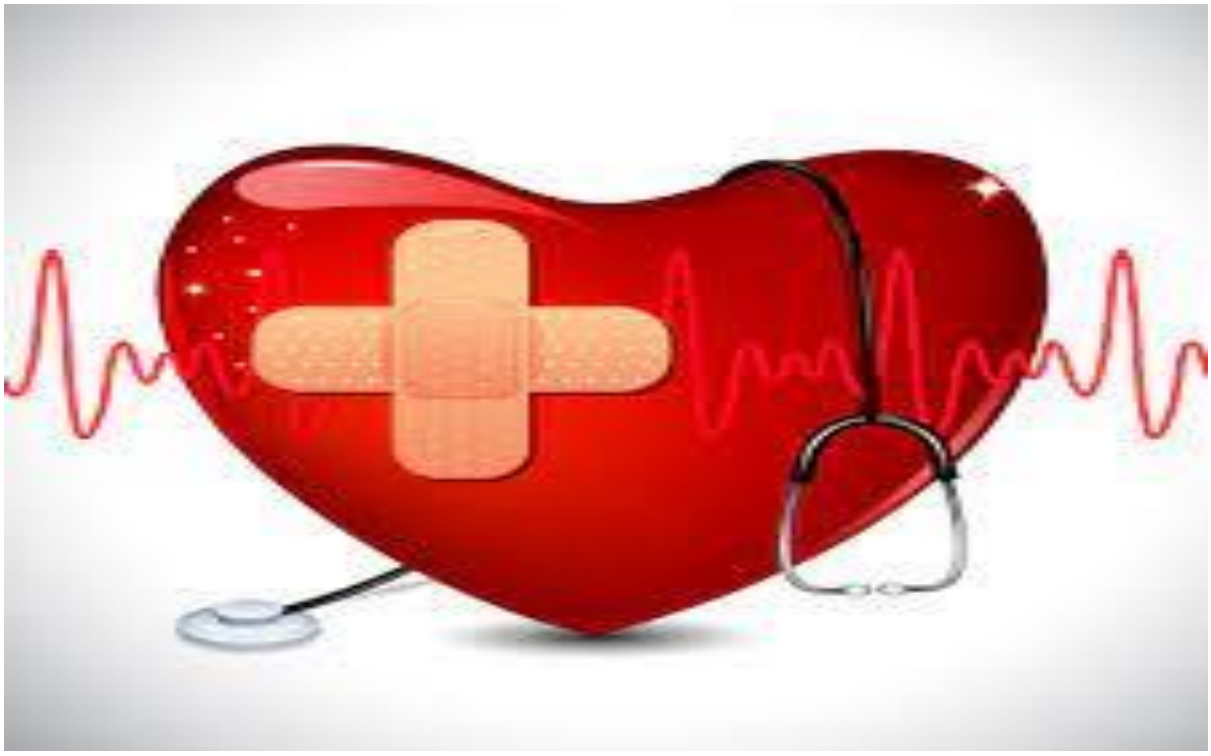
DIABETES PREDICTION USING MACHINE LEARNING CLASSIFICATION TECHNIQUES

An Internship project report submitted by

HUDHA K



EXPOSYS DATALABS ,BANGALORE

March 2024

CONTENTS

ABSTRACT

The dataset was preprocessed to handle missing values, normalize features, and address any outliers. Exploratory data analysis (EDA) techniques were applied to gain insights into the distribution and relationships between variables. Subsequently, multiple machine learning algorithms, including logistic regression, decision trees, random forests, and support vector machines, were trained and evaluated using cross-validation techniques. The performance of each model was assessed using metrics such as accuracy, precision, recall, and F1-score. The results indicate that the logistic regression model achieved the highest accuracy among the tested algorithms. This study demonstrates the potential of machine learning approaches in assisting healthcare professionals in early diagnosis and intervention for individuals at risk of diabetes. Further research and validation with larger datasets are recommended to improve the robustness and generalizability of the predictive model.

1.INTRODUCTION

1.1 INTRODUCTION:

Diabetes mellitus is a chronic metabolic disorder characterized by elevated blood sugar levels resulting from defects in insulin secretion, insulin action, or both. It poses a significant global health burden, affecting millions of individuals worldwide and leading to various complications such as cardiovascular diseases, kidney failure, and blindness if left untreated. Early detection and management of diabetes are crucial for preventing or delaying its complications and improving patients' quality of life.

In recent years, machine learning (ML) techniques have emerged as promising tools for predicting and diagnosing various medical conditions, including diabetes. ML models can analyze large datasets containing diverse patient information, such as demographic details, clinical measurements, and lifestyle factors, to identify patterns and trends associated with disease risk. By leveraging these insights, healthcare professionals can make more accurate and timely decisions, leading to better patient outcomes.

This project aims to develop an ML-based predictive model for diabetes detection using a dataset comprising various health-related features such as glucose levels, blood pressure, body mass index (BMI), and genetic predisposition. The model will be trained on historical data from patients diagnosed with or without diabetes, enabling it to learn the underlying patterns and relationships between different variables. Subsequently, the trained model will be evaluated for its accuracy, sensitivity, and specificity in identifying individuals at risk of diabetes.

1.2 OBJECTIVES:

• The number of people diagnosed with diabetes has risen significantly. The current human lifestyle is the primary cause of diabetes rise.

• Main objective of this project is to analyze the data, and see if it is possible to gleam any further information from the data to determine correlation between parameters and diabetes.

• The second is to attempt to get the best accuracy score using various supervised learning machine learning algorithms. To find out which algorithm is able to best predict whether a person has diabetes or not based on this dataset.

• The accuracy of the algorithms used are compared and discussed. The study's comparison of the various machine learning techniques shows which algorithm is better suited for diabetes prediction. Using machine learning methods, this project aims to assist doctors and physicians for predicting whether a person has diabetes or not.

1.3 OVERVIEW OF PROJECT

The diabetes ML prediction project aims to develop machine learning models capable of predicting whether a patient is diabetic based on several health-related features. This project involves various stages, including data preprocessing, exploratory data analysis (EDA), model building, evaluation, and interpretation of results. The ultimate goal is to create accurate and reliable predictive models that can assist healthcare professionals in diagnosing diabetes more effectively.

Here's an overview of the project:
Problem Statement: The project addresses the need for accurate and efficient prediction of diabetes, which is a prevalent health condition affecting millions of people worldwide.
Dataset Description: The dataset used in the project contains information about patients' pregnancies, glucose levels, blood pressure, skin thickness, insulin levels, BMI, diabetes pedigree function, and age, along with their diabetes status (diabetic or non-diabetic).
Data Preprocessing: This stage involves cleaning the dataset, handling missing values, performing feature engineering, and normalizing the data to prepare it for analysis and model training.
Exploratory Data Analysis (EDA): EDA involves visualizing and summarizing the dataset to gain insights into the relationships between variables, identify patterns, and understand the distribution of data.
Model Building: Various machine learning algorithms, such as logistic regression, decision trees, random forests, and support vector machines, are employed to build predictive models. The models are trained on the dataset and evaluated using appropriate performance metrics.
Evaluation: The performance of the trained models is evaluated using metrics such as accuracy, precision, recall, F1-score, and confusion matrices. Cross-validation techniques may also be employed to ensure the robustness of the models.
Interpretation of Results: The results obtained from the models are interpreted to understand their effectiveness in predicting diabetes. Strengths, weaknesses, and areas for improvement are discussed, along with the importance of features in making predictions.
Conclusion: The project concludes by summarizing the key findings, highlighting the significance of the analysis, and providing recommendations for future research or improvements in model performance.
References: A list of references, including documentation, academic papers, datasets, and libraries used in the project, is provided to acknowledge the sources of information and data.

Overall, the project aims to contribute to the field of healthcare by developing accurate predictive models for diabetes diagnosis, thereby facilitating early detection and better management of the condition.

2.DATA ANALYSIS

2.1 STRUCTURE OF DATA:

The dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The datasets consist of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age etc.

Loading the dataset to understand data structure:

```
[2]: data=pd.read_csv("C:\\Users\\hudha\\Downloads\\diabetes.csv")
     data
```

[2]:

	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 × 9 columns

Shape of dataset:

represent total number of rows and columns in Dataset

```
data.shape
```

```
(768, 9)
```

The dataset contains 768 rows and 9 columns

2.2 PARAMETERS IMPLEMENTED:

```
[4]: data.columns  # showing all features

[4]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

It seems like you've provided an overview of the parameters included in the diabetes dataset. These parameters represent various physiological and demographic factors that are commonly associated with the risk of developing diabetes. Here's a summary of each parameter:

Pregnancies: The number of times a person has been pregnant.
Glucose: Plasma glucose concentration measured after a 2-hour oral glucose tolerance test. Elevated glucose levels are indicative of impaired glucose metabolism, a hallmark of diabetes.
Blood Pressure: Diastolic blood pressure, measured in millimeters of mercury (mm Hg). High blood pressure is a risk factor for cardiovascular diseases, which are often associated with diabetes.
Skin Thickness: Triceps skin fold thickness, measured in millimeters. Thicker skin folds have been correlated with insulin resistance and diabetes.
Insulin: 2-hour serum insulin levels measured in micro-units per milliliter (mu U/ml). Elevated insulin levels may indicate insulin resistance, a condition commonly observed in individuals with type 2 diabetes.
BMI (Body Mass Index): A measure of body fat based on height and weight. Higher BMI values are associated with an increased risk of developing diabetes.
DiabetesPedigreeFunction: A function that scores the likelihood of diabetes based on family history.
Age: The age of the individual.
Outcome: The target variable indicating whether the individual has diabetes (1) or not (0). These parameters provide valuable information for predicting the likelihood of diabetes in individuals. Analyzing these parameters and their relationships can help in developing effective predictive models for diabetes diagnosis and management.

2.3 EXPLORATORY DATA ANALYSIS:

Analyzing the dataset and checking any missing values and Dataset information's are checked.

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

There are 768 entries (rows) in the dataset with 7 features and an Outcome column (target variable).

Each column has 768 non-null entries, indicating that there are no missing values.

The data types of the columns are as follows:

Integer: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, Age, and Outcome.

Float: BMI and DiabetesPedigreeFunction.

[7]:
```
data.duplicated().sum()
```

[7]:
```
0
```

From above,it can be inferred that there are no duplicated rows and null values in the dataset.

Summary Statistics:

Calculate descriptive statistics such as mean, median, standard deviation, minimum, maximum, and quartiles for numerical features.

[9]:
```
data.describe()   #  statistical summary of numerical data
```

[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

[10]:
```
df=data.copy()
features=['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI']

for i in features:
    df[i] = df[i].replace(0, df[i].mean())

df.head(2)
```

[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	79.799479	33.6	0.627	50	1
1	1	85.0	66.0	29.0	79.799479	26.6	0.351	31	0

Based on the understanding of the parameters, it seems highly unlikely that glucose, blood pressure, skin thickness, insulin and BMI levels are 0. So, a copy is created.Then replace the 0 values of the impacted columns with the mean values

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Plot histograms, box plots, or kernel density plots to visualize the distribution of numerical features. Use bar plots or pie charts to visualize the distribution of categorical variables.

Based on Target Variable,

2.3.1 PIE DIAGRAM:

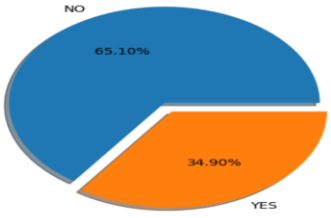
[13]:
```
outcome_count=df["Outcome"].value_counts()
outcome_count
```

[13]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

0 = Non-diabetic patients 1 = diabetic patients

From the above a total of 268 persons have diabeties.

[14]:
```
plt.axis("equal")
plt.pie(outcome_count,labels=["NO","YES"], shadow=True, autopct='%1.2f%%',radius=1,explode=[0.05,0.05])
plt.show()
```

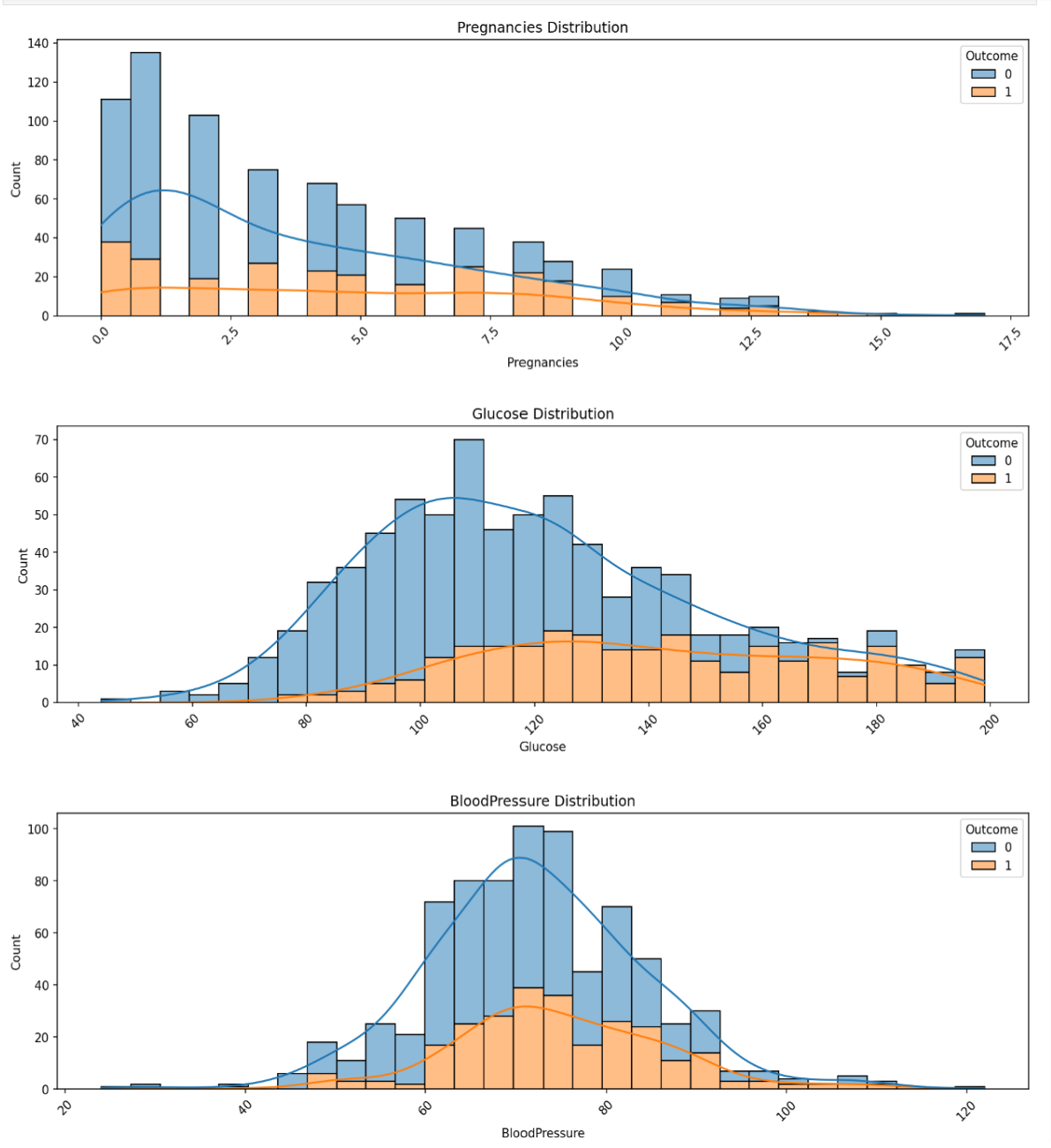


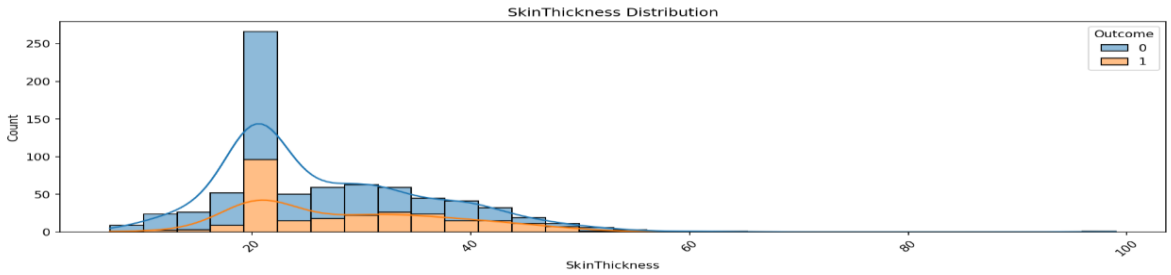
The diabetic rate is 34.90%. This represents a significant percentage of individuals with diabetes, and it is important to understand why these people have the condition.

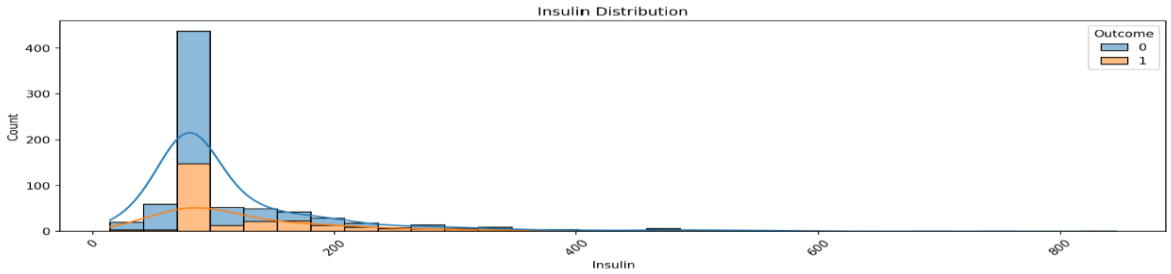
Based on all numerical features,

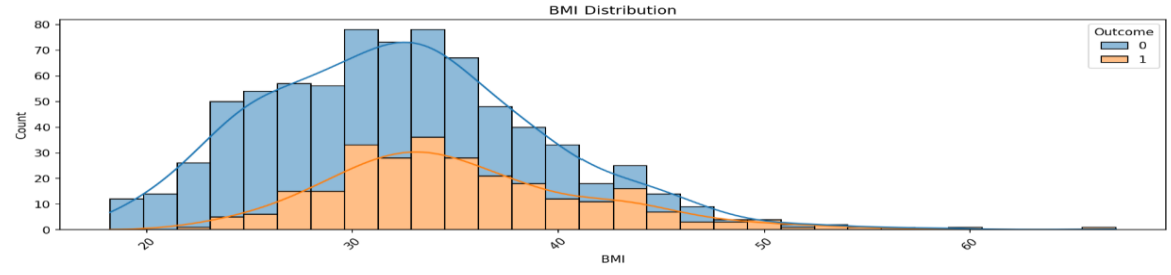
2.3.2 HISTOGRAM:

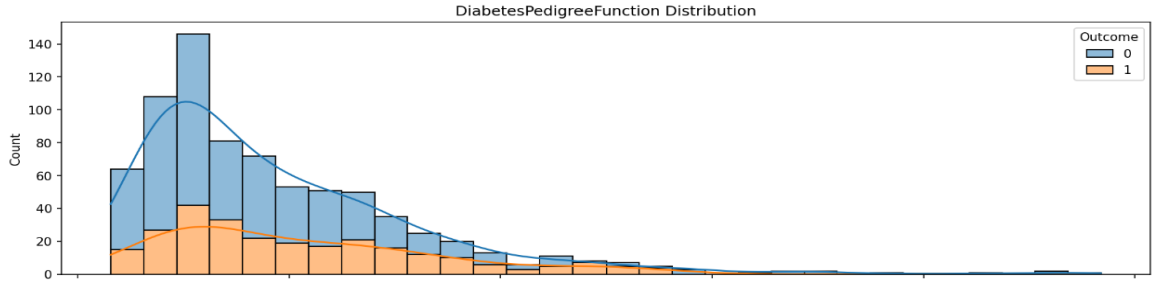
```
[15]: # Define the numerical features
numerical_features=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']

# Set up the subplots
fig, axes = plt.subplots(len(numerical_features), 1, figsize=(15, 5 * len(numerical_features)), dpi=150)

# Iterate through each numerical feature and create a histogram
for i, feature in enumerate(numerical_features):
    sns.histplot(ax=axes[i], data=df, x=feature, hue="Outcome", kde=True, bins=30, multiple="stack")
    axes[i].set_title(f'{feature} Distribution')
    axes[i].tick_params(axis='x', rotation=45, labelsize=10)

# Adjusting layout
plt.subplots_adjust(top=0.95, hspace=0.4)

plt.show()
```

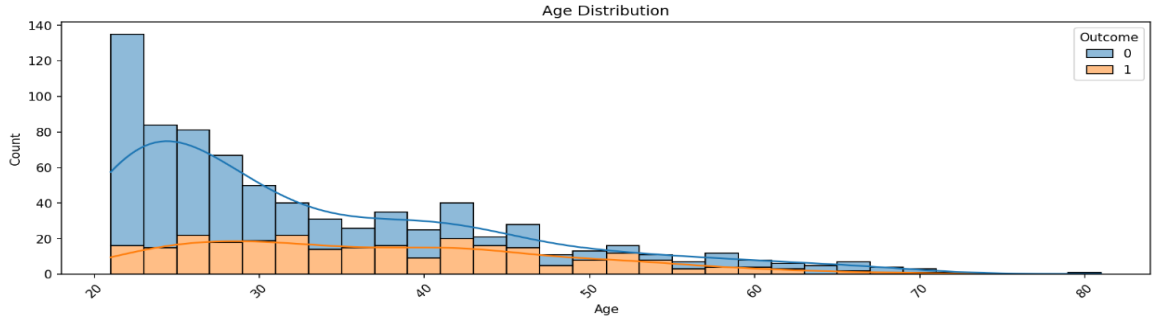








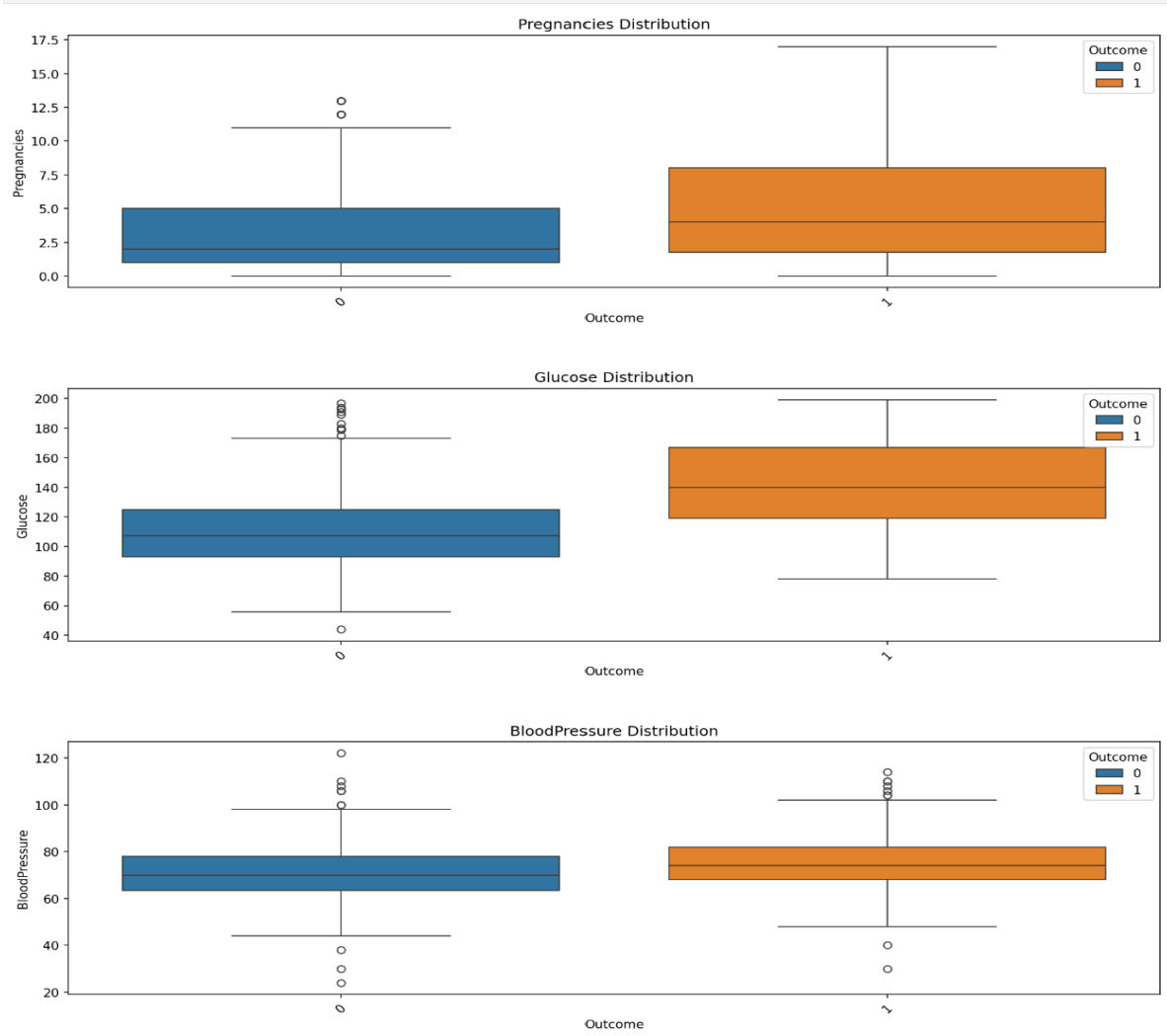


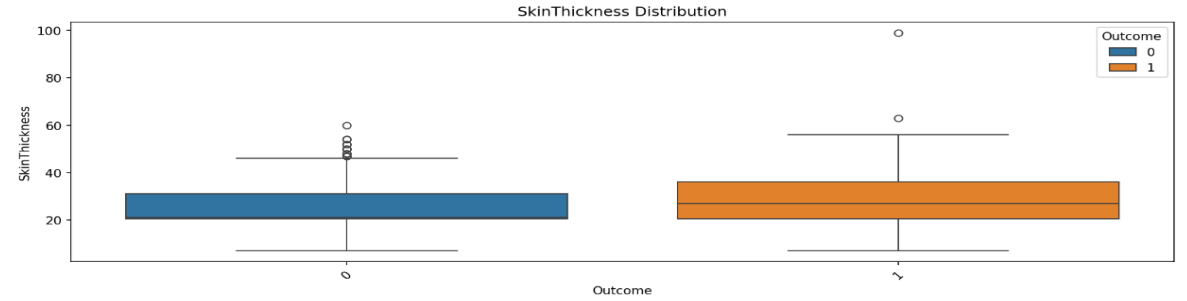


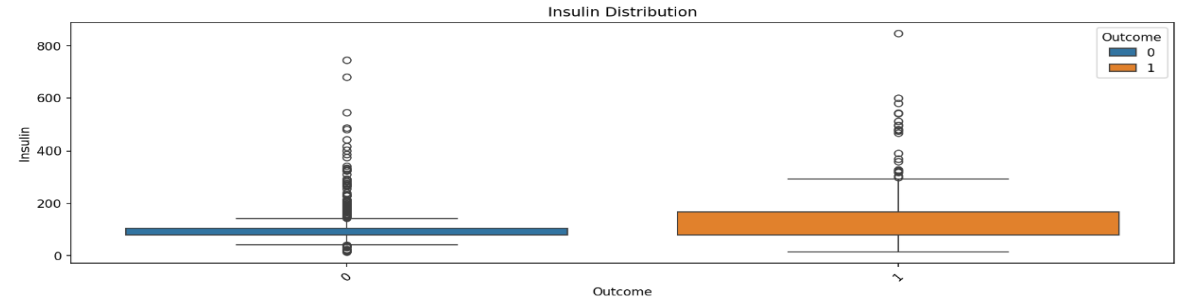
The histograms above show that most of them appear to be positively skewed, with Glucose and Blood Pressure having the closest distribution to a normal distribution.

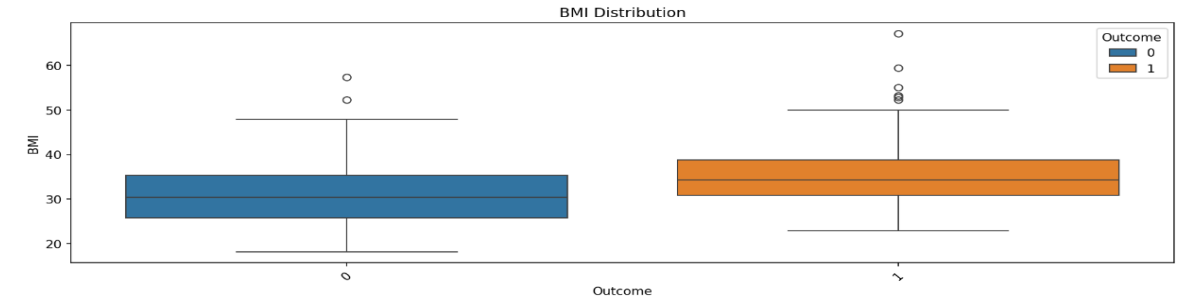
2.3.3 BOX PLOT:

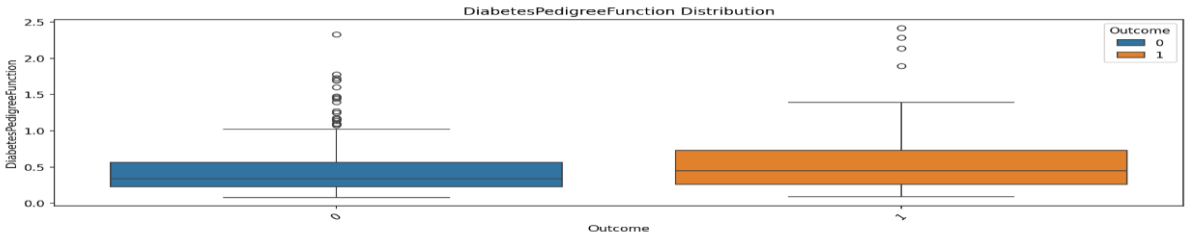
[16]:
```
# Define the numerical features
numerical_features=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']

# Set up the subplots
fig, axes = plt.subplots(len(numerical_features), 1, figsize=(15, 5 * len(numerical_features)), dpi=150)

# Iterate through each numerical feature and create a box plot
for i, feature in enumerate(numerical_features):
    sns.boxplot(ax=axes[i], data=df, x="Outcome", y=feature ,hue="Outcome")
    axes[i].set_title(f'{feature} Distribution')
    axes[i].tick_params(axis='x', rotation=45, labelsize=10)

# Adjusting layout
plt.subplots_adjust(top=0.95, hspace=0.4)

plt.show()
```

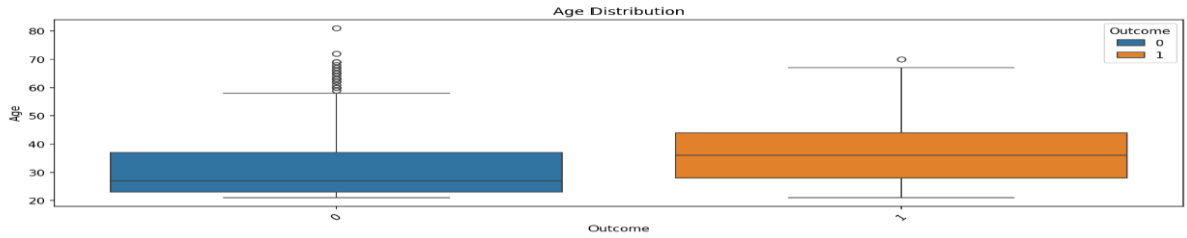








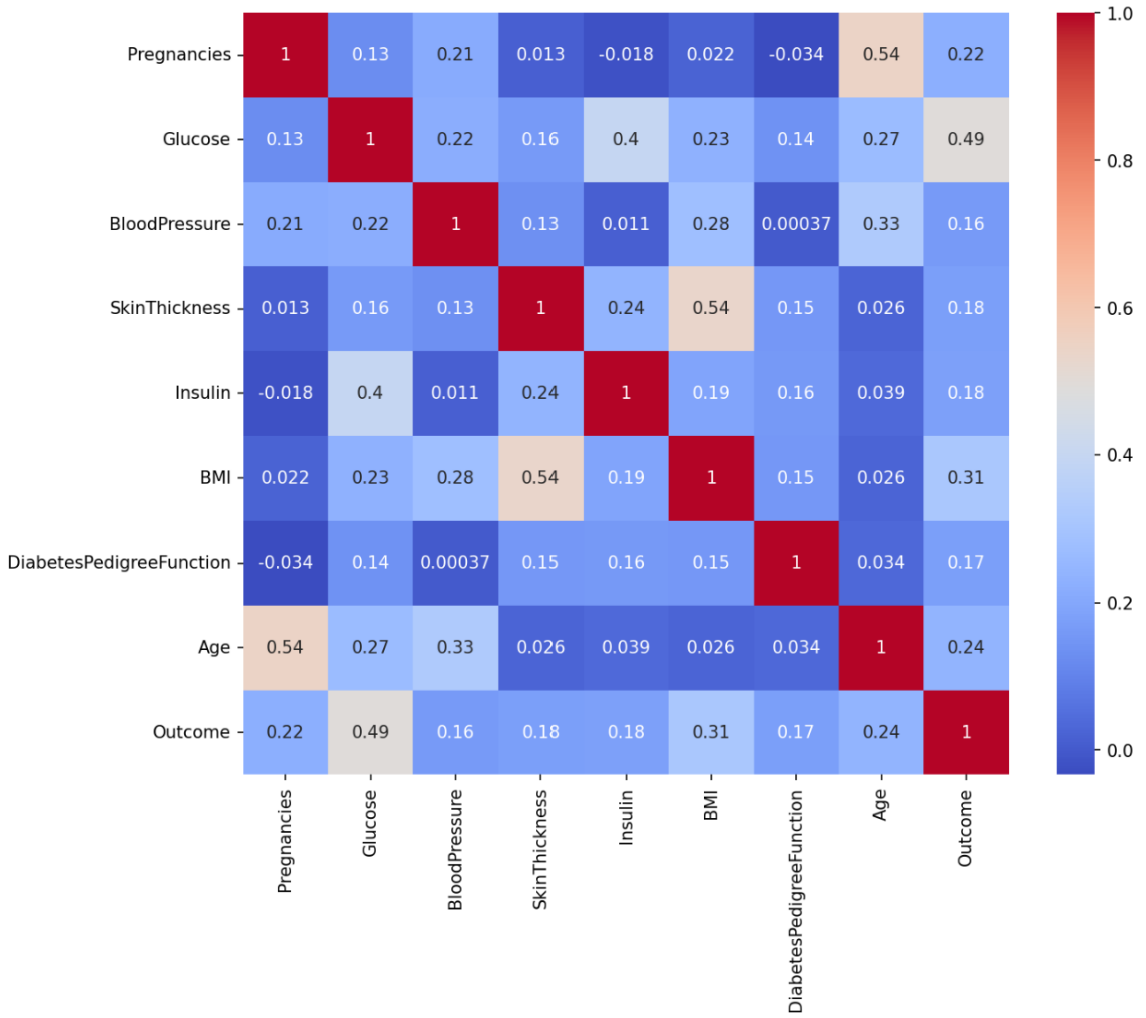




From the boxplots, we observe that as the values of other features such as BMI, blood pressure, insulin,glucose and age increase, there is an increasing trend in the likelihood of individuals having diabetes. It suggests that these features may play a significant role in predicting the likelihood of diabetes.

2.3.4 HEAT MAP:

```
plt.figure(figsize=(10,8),dpi=150)
# Create correlation matrix
corr = df.corr(numeric_only=True)
# Create heatmap
sns.heatmap(corr, annot=True, cmap='coolwarm');
# Creating a heatmap of the correlation matrix is an excellent way to visually represent the relationships
```



Features such as Glucose (0.49) and BMI (0.31) exhibit relatively strong positive correlations with the target variable "Outcome," indicating that higher values of Glucose and BMI are associated with a higher likelihood of diabetes.

Age also shows a moderate positive correlation (0.24) with the target variable, suggesting that older individuals are more likely to have diabetes.

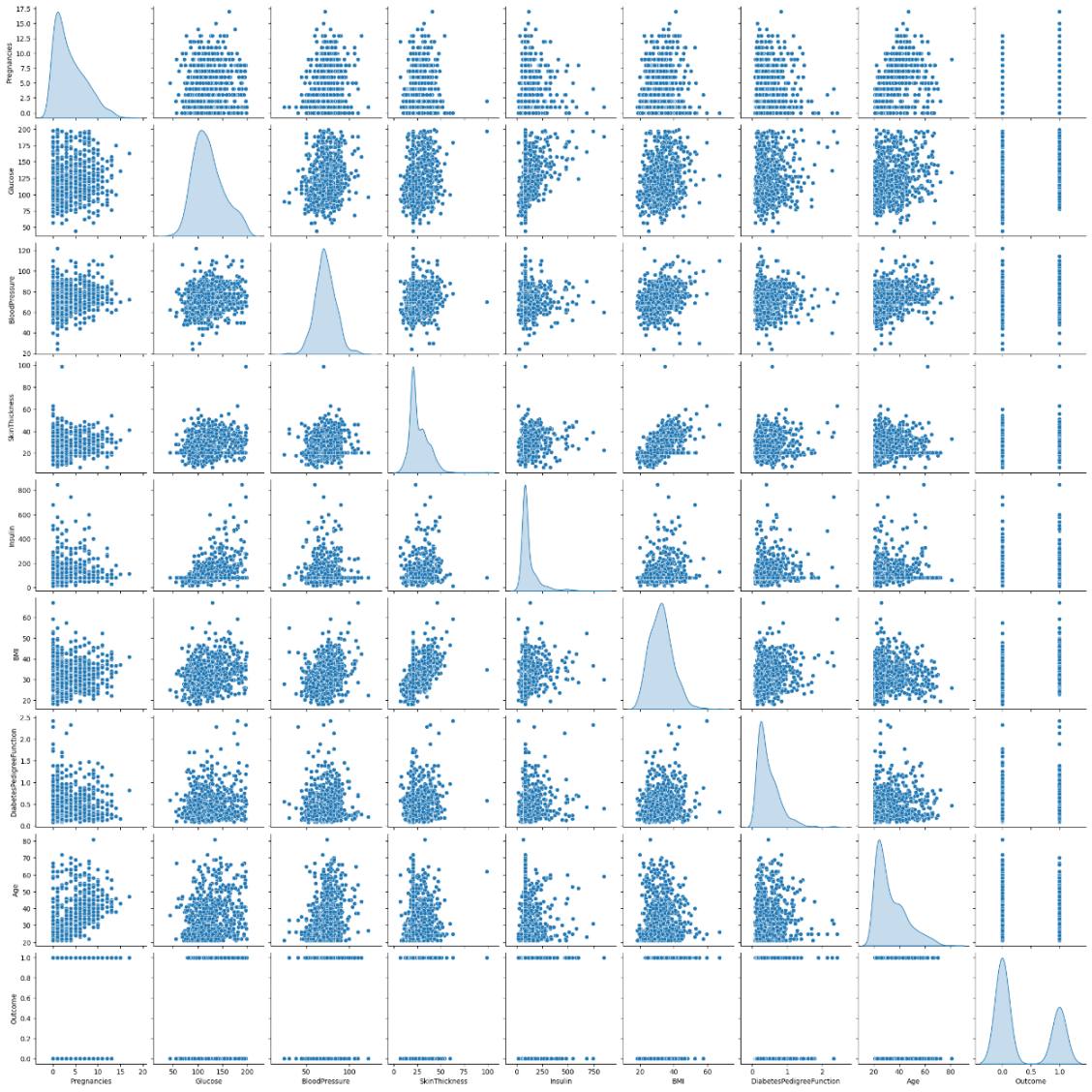
Other features have weaker correlations with the target variable, but some still show notable correlations, such as Insulin (0.18) and SkinThickness (0.18).

The parameter with the highest positive correlation to each other is BMI and Skin Thickness. This is further confirmed by the SNS pair plot. The rest do not have strong multi-collinearity to each other.

2.3.5 SNS PAIR PLOT:

```
[18]: sns.pairplot(df,diag_kind='kde')
```

```
[18]: <seaborn.axisgrid.PairGrid at 0x243dc3641d0>
```



3. IMPLEMENTATION

3.1 FEATURE SCALING:

Here StandardScaler() is used to perform feature scaling. This will retain the mean and the standard deviation of the sample distribution of the data set, and reuse it to transform the X_train and X_test subsequently. I try to reuse the mean and standard deviation obtained from the training set and apply it to the testing set as well. Standardizing data after data splitting is to prevent data leakage from test dataset into train dataset.

[22]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Fit the scaler on the data
scaler.fit(x)

# Transform the data
scaled_data = scaler.transform(x)

# Print the scaled data
print(scaled_data)
```

3.2 SPLITTING OF DATASET (TRAINING/VAILDATION/TESTING) :

The splitting of the dataset for validation and testing. Training Dataset: Dataset sample that is used to fit the model. Validation Dataset: Dataset sample that is used for hyper tuning the parameters, and comparing the accuracy and error rates of the model performance between using the training dataset and the validation dataset. Testing Dataset: Dataset sample that is used to test the model performance (predictive power).

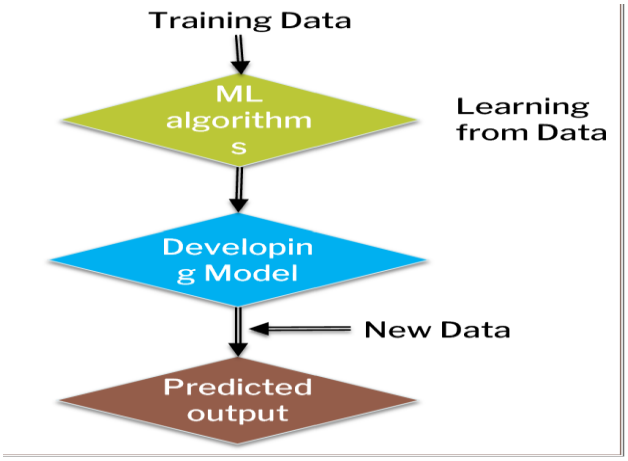
]:
```
from sklearn.model_selection import train_test_split

X = scaled_data
Y = df1['Outcome']
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, random_state=0)
```

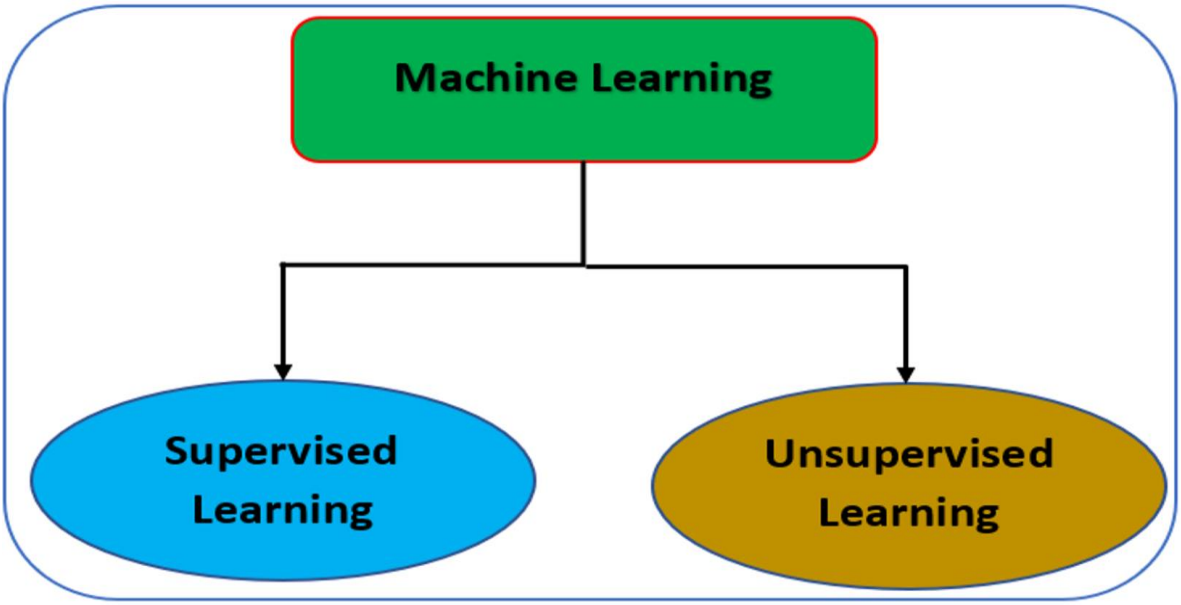
]:
```
print("Shape of x_train: ",x_train.shape)
print("Shape of x_test: ", x_test.shape)
print("Shape of y_train: ",y_train.shape)
print("Shape of y_test:",y_test.shape)
```

```
Shape of x_train:  (614, 8)
Shape of x_test:  (154, 8)
Shape of y_train:  (614,)
Shape of y_test: (154,)
```

3.3 IMPLEMENTING MACHINE LEARNING ALGORITHMS:

- **Machine Learning is the subset of Artificial Intelligence.**
- **ML enables computers to learn and improve from experience without being explicitly programmed.**
- **ML is based on algorithms that learn from previous data.**
- **With minimal to zero human intervention, machines and systems are able to analyse and understand data, learn from it and make decisions.**
- **The level of performance will increase as we supply more information.**





	Classification	Regression
Role	To predict a discrete class label	To predict a continuous quantity
Type of output/response variable	Output value is discrete/categorical	Output value is continuous(Numeric)
Type of algorithm	Supervised Learning	Supervised Learning
Performance measure	Percentage of correct classifications	Root mean squared error
Applications	Classifying an email as spam or non-spam, type of tumour i.e. harmful or not harmful, credit card transaction is fraud or not, image classification	Predicting the price of a stock over a period of time, predicting house selling price prediction

Supervised learning alogorithm using here,

1--Logistic Regression

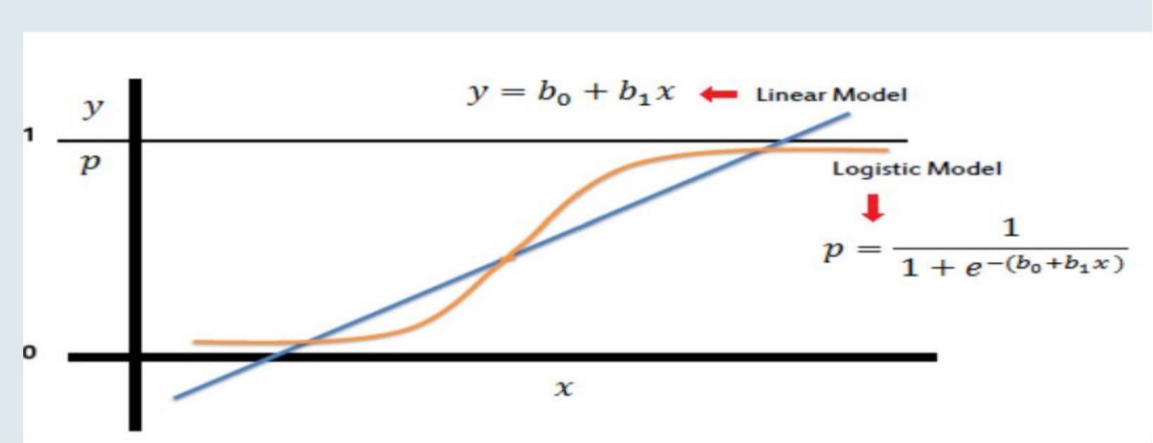
2--Decision Tree Classifier

3--Random Forest Classifier

4--Support Vector Classifier

3.3.1 LOGISTIC REGRESSION MODEL:

Logistic regression is a statistical method used for predicting categorical outcomes. It is a supervised learning algorithm that is closely related to linear regression but adapted for binary or multi-class classification problems. The goal of logistic regression is to estimate the probability of an event occurring, which is typically represented as 0 or 1.



[25]:
```
from sklearn.linear_model import LogisticRegression

#creating Logistic regression model
logistic_model = LogisticRegression()

#training
logistic_model.fit(x_train, y_train)

# prediction
logistic_pred=logistic_model.predict(x_test)

logistic_model.score(x_test,y_test)
```

[25]:
```
0.8181818181818182
```

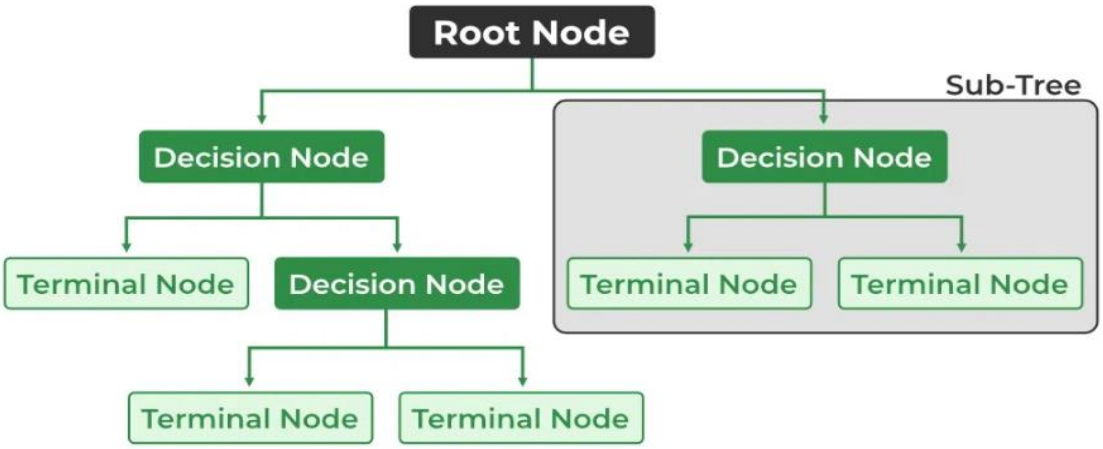
Evaluate model

[26]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
logistic_accuracy = accuracy_score(y_test, logistic_pred)
print("Logistic_Accuracy:", logistic_accuracy)

# Print a classification report
print(classification_report(y_test,logistic_pred))

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test,logistic_pred))
```

```
Logistic_Accuracy: 0.8181818181818182
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       107
           1       0.76      0.60      0.67        47

    accuracy                           0.82       154
   macro avg       0.80      0.76      0.77       154
weighted avg       0.81      0.82      0.81       154

Confusion Matrix:
[[98  9]
 [19 28]]
```

3.3.2 DECISION TREE MODEL:

- A decision tree is a supervised learning algorithm that can be used for classification and regression modeling, but mostly it is preferred for solving Classification problems
- It is a flowchart-like tree structure where each internal node denotes the feature, branches denote the rules and the leaf nodes denote the result of the algorithm
- It is called as decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure



[27]:
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Create a Decision Tree classifier
DT_model= DecisionTreeClassifier(criterion="gini")

# Fit the model to the training data
DT_model.fit(x_train, y_train)

# Make predictions on the test data
DT_pred = DT_model.predict(x_test)

DT_model.score(x_test,y_test)
```

[27]:
```
0.7532467532467533
```

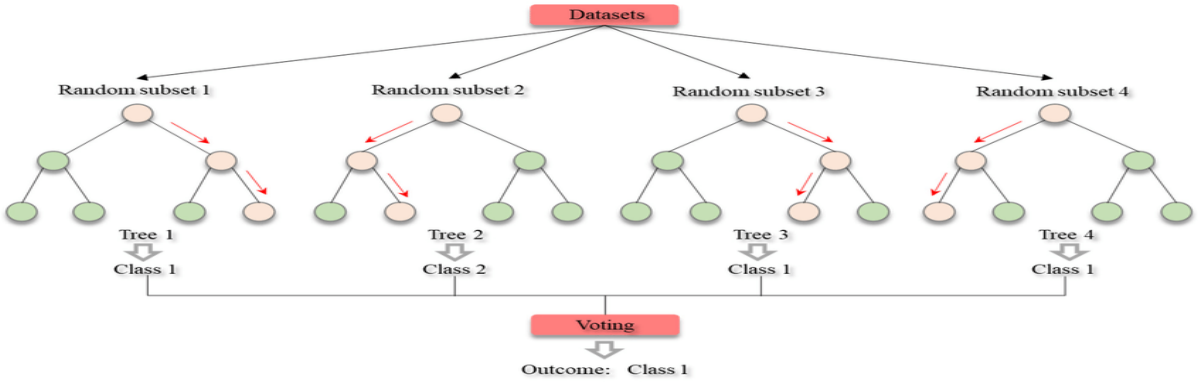
Evaluate model

[28]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
DT_accuracy = accuracy_score(y_test, DT_pred)
print("DT_Accuracy:", DT_accuracy)

# Print a classification report
print(classification_report(y_test,DT_pred))

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test,DT_pred))
```

```
DT_Accuracy: 0.7532467532467533
              precision    recall  f1-score   support

           0       0.84      0.79      0.82       107
           1       0.58      0.66      0.62        47

    accuracy                           0.75       154
   macro avg       0.71      0.73      0.72       154
weighted avg       0.76      0.75      0.76       154

Confusion Matrix:
[[85 22]
 [16 31]]
```

3.3.3 RANDOM FOREST MODEL:

- Random forests or random decision forests is an ensemble learning method
- Ensemble learning : a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model
- It can be used for both Classification and Regression problems in ML
- For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned



[29]:
```
from sklearn.ensemble import RandomForestClassifier

# Random Forest classifier
RF_model = RandomForestClassifier(n_estimators=20)  # You can adjust the number of estimators as needed

# Fit the model to the training data
RF_model.fit(x_train, y_train)

# Make predictions on the test data
RF_pred = RF_model.predict(x_test)


RF_model.score(x_test,y_test)
```

[29]:
```
0.7857142857142857
```

Evaluate model

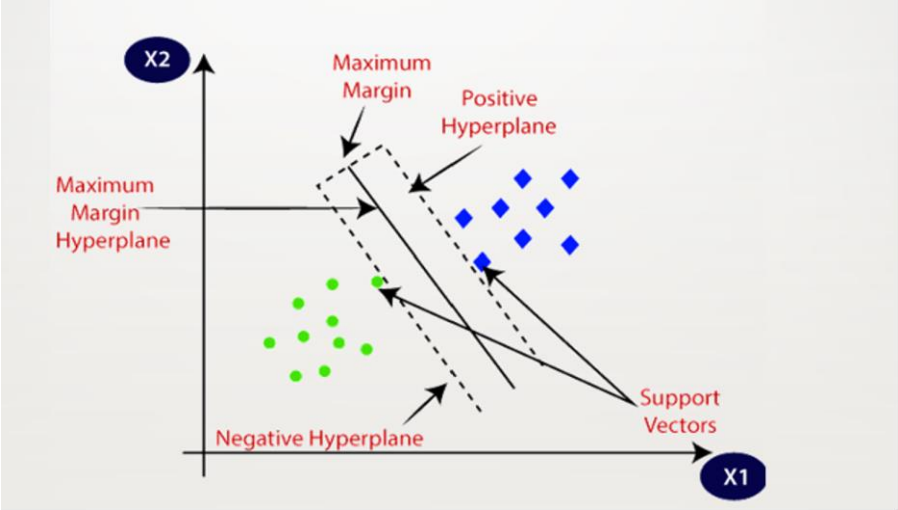
[30]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
RF_accuracy = accuracy_score(y_test, RF_pred)
print("RF_Accuracy:", RF_accuracy)

# Print a classification report
print(classification_report(y_test,RF_pred))

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test,RF_pred))
```

```
RF_Accuracy: 0.7857142857142857
              precision    recall  f1-score   support

           0       0.84      0.86      0.85       107
           1       0.66      0.62      0.64        47

    accuracy                           0.79       154
   macro avg       0.75      0.74      0.74       154
weighted avg       0.78      0.79      0.78       154

Confusion Matrix:
[[92 15]
 [18 29]]
```

3.3.4 SUPPORT VECTOR MACHINE MODEL:

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning
- Goal: To create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

Two different categories are classified using a decision boundary or hyperplane.



[31]:
```
from sklearn.svm import SVC

# Create an SVM classifier
svm_model = SVC(kernel='linear')  # You can choose different kernel functions (linear, rbf, etc.)

# Fit the model to the training data
svm_model.fit(x_train, y_train)

# Make predictions on the test data
svm_pred = svm_model.predict(x_test)


svm_model.score(x_test,y_test)
```

[31]:
```
0.8051948051948052
```

Evaluate model

[32]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
svm_accuracy= accuracy_score(y_test, svm_pred)
print("SVM_Accuracy:", svm_accuracy)

# Print a classification report
print(classification_report(y_test,svm_pred))

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test,svm_pred))
```

```
SVM_Accuracy: 0.8051948051948052
              precision    recall  f1-score   support

           0       0.83      0.91      0.87       107
           1       0.73      0.57      0.64        47

    accuracy                           0.81       154
   macro avg       0.78      0.74      0.75       154
weighted avg       0.80      0.81      0.80       154

Confusion Matrix:
[[97 10]
 [20 27]]
```

4. TEST RESULTS

4.1 COMPARING DIFFERENT MODELS:

[33]:

	Model	Model Accuracy
0	Logistic Regression	0.818182
3	Support Vector Machines	0.805195
2	Random Forest	0.785714
1	Decission Tree	0.753247

According to the accuracy scores, Logistic Regression has the highest accuracy, followed by Random Forest and Support Vector Machines. Decision Tree has the lowest accuracy among the models evaluated.

Logistic model accuracy is 0.8182

4.2 CROSS VALIDATION TECHNIQUE:

For more clarification we can use cross validation technique,

Cross-validation is a technique used in machine learning to assess the performance of a predictive model. It involves splitting the dataset into multiple subsets, typically called folds. The model is trained on a portion of the data (training set) and evaluated on the remaining portion (validation set). This process is repeated multiple times, with each fold serving as the validation set exactly once.

The main purpose of cross-validation is to provide a more reliable estimate of the model's performance compared to a single train-test split. It helps to assess the model's generalization ability by reducing the variance associated with a single train-test split.

Common types of cross-validation include k-fold cross-validation, stratified k-fold cross-validation, and leave-one-out cross-validation. These techniques help ensure that the model's performance evaluation is robust and not overly dependent on a specific train-test split.

```
from sklearn.model_selection import cross_val_score

# Define the models
models = [
    LogisticRegression(solver='liblinear',multi_class="ovr"),
    DecisionTreeClassifier(criterion="gini"),
    RandomForestClassifier(n_estimators=40),
    SVC(kernel='linear')
]

# Perform cross-validation for each model and calculate the mean accuracy
cv_results = []
for model in models:
    scores = cross_val_score(model, X, Y, cv=5)  # 5-fold cross-validation
    cv_results.append(scores.mean())

# Create a dataframe to display the results
cv_df = pd.DataFrame({'Model': ['Logistic Regression', 'Decision Tree', 'Random Forest', 'Support Vector Machines'],
                      'Cross-Validation Accuracy': cv_results})

# Sort the dataframe by cross-validation accuracy
cv_df = cv_df.sort_values(by='Cross-Validation Accuracy', ascending=False)

# Display the results
print(cv_df)
```

	Model	Cross-Validation Accuracy
0	Logistic Regression	0.769570
3	Support Vector Machines	0.766972
2	Random Forest	0.761727
1	Decision Tree	0.712367

Logistic Regression has the highest cross-validation accuracy among the models evaluated, followed closely by Support Vector Machines and Random Forest. Decision Tree has the lowest cross-validation accuracy.

4.3 FINAL RESULT:

From the above results, we determine that the best model for prediction is the Logistic Regression model.

```
print(classification_report(y_test,logistic_pred))
```

	precision	recall	f1-score	support
0	0.84	0.92	0.88	107
1	0.76	0.60	0.67	47
accuracy			0.82	154
macro avg	0.80	0.76	0.77	154
weighted avg	0.81	0.82	0.81	154

Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positives. It measures the accuracy of positive predictions. A high precision indicates that the classifier has a low false positive rate.

Recall (also called sensitivity or true positive rate): Recall is the ratio of correctly predicted positive observations to the all observations in actual class. It measures the ability of the classifier to find all positive instances. A high recall indicates that the classifier has a low false negative rate.

F1-score: The F1-score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall. It's useful when you have an uneven class distribution.

Support: Support is the number of actual occurrences of the class in the specified da

Accuracy: Accuracy is the ratio of correctly predicted observations to the total observations in the dataset. In classification, it measures the overall correctness of the model's predictions across all classes. A high accuracy indicates that the model is making correct predictions for the majority of the observations.

Macro Average: The macro average calculates the metric (such as precision, recall, or F1-score) separately for each class and then takes the average across all classes. It treats all classes equally, regardless of their support (number of instances). Each class contributes

equally to the final score, making it useful when you want to evaluate the model's performance across all classes equally.

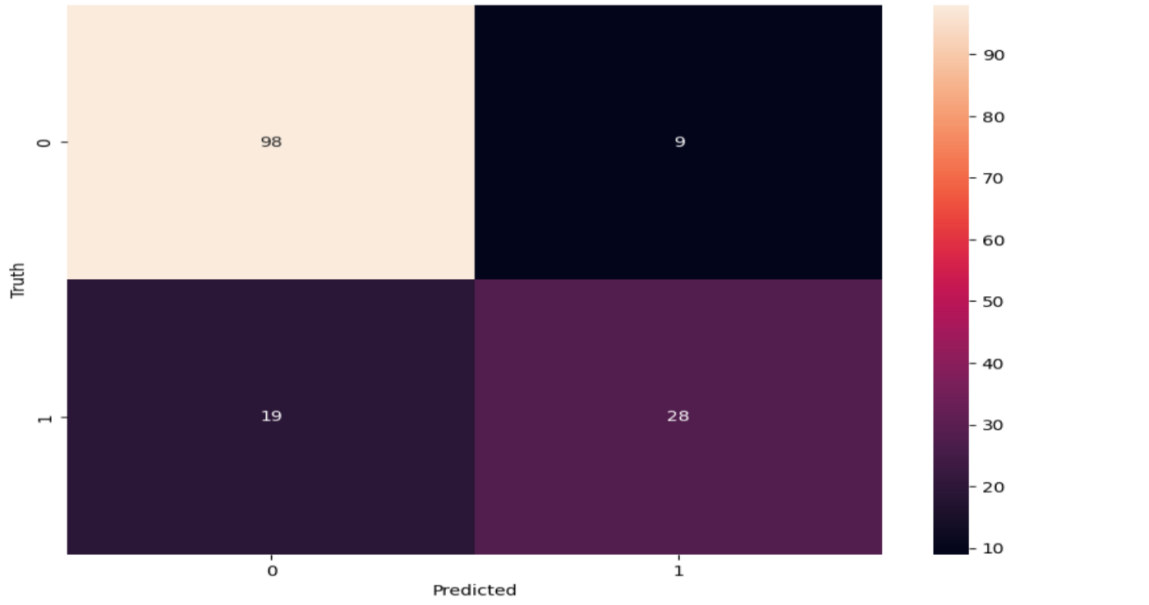
Weighted Average: The weighted average calculates the metric (such as precision, recall, or F1-score) for each class, weighted by its support (number of instances), and then takes the average across all classes. It gives more weight to classes with more instances, making it useful when you have an imbalanced dataset. Weighted average is often preferred in scenarios where classes have significantly different supportaset.

Confusion matrix:

A confusion matrix is a table used to evaluate the performance of a classification model. It provides a summary of the model's predictions and how they compare to the actual labels in the dataset. The matrix is structured as follows:

True Positive (TP): Instances where the model correctly predicts a positive class.
True Negative (TN): Instances where the model correctly predicts a negative class.
False Positive (FP): Instances where the model incorrectly predicts a positive class (Type I error).
False Negative (FN): Instances where the model incorrectly predicts a negative class (Type II error).

```
[36]: confusion_matrix = confusion_matrix(y_test, logistic_pred)

      plt.figure(figsize=(10, 7))
      sns.heatmap(confusion_matrix, annot=True,fmt='d')    # 'd' stands for decimal format
      plt.xlabel('Predicted')
      plt.ylabel('Truth')
      plt.show()
```



Conclusion:

From the confusion matrix we can see that: There are total 98+9=107 actual non_diabetic patients and the algorithm predicts 98 of them as non diabetic and 9 of them as diabetic. While there are 19+28=47 actual diabetic patients and the algorithm predicts 19 of them as non-diabetic and 28 of them as diabetic.

5. PREDICTION

sample_data 1

[38]:
```
Pregnancies = int(input("No.of times Pregnancies:"))
Glucose = float(input("Glucose:"))
BloodPressure = float(input("BloodPressure:"))
SkinThickness = float(input("SkinThickness:"))
Insulin = float(input("Insulin:"))
BMI = float(input("BMI:"))
DiabetesPedigreeFunction = float(input("DiabetesPedigreeFunction:"))
Age = int(input("Age:"))

new_data = np.array([[Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin,
                      BMI, DiabetesPedigreeFunction, Age]])
std_data = scaler.transform(new_data)
logistic_pred_new = logistic_model.predict(std_data)

# Mapping the prediction
print("-----------------------")
if logistic_pred_new[0] == 0:
    print(logistic_pred_new, "- The person is not diabetic")
else:
    print(logistic_pred_new, "- The person is diabetic")
```

```
No.of times Pregnancies: 1
Glucose: 50
BloodPressure: 23
SkinThickness: 111
Insulin: 12
BMI: 123
DiabetesPedigreeFunction: 11
Age: 23
-----------------------
[1] - The person is diabetic
```

sample_data 2

```
Pregnancies = int(input("No.of times Pregnancies:"))
Glucose = float(input("Glucose:"))
BloodPressure = float(input("BloodPressure:"))
SkinThickness = float(input("SkinThickness:"))
Insulin = float(input("Insulin:"))
BMI = float(input("BMI:"))
DiabetesPedigreeFunction = float(input("DiabetesPedigreeFunction:"))
Age = int(input("Age:"))

new_data = np.array([[Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin,
                      BMI, DiabetesPedigreeFunction, Age]])
std_data = scaler.transform(new_data)
logistic_pred_new = logistic_model.predict(std_data)

# Mapping the prediction
print("-----------------------")
if logistic_pred_new[0] == 0:
    print(logistic_pred_new, "- The person is not diabetic")
else:
    print(logistic_pred_new, "- The person is diabetic")
```

```
No.of times Pregnancies: 6
Glucose: 148
BloodPressure: 72
SkinThickness: 35
Insulin: 0
BMI: 33.6
DiabetesPedigreeFunction: 0.627
Age: 50
-----------------------
[1] - The person is diabetic
```

sample_data 3

[40]:
```
Pregnancies = int(input("No.of times Pregnancies:"))
Glucose = float(input("Glucose:"))
BloodPressure = float(input("BloodPressure:"))
SkinThickness = float(input("SkinThickness:"))
Insulin = float(input("Insulin:"))
BMI = float(input("BMI:"))
DiabetesPedigreeFunction = float(input("DiabetesPedigreeFunction:"))
Age = int(input("Age:"))

new_data = np.array([[Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin,
                      BMI, DiabetesPedigreeFunction, Age]])
std_data = scaler.transform(new_data)
logistic_pred_new = logistic_model.predict(std_data)

# Mapping the prediction
print("-----------------------")
if logistic_pred_new[0] == 0:
    print(logistic_pred_new, "- The person is not diabetic")
else:
    print(logistic_pred_new, "- The person is diabetic")
```

```
No.of times Pregnancies: 5
Glucose: 121
BloodPressure: 72
SkinThickness: 23
Insulin: 112
BMI: 26.2
DiabetesPedigreeFunction: 0.245
Age: 30
-----------------------
[0] - The person is not diabetic
```

sample_data 4

[41]:
```
Pregnancies = int(input("No.of times Pregnancies:"))
Glucose = float(input("Glucose:"))
BloodPressure = float(input("BloodPressure:"))
SkinThickness = float(input("SkinThickness:"))
Insulin = float(input("Insulin:"))
BMI = float(input("BMI:"))
DiabetesPedigreeFunction = float(input("DiabetesPedigreeFunction:"))
Age = int(input("Age:"))

new_data = np.array([[Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin,
                      BMI, DiabetesPedigreeFunction, Age]])
std_data = scaler.transform(new_data)
logistic_pred_new = logistic_model.predict(std_data)

# Mapping the prediction
print("------------------------")
if logistic_pred_new[0] == 0:
    print(logistic_pred_new, "- The person is not diabetic")
else:
    print(logistic_pred_new, "- The person is diabetic")
```

```
No.of times Pregnancies: 1
Glucose: 93
BloodPressure: 70
SkinThickness: 31
Insulin: 0
BMI: 31
DiabetesPedigreeFunction: 0.315
Age: 23
------------------------
[0] - The person is not diabetic
```

6. REFFERENCES

- Dataset :Pima Indians Diabetes Database (kaggle.com)
- Numpy :https://numpy.org/doc/
- Pandas: https://pandas.pydata.org/docs/
- Matplotlib: https://matplotlib.org/stable/users/index.html
- Seaborn: https://seaborn.pydata.org/tutorial.html
- Scikit-learn: https://scikit-learn.org/stable/index.html