Importing python libraries

In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

Load the dataset

In [4]:
```
df = pd.read_csv('diabetes.csv')
```

Displaying the first 5 rows of the dataframe

In [5]:
```
df.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

Display the shape of the dataframe

In [6]:
```
df.shape
```

Out[6]:
```
(768, 9)
```

Statistical summary of the data frame

In [7]:
```
df.describe()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dial
count	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	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

Counting the 'Outcome' c0lumn

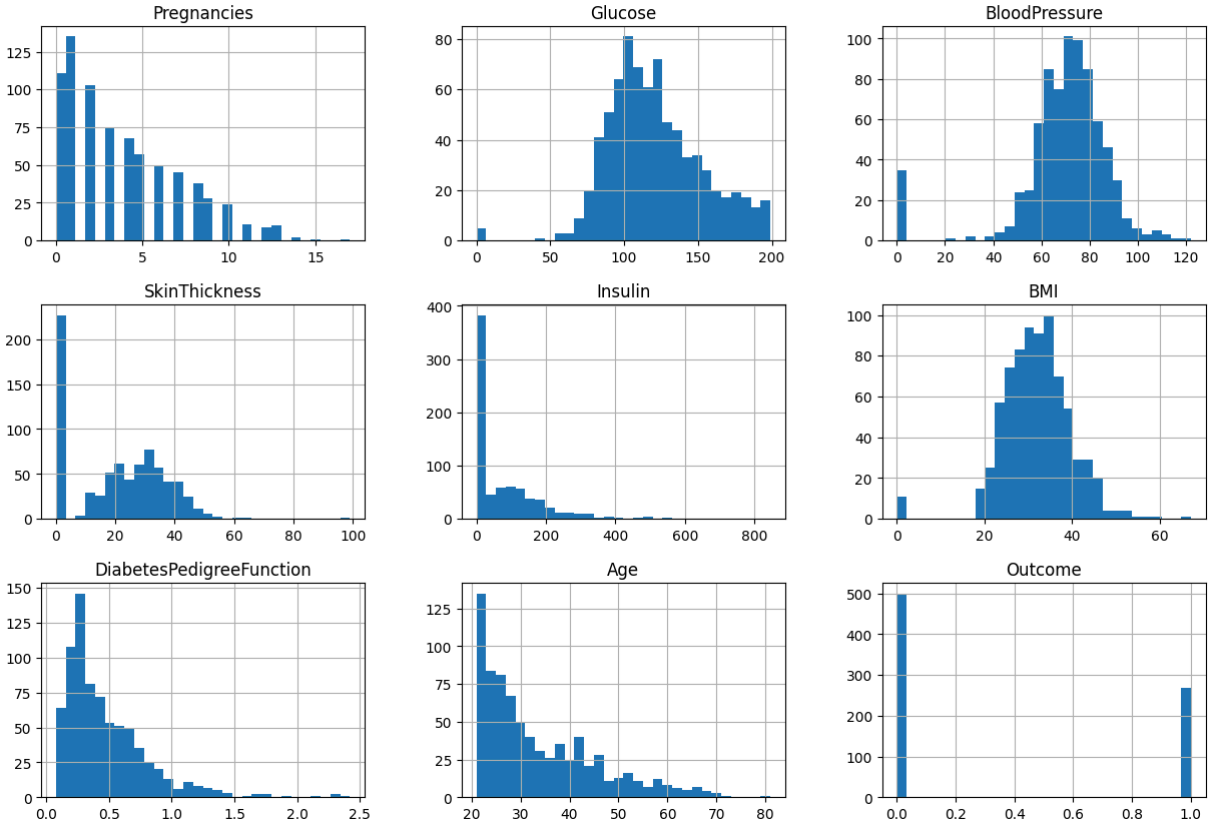
In [8]:
```
df['Outcome'].value_counts()
```

Out[8]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

Histogram

In [9]:
```
df.hist(bins=30,figsize =(15,10))
plt.suptitle("Histograms of all features")
plt.show()
```

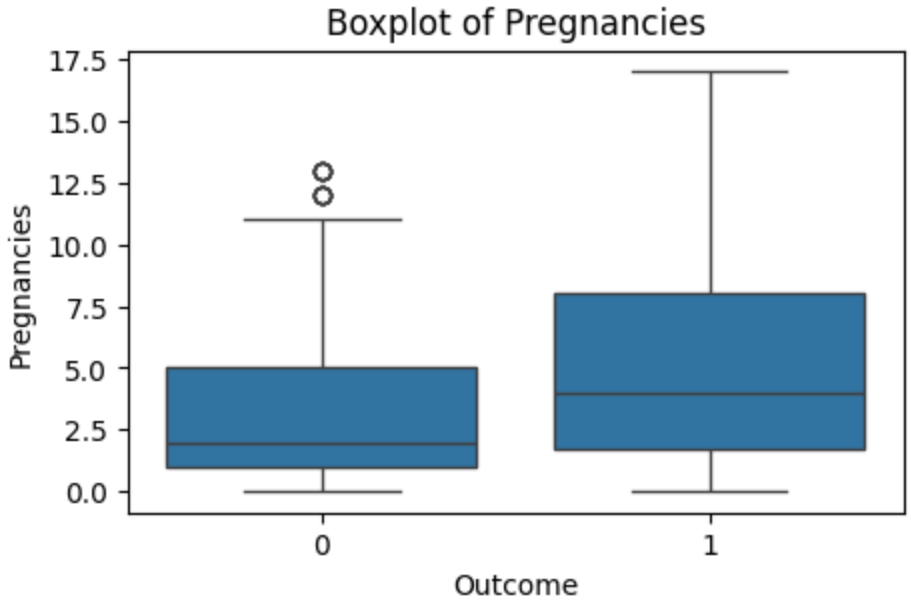
Histograms of all features

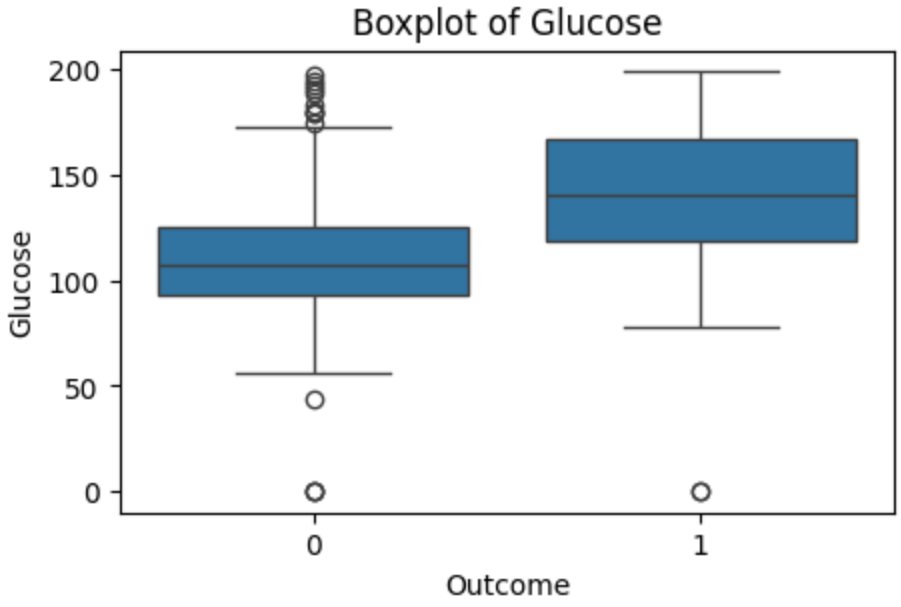


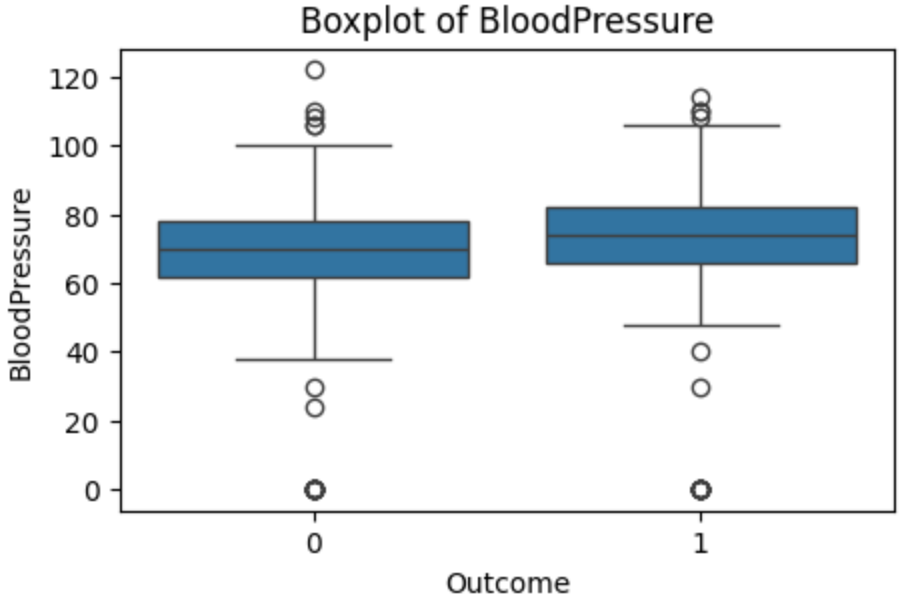
Separate the features and the target variable

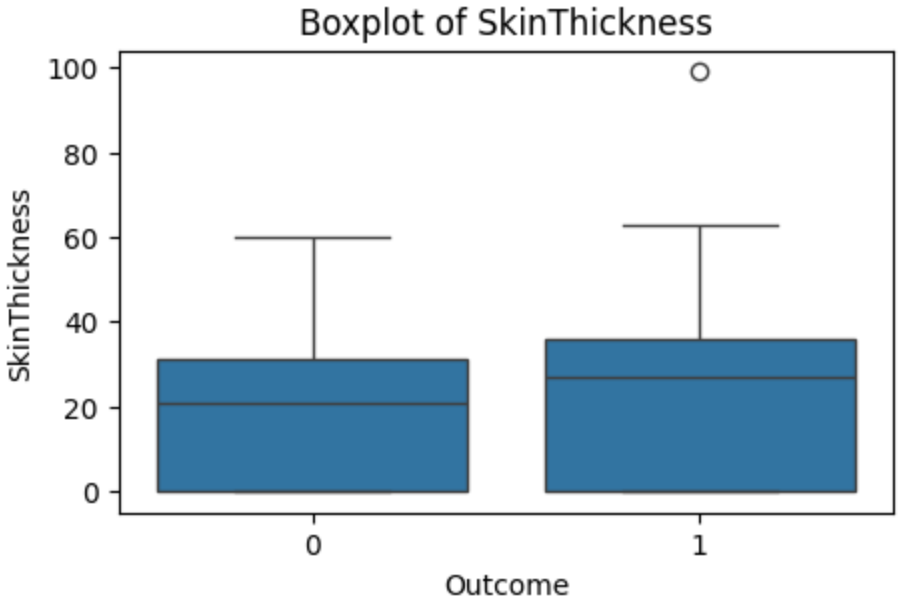
Boxplots to compare distributions of features for diabetic and non-diabetic patients

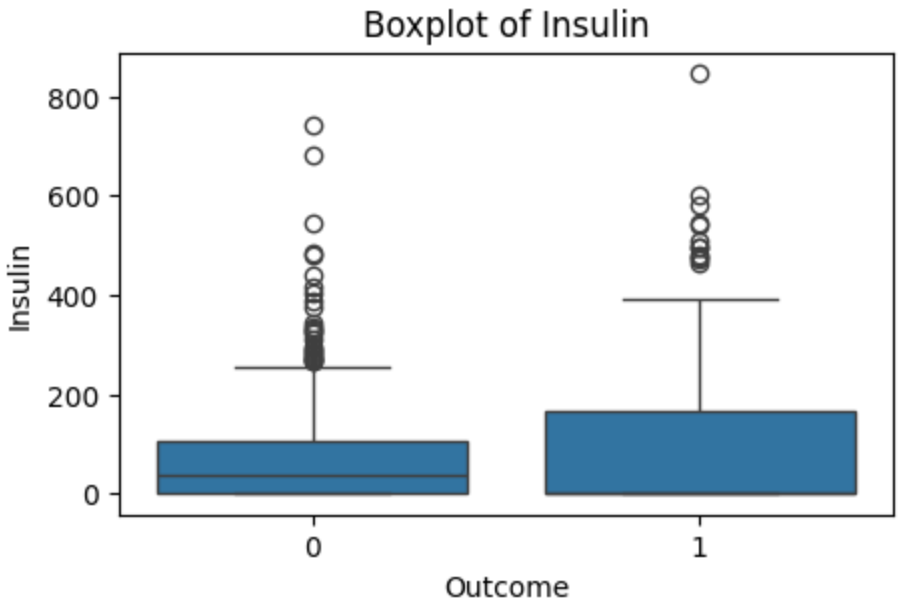
In [10]:
```
for column in df.columns[:-1]:  # Exclude the 'Outcome' column
    plt.figure(figsize=(5, 3))
    sns.boxplot(x='Outcome', y=column, data=df)
    plt.title(f"Boxplot of {column}")
    plt.show()
```

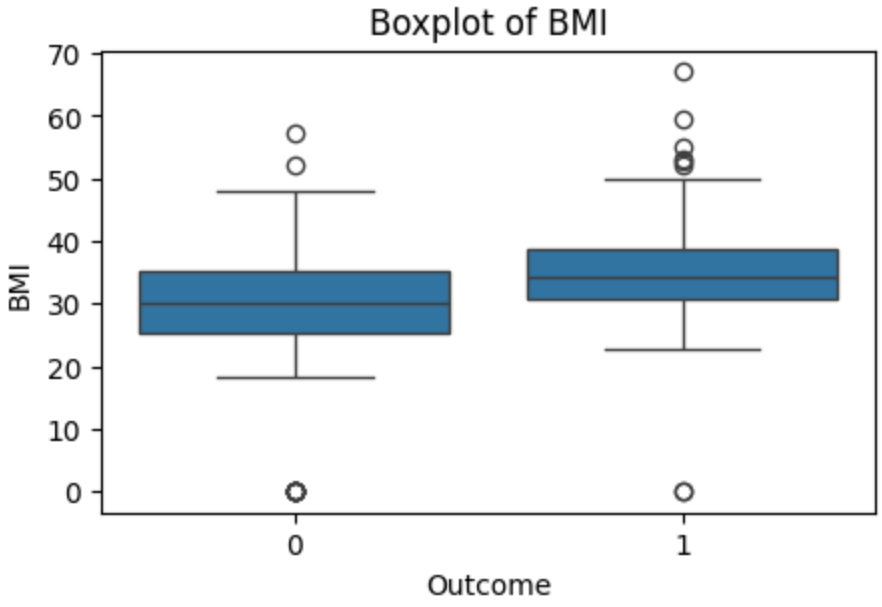

Boxplot of Pregnancies
Pregnancies
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
Outcome

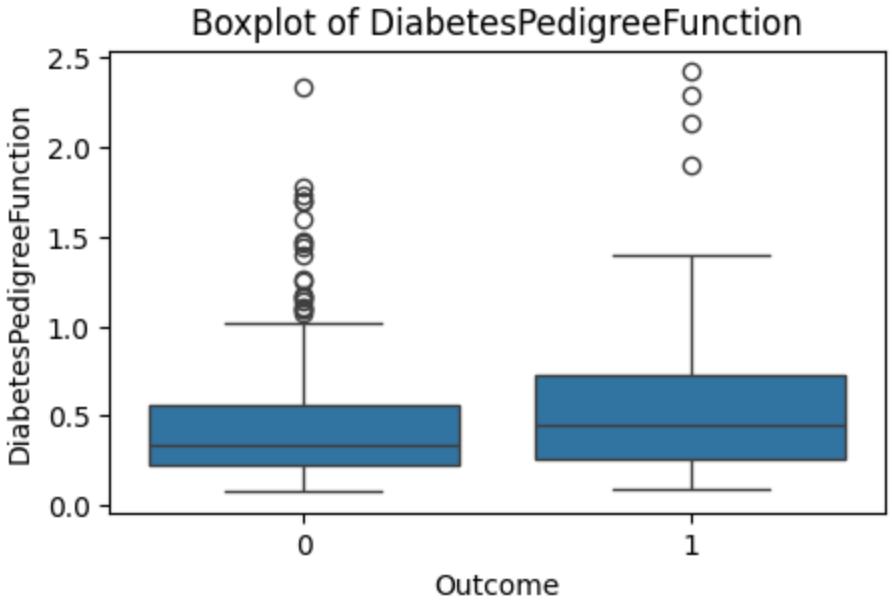

Boxplot of Glucose
Glucose
200
150
100
50
0
0
1
Outcome

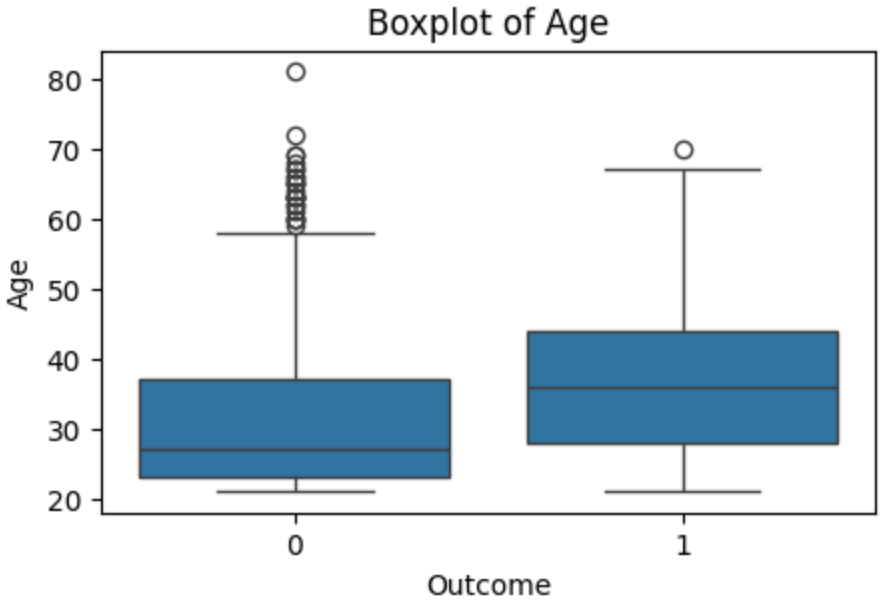

Boxplot of BloodPressure
BloodPressure
120
100
80
60
40
20
0
0
1
Outcome


Boxplot of SkinThickness
SkinThickness
100
80
60
40
20
0
0
1
Outcome


Boxplot of Insulin
Insulin
800
600
400
200
0
0
1
Outcome


Boxplot of BMI
BMI
70
60
50
40
30
20
10
0
0
1
Outcome





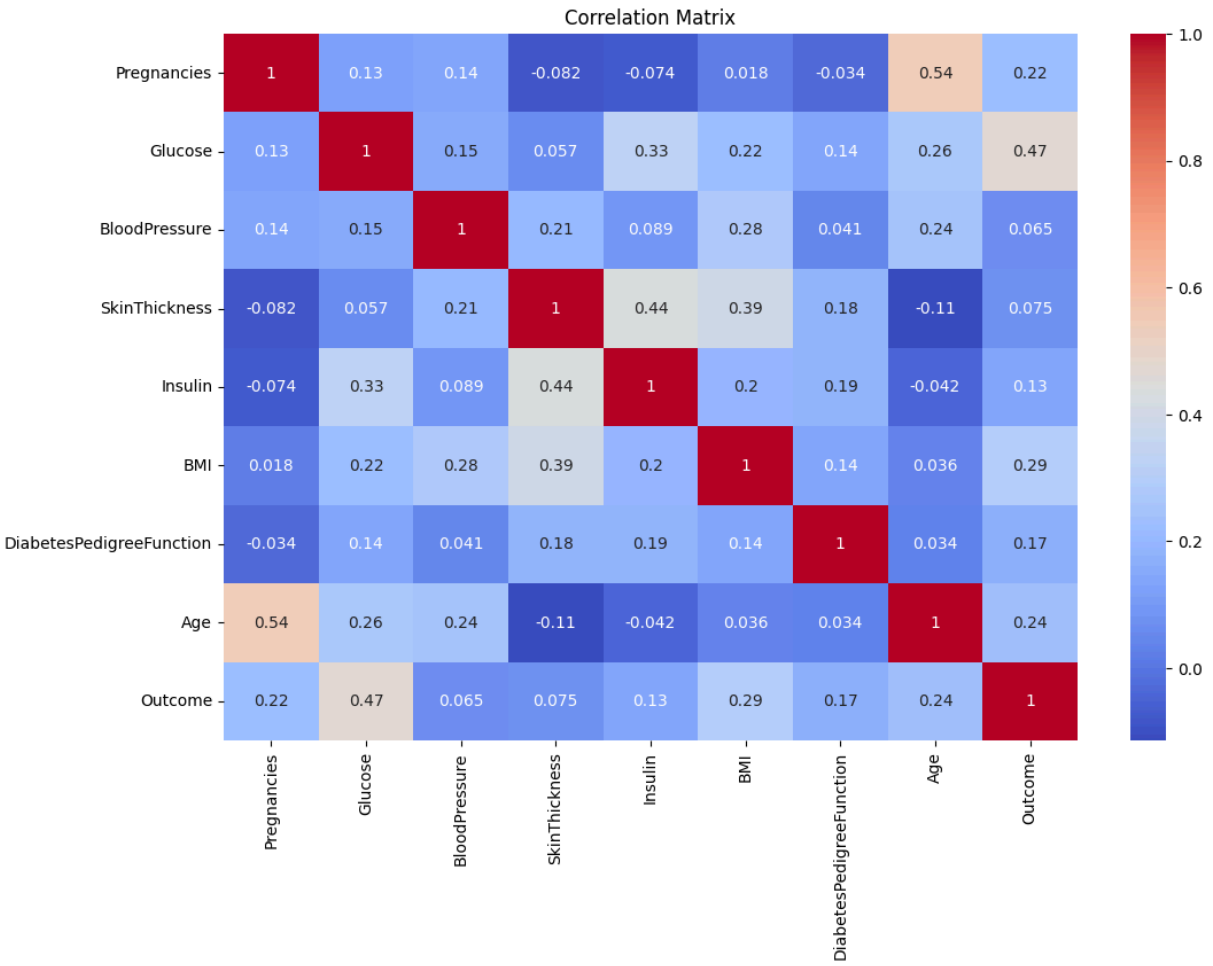
Correlation matrix plus creating a heatmap

The values ranges from -1 to 1: +1 shows perfect correlation.As one feature increases,the other features increase.

0 means No correlation

-1: Perfect Negative correlation. AS one feature increases, the other feature decreases

In [11]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



In [12]:
```
X = df.drop(columns='Outcome',axis=1)
y = df['Outcome']
```

In [13]:
```
y
```

Out[13]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Reasons for data standardization

1.Consistent scale - some columns have different scales.It brings all features in the same scale with a mean of 0 and a standard deviation of 1. 2.Better performance 3.Preventing Numerical Instability

Data Standardization

In [14]:
```
scaler = StandardScaler()
```

In [15]:
```
scaler.fit(X)
```

Out[15]:
```
StandardScaler
StandardScaler()
```

In [16]:
```
standardized_data = scaler.transform(X)
```

In [17]:
```
standardized_data
```

Out[17]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.47378505, -0.87137393]])
```

In [18]:
```
X = standardized_data
```

In [19]:
```
X
```

Out[19]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.47378505, -0.87137393]])
```

In [20]:
```
y
```

Out[20]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Splitting the data into training and testing sets

In [21]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

Printing the shape of training set

In [22]:
```
X_train.shape
```

Out[22]:
```
(614, 8)
```

Printing the shape of testing sets

In [23]:
```
X_test.shape
```

Out[23]:
```
(154, 8)
```

Train the model

Train the SVM model with a linear kernel

In [24]:
```
#linear kernel is faster to train than non-linear
clf = svm.SVC(kernel='linear')
```

In [25]:
```
clf.fit(X_train,y_train)
```

Out[25]:
```
SVC
SVC(kernel='linear')
```

Accuracy on train data

In [26]:
```
X_train_prediction = clf.predict(X_train)
accuracy_score(X_train_prediction,y_train)
```

Out[26]:
```
0.7719869706840391
```

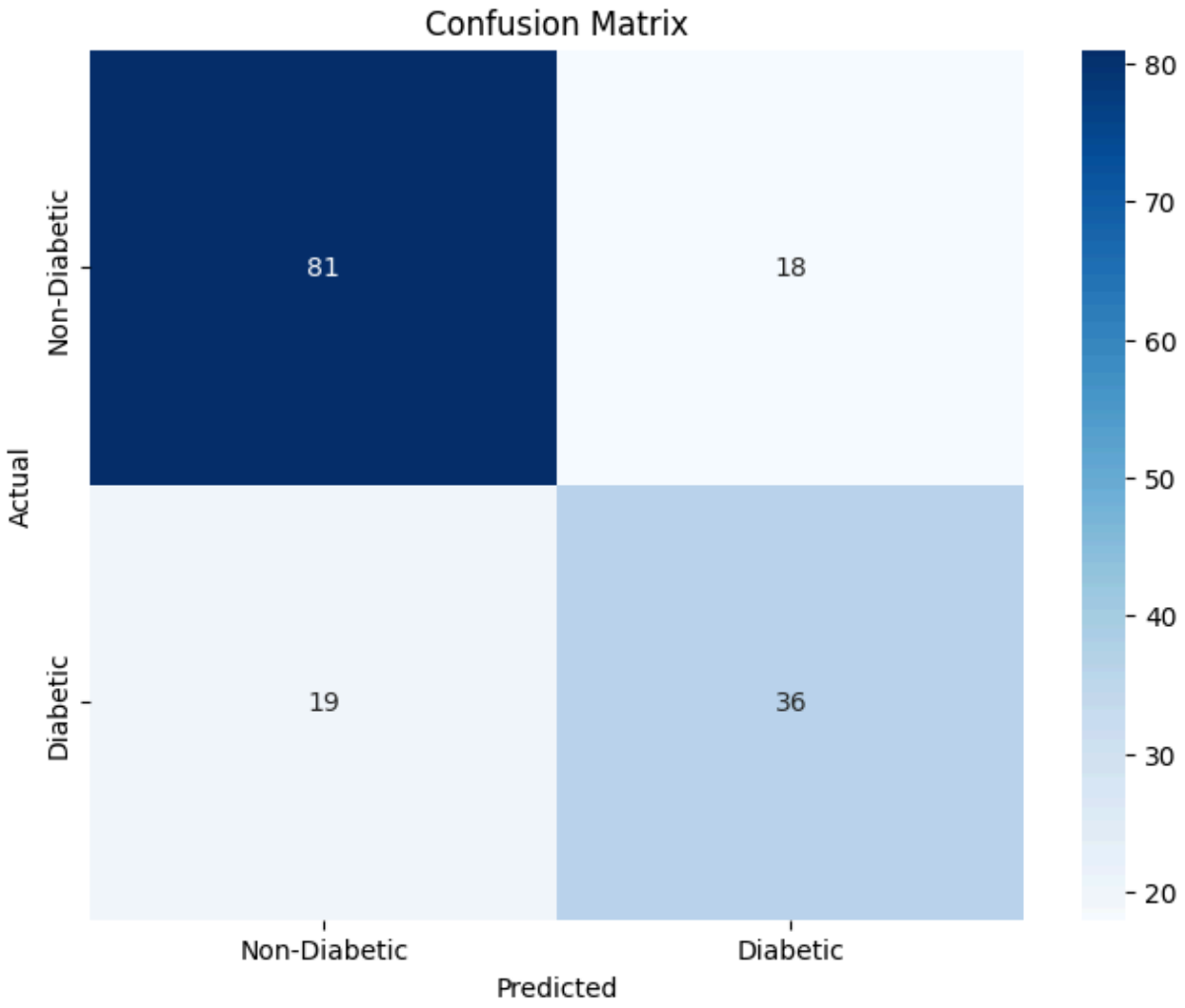
Accuracy on test data

In [27]:
```
X_test_prediction = clf.predict(X_test)
accuracy_score(X_test_prediction,y_test)
```

Out[27]:
```
0.7597402597402597
```

Confusion matrix (Visualizes the performance classification model)

In [28]:
```
conf_matrix = confusion_matrix(y_test, X_test_prediction)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Diabe
plt.title("Confusion Matrix")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Example input for prediction

In [ ]:
```
input_sample = (5,166,72,19,175,22.7,0.6,51)
```

In [ ]:
```
input_np_array = np.asarray(input_sample)
```

In [ ]:
```
input_np_arry_reshaped = input_np_array.reshape(1,-1)
```

In [ ]:
```
std_data = scaler.transform(input_np_arry_reshaped)
```

In [ ]:
```
std_data
```

Predict the outcome

In [ ]:
```
prediction = clf.predict(std_data)
```

In [ ]:
```
prediction
```

Interpreting the prediction

In [ ]:
```
if(prediction[0] ==0):
    print("Person is not diabetic")
else:
    print("Person is  diabetic")
```

1. Title Diabetes Prediction

2. Description This is a diabetes dataset.It has 769 rows and 9 columns Problem Statement Diabetes is a chronic disease. It results to an economic burden to those affected. My aim is to come up with a model which will help in predicting diabetes. The ability to predict a persons chances of getting diabetes can help people know their chances, early detection and health providers can advise on a healthy lifestyle.

Column Descriptions: Pregnancies: Number of times pregnant. Glucose: Plasma glucose concentration. BloodPressure: Diastolic blood pressure (mm Hg). SkinThickness: Triceps skin fold thickness (mm). Insulin: 2-Hour serum insulin (mu U/ml). BMI: Body mass index (weight in kg/(height in m)^2). DiabetesPedigreeFunction: Diabetes pedigree function. Age: Age (years). Outcome: Class variable (0 or 1, where 1 indicates the presence of diabetes)

The data contains a mix of integer and float types.

3.Why the choice of data It has the right columns and data to help me predict whether a person has diabetes or not. This is a diabetes dataset. It has 769 rows and 9 columns

4.Data Preparation I downloaded the data, it had no missing values

5.Training I used the following models for training. SVM (Support Vector Model) Its a machine learning algorithm that uses supervised learning model to solve complex classification ,regression and outlier detection. In our case, the diabetes dataset is supervised.

6.Results Accuracy on train results = 77% Accuracy on test results = 76%

7.Importance of the results Through the results, we get to know if the person is diabetic or not If 1 then you are diabetic If 0 then you are not diabetic The model needs improvement since its health data. There are several ways of improving the models performance one being combining different models(hybrid)

In [ ]:
```
```