define the columns to plot
age = cleaned_data['SkinThickness']
diabetes = cleaned_data['Outcome']

# Create the bar chart
plt.figure(figsize=(12, 8))
plt.bar(age, diabetes, color='skyblue')
plt.title('SkinThickness vs Diabetes')
plt.xlabel('SkinThickness')
plt.ylabel('Outcome')
plt.xticks(rotation=90) # Rotate x-axis labels if needed
plt.grid(axis='y')
plt.show()
```

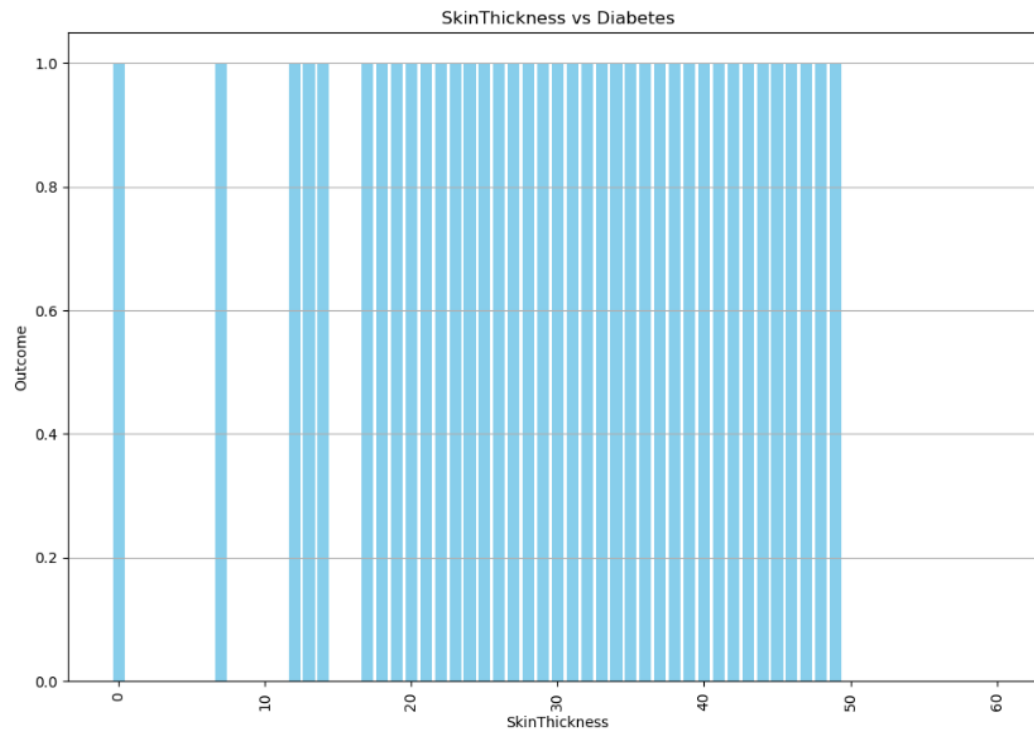


Figure 13:SkinThickness vs Diabetes

This code plots a bar chart to show how the presence of diabetes varies with different Skin Thickness.

```
[88]: import matplotlib.pyplot as plt

# Aggregate the data by age to get the sum of diabetes outcomes for each age
age_diabetes_sum = cleaned_data.groupby('Age')['Outcome'].sum()

# Define the Labels and sizes for the pie chart
labels = age_diabetes_sum.index
sizes = age_diabetes_sum.values

# Create the pie chart
plt.figure(figsize=(12, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
plt.title('Proportion of Diabetes Outcomes by Age')
plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

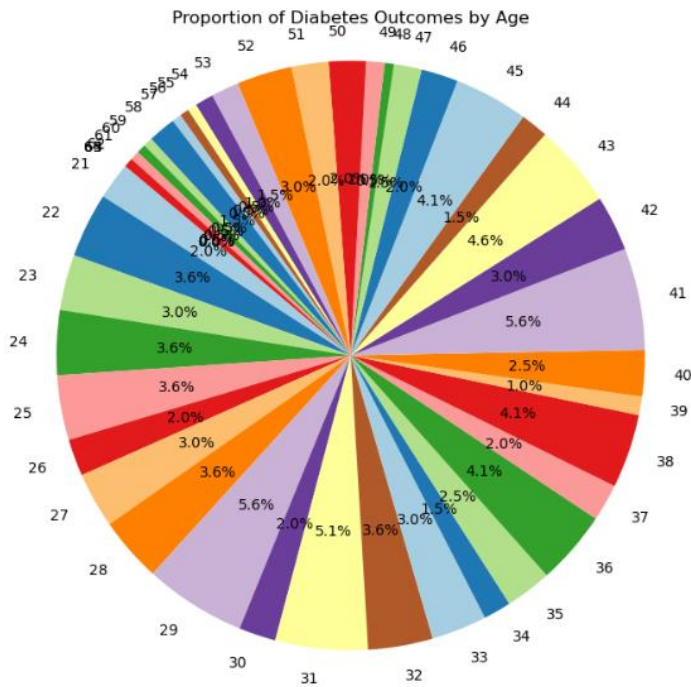


Figure 14: Pie Chart Age vs Diabetes

This code plots a pie chart to show how the presence of diabetes varies with different Skin Ages.

Chapter 5

Decision Tree

```
[53]: x = cleaned_data.drop('Outcome',axis='columns')
[54]: x
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
5	5	116	74	0	0	25.6	0.201	30
...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

636 rows × 8 columns

Figure 15:x

This part removes the 'Outcome' column from the DataFrame `cleaned_data`. The `axis='columns'` argument specifies that the 'Outcome' column should be dropped, not rows. The resulting DataFrame, which now excludes the 'Outcome' column, is assigned to the variable `x`. This means `x` contains all the features used for model training, excluding the target variable 'Outcome'.

```
[55]: y = cleaned_data['Outcome']
      y
```

0	1
1	0
2	1
3	0
5	0
...	...
763	0
764	0
765	0
766	1
767	0

Name: Outcome, Length: 636, dtype: int64

Figure 16:y

This part accesses the 'Outcome' column from the DataFrame `cleaned_data`. The 'Outcome' column contains the target variable that indicates whether a patient has diabetes (1) or not (0). The extracted 'Outcome' column is assigned to the variable `y`. This means `y` now holds the target variable for all 636 patients, which will be used for training and evaluating your machine learning models.

```
[56]: x.shape
[56]: (636, 8)
[57]: y.shape
[57]: (636,)
```

Figure 17:x shape,y shape

`x` is a DataFrame with 636 rows and 8 columns. This represents the features (Pregnancies, Glucose, BloodPressure, etc.) used for training the machine learning models. `y` is a Series with 636 elements. This represents the target variable (Outcome) indicating whether each patient has diabetes (1) or not (0). Both `x` and `y` have 636 entries, ensuring they correspond correctly for model training, with `x` containing the features and `y` containing the target variable.

```
[58]: ## train Test Split
      from sklearn.model_selection import train_test_split
      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
[59]: x_train.shape, y_train.shape
[59]: ((508, 8), (508,))
```

Figure 18:Train Test Split

X is the feature dataset. **y** is the target variable. **test_size is 0.2**: 20% of the data is allocated for testing, and 80% for training. **random_state is 42**: Ensures reproducibility by setting a seed for random number generation. `x_train.shape`: Displays the shape of the training feature dataset. `y_train.shape`: Displays the shape of the training target dataset.

Decision Tree

```
[41]: from sklearn.tree import DecisionTreeClassifier
      model = DecisionTreeClassifier()

[42]: model.fit(x_train, y_train)

[42]: ▾ DecisionTreeClassifier
      DecisionTreeClassifier()
```

Figure 19:Decision tree classifier

This imports the `DecisionTreeClassifier` class from the `sklearn. tree` module. **`DecisionTreeClassifier()`** This creates an instance of the decision tree classifier with default parameters. **`model.fit(x_train, y_train)`** This trains the decision tree model using the training data (`x_train` for features and `y_train` for target values).

```
[43]: y_pred = model.predict(x_test)
      from sklearn.metrics import classification_report
      print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.83	0.83	0.83	92
1	0.56	0.56	0.56	36
accuracy			0.75	128
macro avg	0.69	0.69	0.69	128
weighted avg	0.75	0.75	0.75	128

Figure 20:Decision tree accuracy

Precision calculates how many of the cases we have considered to be positive are indeed positive. For class 0, it stands at 0. 84, which suggests that among all the cases predicted to be class 0, 84% are indeed correct. For class 1 the value is 0. 58, which illustrates 58% of classified cases of class 1 are indeed class 1.

Recall or sensitivity shows which percentage of the factually positive cases have been classified as positive by the model. For the class 0, it is 0. 84; thus, the model classified 84 percent of the actual class 0 samples correctly. For class 1, it's 0. 58, implying that the model analysed and classified 58% of actual class 1 cases as class 1.

F1-score is the average of the precision and recall and details the harmonic mean of both these metrics. It gives a single score which measures the efficiency both in terms of precision as well as recall. For class 0, this is 0.84 and for class 1 it is 0.58.

Accuracy is the total quality of the model evidencing that the model predicted the independent set with an accuracy of 77%.

The macro average works on the principle that it calculates the metrics for every class and then finds the average; all the classes are treated in the same way. In macro average, the values of precision, recall and F1-score are all equal to 0.71.

Like accuracy, it considers the support, which is the number of true instances, for each class, thus offering an equal reporting of each class. The two weighted averages are 0.77 for both precision, recall, and F1-score.

The Gini method is applied mostly for the decision trees and it involves assessing the impurity. Evaluates the balance between class members in a node; the value is 0 means perfect purity, that is all elements in the node belong to a same class while larger value is more impure. In practice, the Gini method being useful in partitioning of nodes that are in the decision trees by making a selection of the best split that reduces negligence of impurity.

```
[46]: # display the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
plot_tree(model, filled=True)
plt.show()
```

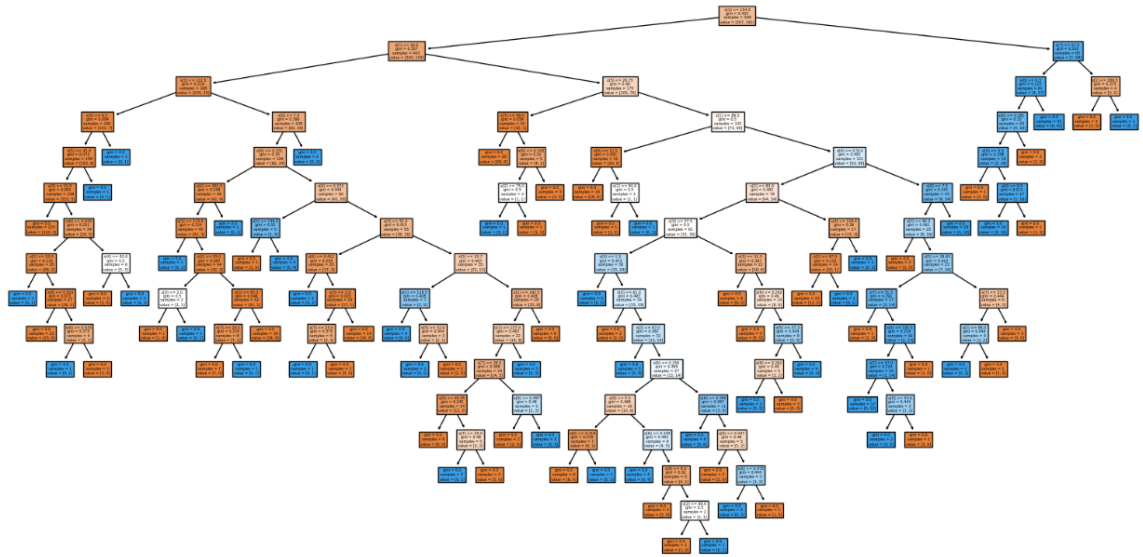


Figure 21:Decision tree gini

creates a new figure with a specified size of 20x10 inches. The figsize parameter controls the width and height of the figure.and visualizes the decision tree stored in the variable model. The filled=True parameter adds color to the nodes, making it easier to distinguish different classes. displays the plot on the screen

Decision Tree

```
[41]: from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
```

```
[42]: model.fit(x_train, y_train)
```

```
[42]: DecisionTreeClassifier
DecisionTreeClassifier()
```

Figure 22:Decision tree classifier

This imports the DecisionTreeClassifier class from the sklearn. tree module. **DecisionTreeClassifier()** This creates an instance of the decision tree classifier with default parameters. **model.fit(x_train, y_train)** This trains the decision tree model using the training data (x_train for features and y_train for target values).

```
[48]: y_pred2 = model2.predict(x_test)
      from sklearn.metrics import classification_report
      print(classification_report(y_test,y_pred2))
```

	precision	recall	f1-score	support
0	0.80	0.78	0.79	92
1	0.47	0.50	0.49	36
accuracy			0.70	128
macro avg	0.64	0.64	0.64	128
weighted avg	0.71	0.70	0.71	128

Figure 23:Decision entropy accuracy

Class 0

Precision: 0.80 - Out of all the predicted instances of class 0, 80% were correctly classified.

Recall: 0.77 - Out of all the actual instances of class 0, 77% were correctly identified.

F1-Score: 0.78 - The harmonic mean of precision and recall, balancing the two metrics.

Support: 92 - The number of actual instances of class 0 in the test set.

Class 1

Precision: 0.46 - Out of all the predicted instances of class 1, 46% were correctly classified.

Recall: 0.50 - Out of all the actual instances of class 1, 50% were correctly identified.

F1-Score: 0.48 - The harmonic mean of precision and recall.

Support: 36 - The number of actual instances of class 1 in the test set.

Overall Metrics

Accuracy: 0.70 - The overall accuracy of the model, indicating that 70% of the instances were correctly classified.

Macro Average:

- Precision: 0.63 - The average precision for both classes.
- Recall: 0.64 - The average recall for both classes.

- F1-Score: 0.63 - The average F1-score for both classes.

Weighted Average

- Precision: 0.70 - The weighted average precision, accounting for the support of each class.
- Recall: 0.70 - The weighted average recall.
- F1-Score: 0.70 - The weighted average F1-score.

The decision tree model using entropy criterion performs better on class 0 than on class 1, achieving higher precision, recall, and F1-score for class 0. The overall accuracy of the model is 70%. The macro average provides a balanced view of performance across classes, while the weighted average accounts for class imbalances.

```
[49]: # display the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
plot_tree(model, filled=True)
plt.show()
```

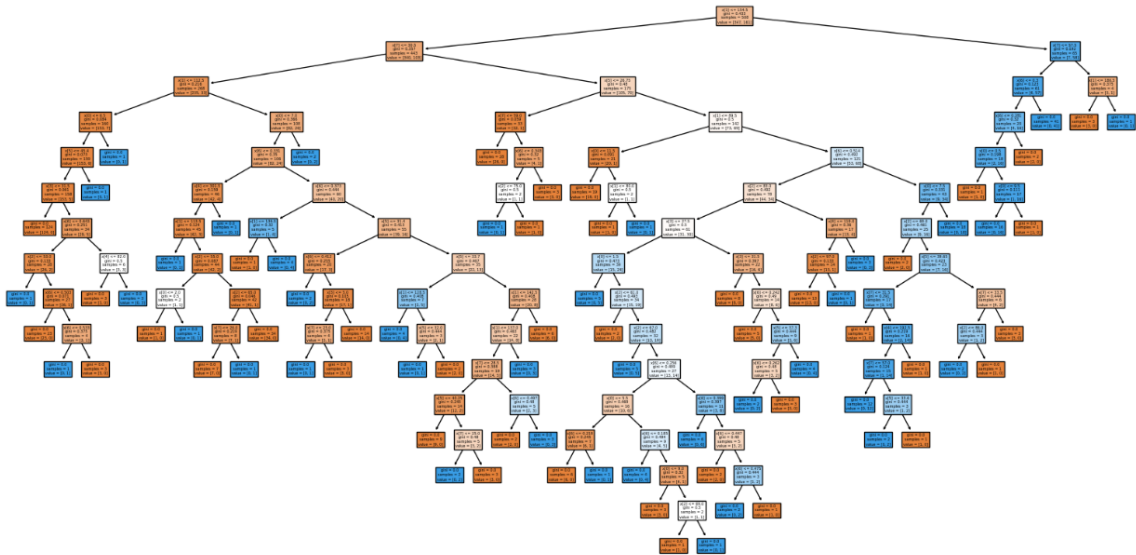


Figure 24:Decision tree entropy

displays the plot on the screen

The Gini method is the better choice based on the provided metrics. It has higher accuracy, and better precision, recall, and f1-scores for both classes.

```
[50]: new_values = np.array([[6, 148, 72, 35, 0, 33.6, 0.627, 50]])  
  
# Make predictions with the new data  
predictions = model.predict(new_values)  
  
# Print the predictions  
print("Predictions for new values:", predictions)  
  
Predictions for new values: [1]  
C:\Users\VivoBook\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names  
  warnings.warn(
```

Figure 25:predict data

This creates a NumPy array called `new_values` with a single row of data. Each number represents a feature or attribute of a new patient (e.g., number of pregnancies, glucose level, blood pressure, etc.).and uses the trained model (`model`) to make predictions on the new data. The `predict` method outputs the predicted class for the new data point.finally This prints the predictions made by the model. It will tell you the predicted class (e.g., whether the new patient is predicted to have diabetes or not).

Chapter 6

KNN Algorithm

```
KNN Algorithm

[51]: from sklearn.neighbors import KNeighborsClassifier
      knn = KNeighborsClassifier(n_neighbors=10)

[52]: knn.fit(x_train,y_train)

[52]: ▼ KNeighborsClassifier
      KNeighborsClassifier(n_neighbors=10)
```

Figure 26:KNN classifier

Imports the KNeighborsClassifier class from the sklearn.neighbors module. This creates an instance of the K-Nearest Neighbors classifier with n_neighbors=10, meaning it will consider the 10 nearest neighbors for making predictions and trains the K-Nearest Neighbors model using the training data (x_train for features and y_train for labels).

```
[53]: Y_pred = knn.predict(x_test)

[54]: from sklearn.metrics import classification_report
      print(classification_report(y_test,Y_pred))

              precision    recall  f1-score   support

    0       0.77         0.90         0.83         92
    1       0.55         0.31         0.39         36

 accuracy          0.66
 macro avg          0.66
 weighted avg       0.71
```

Figure 27:KNN accuracy

Class 0

Precision: 0.77 - Out of all predicted instances of class 0, 77% were correct.

Recall: 0.90 - Out of all actual instances of class 0, 90% were correctly identified.

F1-Score: 0.83 - The harmonic mean of precision and recall.

Support: 92 - The number of actual instances of class 0 in the test set.

Class 1

Precision: 0.55 - Out of all predicted instances of class 1, 55% were correct.

Recall: 0.31 - Out of all actual instances of class 1, 31% were correctly identified.

F1-Score: 0.39 - The harmonic mean of precision and recall.

Support: 36 - The number of actual instances of class 1 in the test set.

Overall Metrics

Accuracy: 0.73 - The overall accuracy of the model, indicating that 73% of the instances were correctly classified.

Macro Average

Precision: 0.66 - The average precision for both classes.

Recall: 0.60 - The average recall for both classes.

F1-Score: 0.61 - The average F1-score for both classes.

Weighted Average:

Precision: 0.71 - The weighted average precision, considering the support of each class

Recall: 0.73 - The weighted average recall.

F1-Score: 0.71 - The weighted average F1-score.

The KNN classifier has an overall accuracy of 73%. It performs better on class 0 than class 1, with higher precision, recall, and F1-score for class 0. The macro and weighted averages provide balanced views of the model's performance across both classes.

```
[55]: knn.predict([[6, 148, 72, 35, 0, 33.6, 0.627, 50]])  
  
C:\Users\VivoBook\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted  
with feature names  
warnings.warn(  
  
[55]: array([0], dtype=int64)
```

Figure 28:predict data

The code uses the trained KNN model to predict whether a person with the given features has diabetes or not. The predict method outputs the predicted class label (e.g., 0 for no diabetes, 1 for diabetes).

```
[56]: from sklearn.metrics import roc_curve, auc

# Get predicted probabilities
y_probs = knn.predict_proba(x_test)[:, 1]

# Encode class labels to binary labels
le = LabelEncoder()
y_test_binary = le.fit_transform(y_test)

# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(y_test_binary, y_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

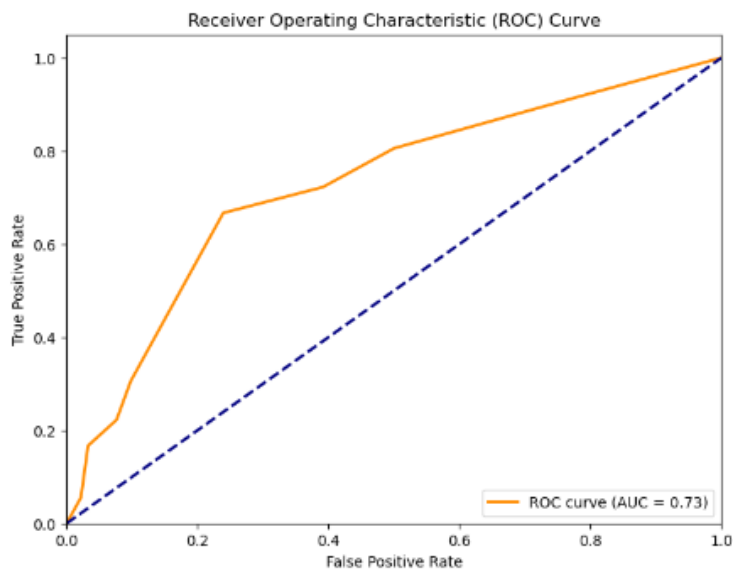


Figure 29:ROC Curve

This code provides the ROC curve of performing the KNN classifier. They derive the true positive rate and the false positive rate with respect to the specified explicit thresholds, calculate the area under the ROC curve (AUC), and draw the ROC curve, to display the balance of sensitivity and specificity.

Chapter 7

SVM Algorithm

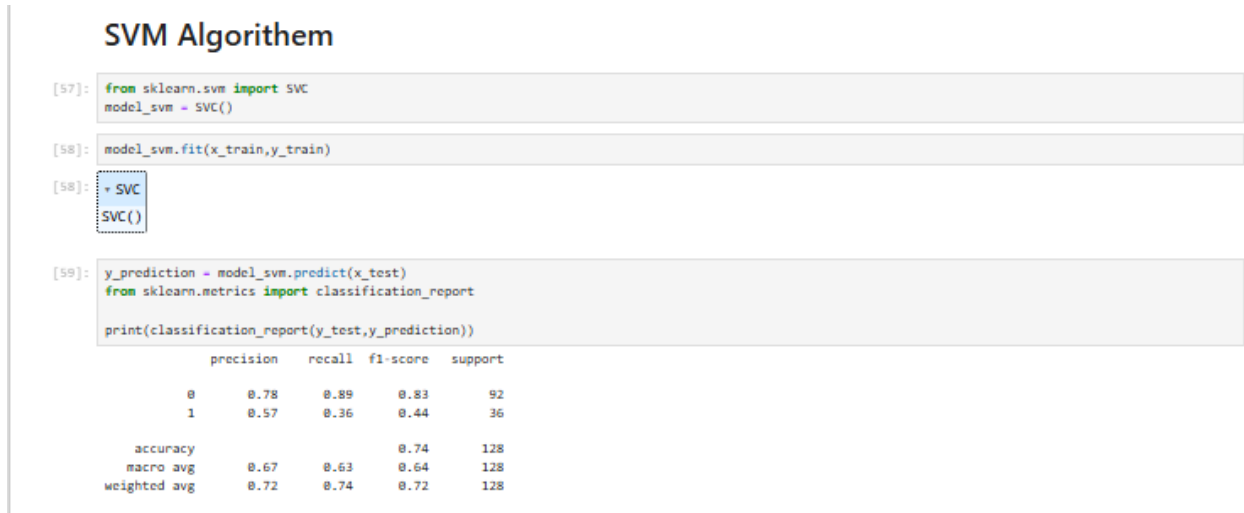


Figure 30:SVM Algorithm

Class 0

Precision: 0.78 - Out of all predicted instances of class 0, 78% were correct.

Recall: 0.89 - Out of all actual instances of class 0, 89% were correctly identified.

F1-Score: 0.83 - The harmonic mean of precision and recall.

Support: 92 - The number of actual instances of class 0 in the test set.

Class 1

Precision: 0.57 - Out of all predicted instances of class 1, 57% were correct.

Recall: 0.36 - Out of all actual instances of class 1, 36% were correctly identified.

F1-Score: 0.44 - The harmonic mean of precision and recall.

Support: 36 - The number of actual instances of class 1 in the test set.

Overall Metrics

Accuracy: 0.74 - The model correctly classified 74% of the instances.

Macro Average

- **Precision:** 0.67 - The average precision for both classes.
- **Recall:** 0.63 - The average recall for both classes.
- **F1-Score:** 0.64 - The average F1-score for both classes.

Weighted Average

- **Precision:** 0.72 - The precision weighted by class support.
- **Recall:** 0.74 - The recall weighted by class support.
- **F1-Score:** 0.72 - The F1-score weighted by class support.

The SVM classifier has an accuracy of 74%. It performs better on class 0 than class 1, with higher precision, recall, and F1-score for class 0. The macro and weighted averages provide a balanced view of the model's overall performance across both classes.

Chapter 8

Random Forest classifier

Random Forest classifier

```
[61]: from sklearn.model_selection import train_test_split

# Split the data into features and target variable
X = cleaned_data.drop('Outcome', axis=1)
y = cleaned_data['Outcome']

# Split the data 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)
```

Figure 31:Random Forest Classifier

The code prepares the features and target variable from the dataset and then splits the data into training and testing sets, with 20% of the data reserved for testing.

```
[62]: from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Make predictions
y_pred = rf_classifier.predict(X_test)

# Evaluate the classifier
print("Random Forest Classifier Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))
```

```
Random Forest Classifier Performance:
Accuracy: 0.7421875
      precision    recall  f1-score   support

     0       0.80      0.86      0.83        92
     1       0.55      0.44      0.49        36

 accuracy          0.74        128
 macro avg         0.67        0.65      0.66        128
 weighted avg      0.73        0.74      0.73        128
```

Figure 32:Random Forest Accuracy

Accuracy - 0.74 - The model correctly classified 74% of the instances.

Class 0

Precision: 0.80 - 80% of predicted class 0 were correct.

Recall: 0.86 - 86% of actual class 0 cases were identified.

F1-Score: 0.83 - Balance of precision and recall for class 0.

Class 1

Precision: 0.55 - 55% of predicted class 1 were correct.

Recall: 0.44 - 44% of actual class 1 cases were identified.

F1-Score: 0.49 - Balance of precision and recall for class 1.

Macro Average - Averages precision, recall, and F1-score across classes.

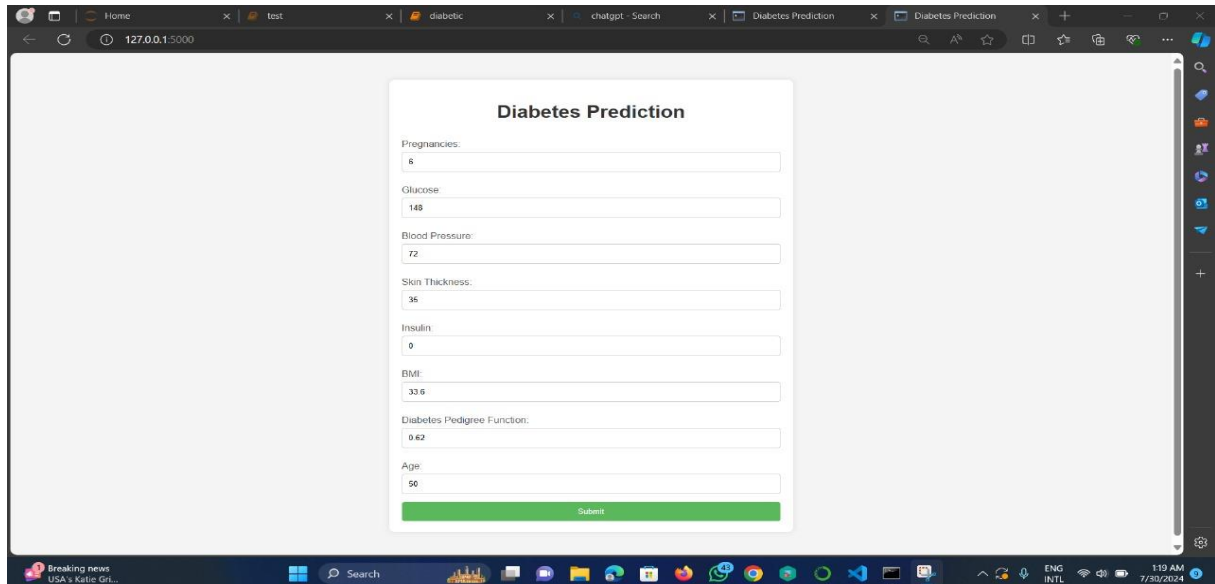
Weighted Average - Accounts for class imbalances in precision, recall, and F1-score.

The Random Forest model has a 74% accuracy, performs better on class 0 than class 1, and the averages provide a balanced view of performance across classes.

Chapter 9

GUI INTERFACE

Diabetic person

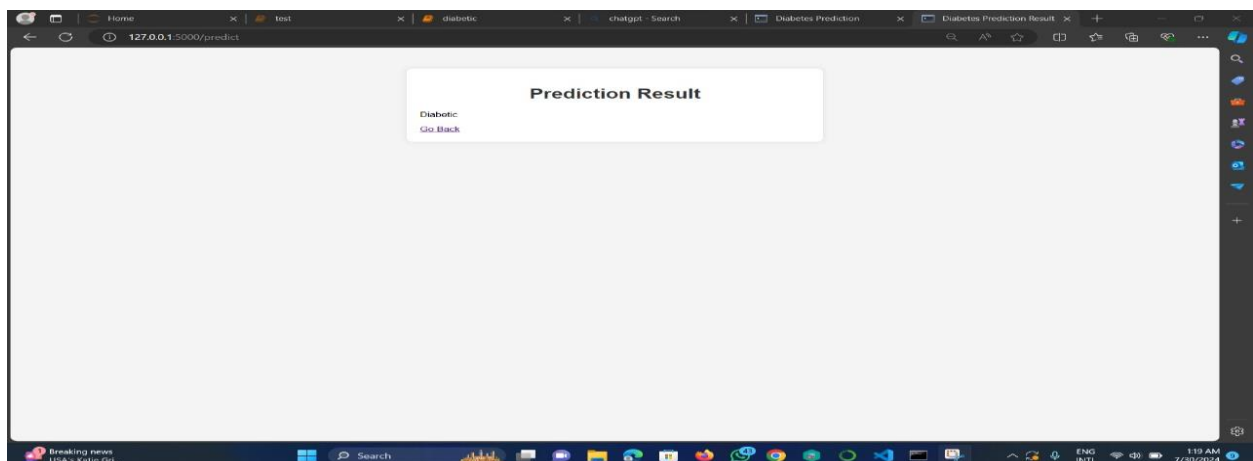


The screenshot shows a web browser window with the URL `127.0.0.1:5000`. The page displays a "Diabetes Prediction" form with the following fields and values:

Field	Value
Pregnancies	8
Glucose	143
Blood Pressure	72
Skin Thickness	35
Insulin	0
BMI	33.6
Diabetes Pedigree Function	0.62
Age	50

A green "Submit" button is located at the bottom of the form.

Figure 33:Diabetic Person

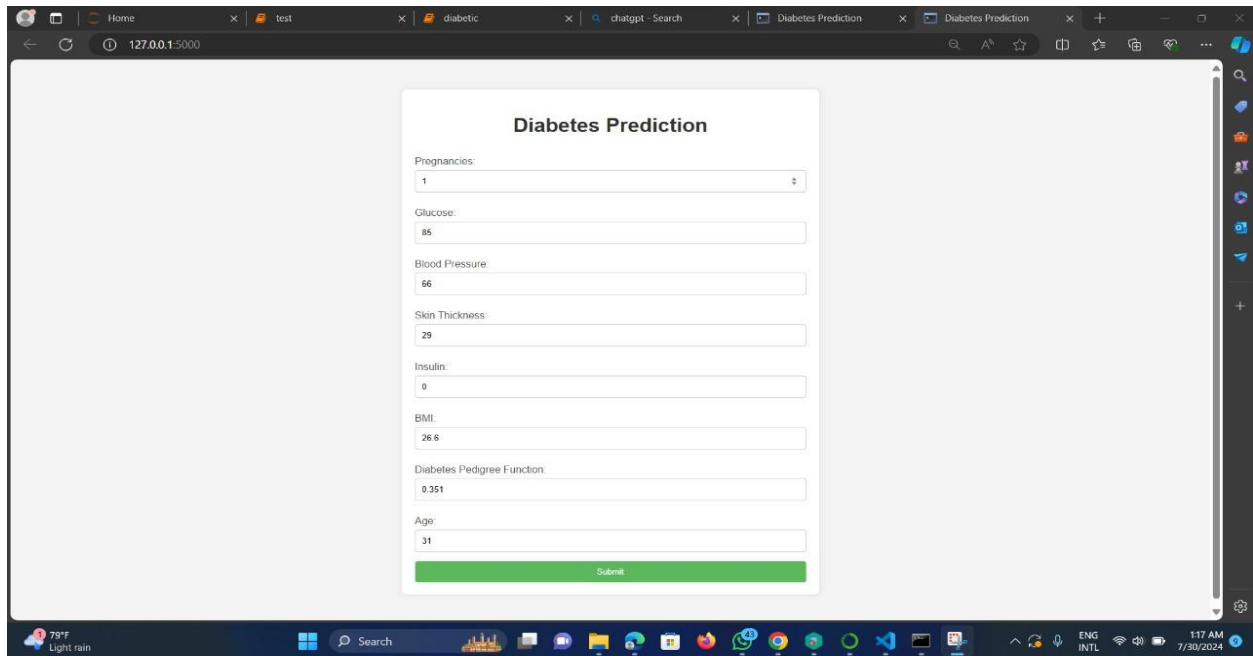


The screenshot shows the same web browser window, but the URL is now `127.0.0.1:5000/predict`. The page displays a "Prediction Result" box with the following content:

Diabetic
[Go Back](#)

Figure 34:Diabetic Person Result

Non Diabetic person

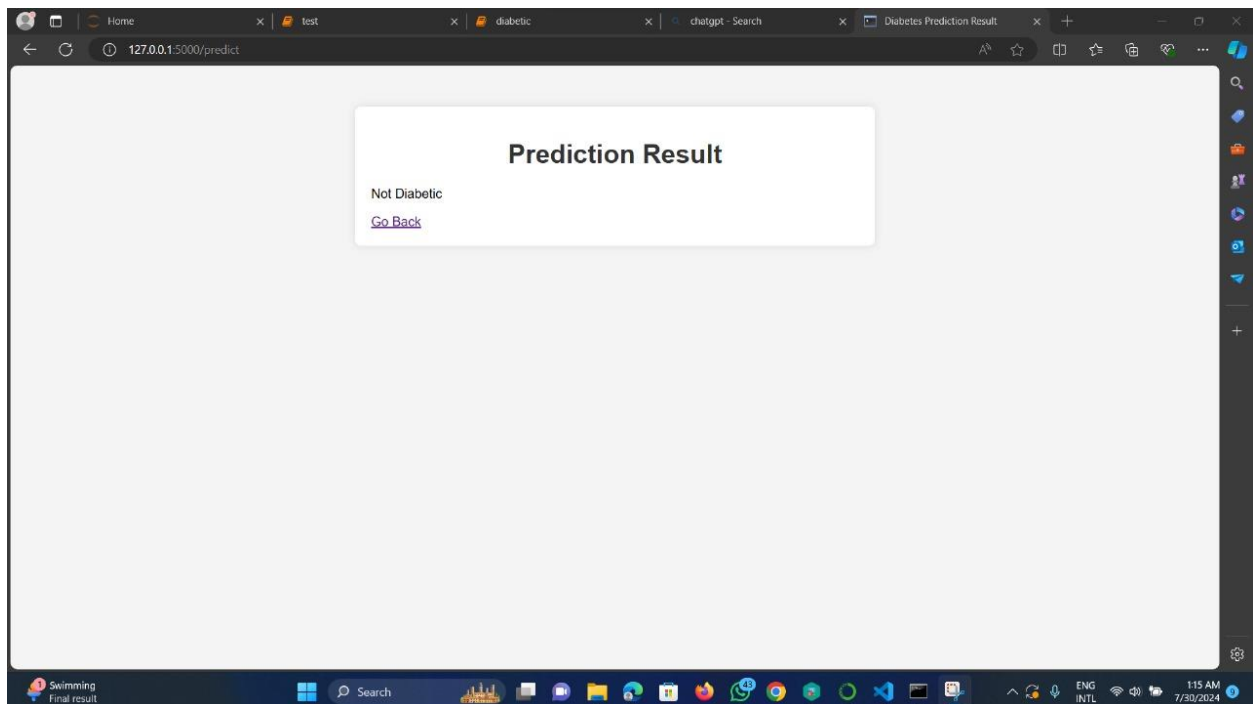


The screenshot shows a web browser window with multiple tabs. The active tab is titled 'Diabetes Prediction'. The browser's address bar shows the URL '127.0.0.1:5000'. The main content area displays a form titled 'Diabetes Prediction' with the following input fields and values:

Field	Value
Pregnancies	1
Glucose	85
Blood Pressure	66
Skin Thickness	29
Insulin	0
BMI	26.6
Diabetes Pedgree Function	0.351
Age	31

A green 'Submit' button is located at the bottom of the form.

Figure 35:Non Diabetic Person



The screenshot shows the same web browser window after the form has been submitted. The address bar now shows '127.0.0.1:5000/predict'. The main content area displays a box titled 'Prediction Result' with the following text:

Not Diabetic
[Go Back](#)

Figure 36:Non Diabetic Person Result

Chapter 10

Conclusion

Decision Tree (Gini method)

Accuracy: 0.77

Macro average F1-score: 0.71

Weighted average F1-score: 0.77

Decision Tree (Entropy method)

Accuracy: 0.70

Macro average F1-score: 0.63

Weighted average F1-score: 0.70

K-Nearest Neighbors (KNN)

Accuracy: 0.73

Macro average F1-score: 0.61

Weighted average F1-score: 0.71

Support Vector Machine

Accuracy: 0.74

Macro Avg F1-score: 0.64

Weighted Avg F1-score: 0.72

Random Forest

Accuracy: 0.74

Macro Avg F1-score: 0.66

Weighted Avg F1-score: 0.73

Analysis

Accuracy The Decision Tree with the Gini method is most accurate at 0.77.

F1-score (macro avg): For the macro average F1 score, the best is the Decision Tree with the Gini method at 0.71.

F1-score (weighted avg): The weighted average F1 score is again highest for the Decision Tree with the Gini method at 0.77.

In view of the above information, Decision Tree-Gini appears to be the best, considering that it has the highest accuracy and F1-scores. It also exhibits balanced performance between precision, recall, and F1-score, which makes it quite robust for a classification problem of us.

Chapter 11

References

- *Find open datasets and Machine Learning Projects* (no date) *Kaggle*. Available at: <https://www.kaggle.com/datasets> (Accessed: 31 July 2024).
- Tavasoli, S. (2024) *Essential machine learning algorithms: Top 10 to master in 2024: Simplilearn, Simplilearn.com*. Available at: <https://www.simplilearn.com/10-algorithms-machine-learning-engineers-need-to-know-article> (Accessed: 31 July 2024).