In [1]:
```
import zipfile

!unzip /content/diabetes_project.zip
```

```
Archive:  /content/diabetes_project.zip
  inflating: diabetes.csv
```

In [2]:
```
import pandas as pd
import numpy as np

Diabetes_df = pd.read_csv("/content/diabetes.csv")

Diabetes_df.head()
```

Out[2]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0

5 rows × 22 columns

In [3]:
```
Diabetes_df.isnull().sum()
```

Out[3]:
```
Diabetes_binary         0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
```

In [4]:
```
print(Diabetes_df.shape)
```

```
(253680, 22)
```

In [5]:
```
display(Diabetes_df)
```

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActi
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	

2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
253675	0.0	1.0	1.0	1.0	45.0	0.0	0.0	0.0
253676	1.0	1.0	1.0	1.0	18.0	0.0	0.0	0.0
253677	0.0	0.0	0.0	1.0	28.0	0.0	0.0	0.0
253678	0.0	1.0	0.0	1.0	23.0	0.0	0.0	0.0
253679	1.0	1.0	1.0	1.0	25.0	0.0	0.0	1.0

253680 rows × 22 columns

In [6]:
```
dbts = 'Diabetes_binary'

yes = Diabetes_df[dbts].value_counts()
no = Diabetes_df[dbts].value_counts()

print(no[0], " people in this survey don't have diabetes")
print(yes[1], " people in this survey have diabetes ")
```

```
218334  people in this survey don't have diabetes
35346  people in this survey have diabetes
```

In [7]:
```
duplicateRows = Diabetes_df.duplicated()
duplicatesTotal = duplicateRows.sum()

print("There are ", duplicatesTotal, " duplicate rows")
```

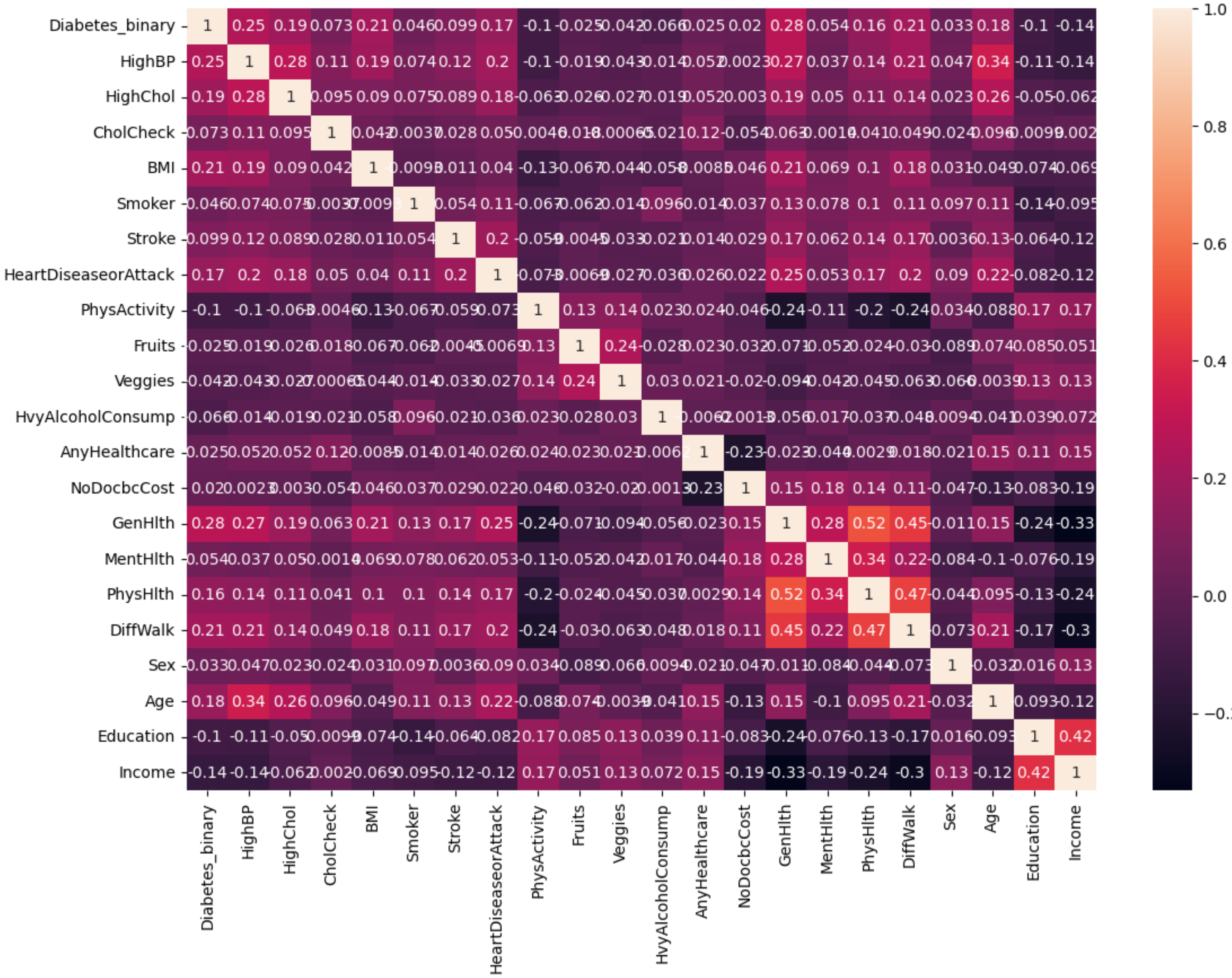
```
There are  24206  duplicate rows
```

In [8]:
```
Diabetes_df = Diabetes_df.drop_duplicates()
```

In [9]:
```
display(Diabetes_df.shape) # After removing duplicate rows,
                           #the number of entries have decreased.
```

```
(229474, 22)
```

In [10]:
```
import seaborn as sn
import matplotlib.pyplot as plt
#Correlation matrix plotted
Correlation = Diabetes_df.corr()
plt.figure(figsize=(13, 9))
sn.heatmap(Correlation, annot=True)
plt.show()
```

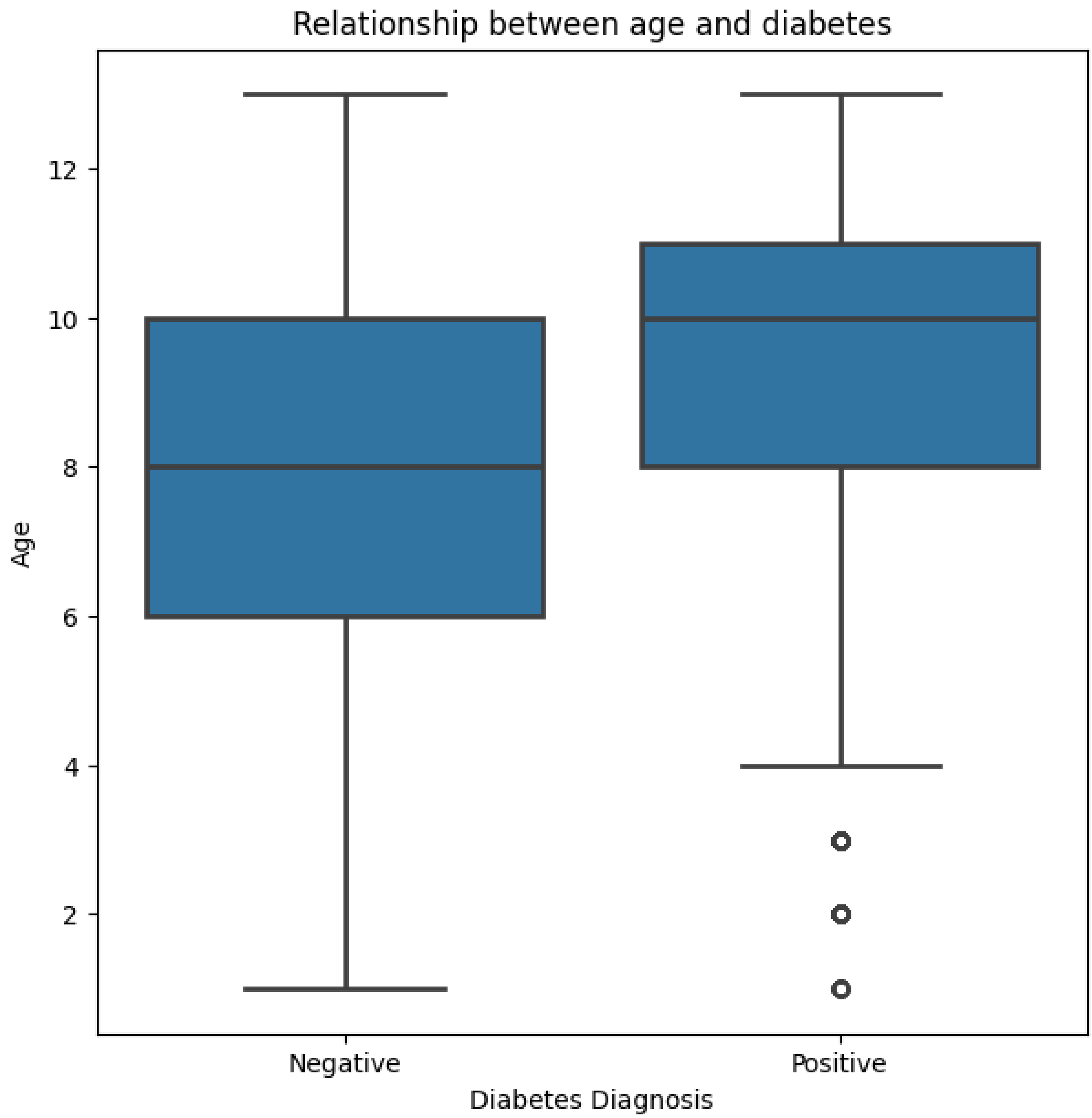


As we can see, after dropping the duplicate rows, we have a better ratio of yes and no diabetes.

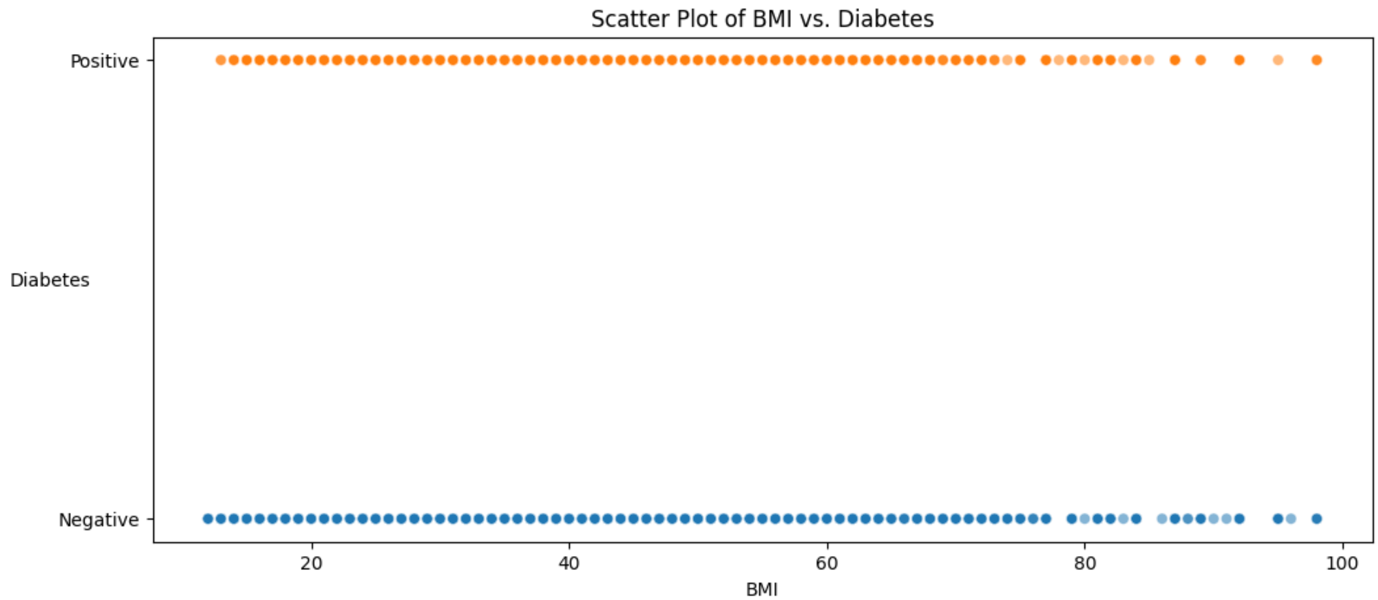
In [11]:
```
dbts = 'Diabetes_binary'

yes = Diabetes_df[dbts].value_counts()
no = Diabetes_df[dbts].value_counts()

print(no[0], " people in this survey don't have diabetes")
print(yes[1], " people in this survey have diabetes ")
```

```
194377  people in this survey don't have diabetes
35097  people in this survey have diabetes
```

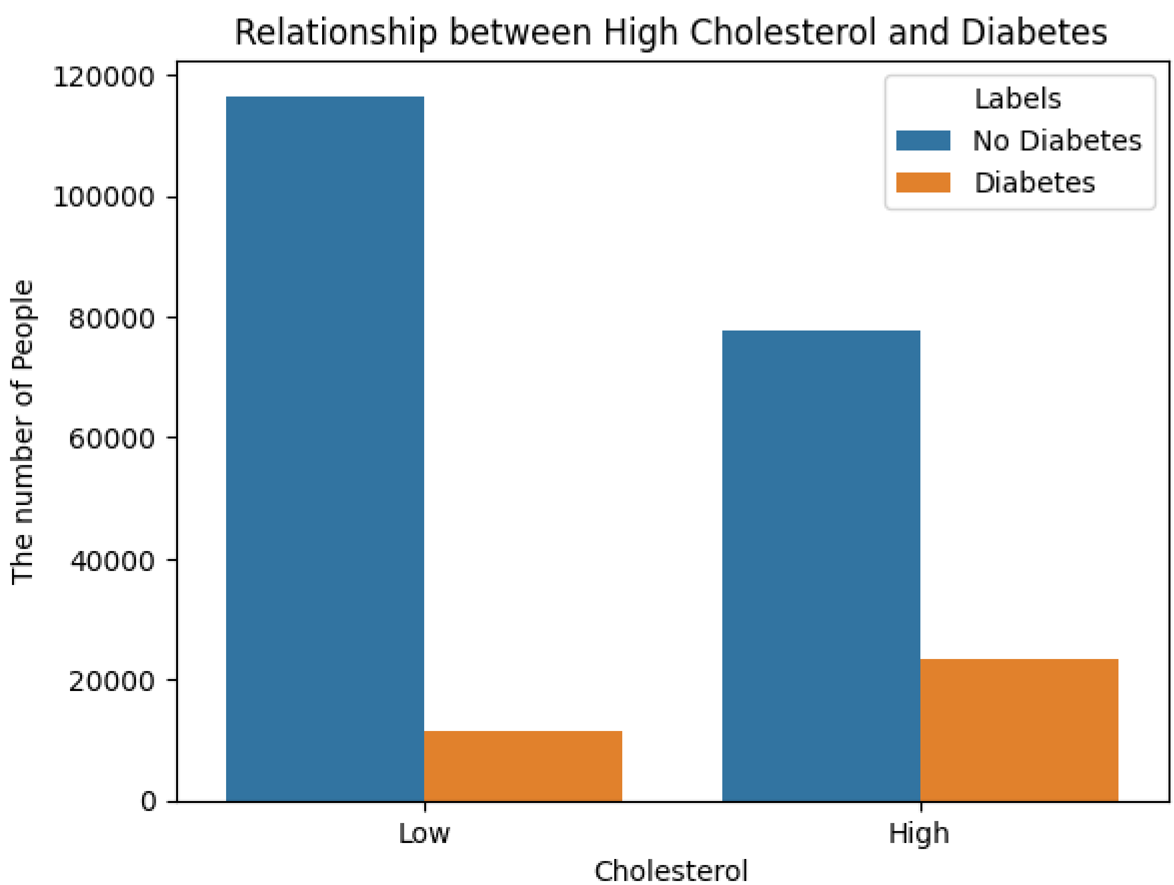
In [12]:
```
#Boxplot to investigate the relationship between diabetes and age
plt.figure(figsize=(7, 7))
sn.boxplot(x='Diabetes_binary', y='Age', data=Diabetes_df, linewidth = 2)
plt.title('Relationship between age and diabetes')
plt.xlabel('Diabetes Diagnosis')
plt.xticks(ticks=[0.0, 1.0], labels=['Negative', 'Positive'])
plt.show()
#Please note teh age values on the y axis here represent a category, not the actual age.
```



In [13]:
```
#A scatter plot demonstrating the link between the prevalance of diabetes and BMI.
plt.figure(figsize=(12, 5))
sn.scatterplot(x='BMI', y='Diabetes_binary', data=Diabetes_df, hue='Diabetes_binary', al
plt.title('Scatter Plot of BMI vs. Diabetes')
plt.xlabel('BMI'),plt.ylabel('Diabetes', rotation =0)
plt.yticks(ticks=[0.0, 1.0], labels=['Negative', 'Positive'])
plt.show()
```

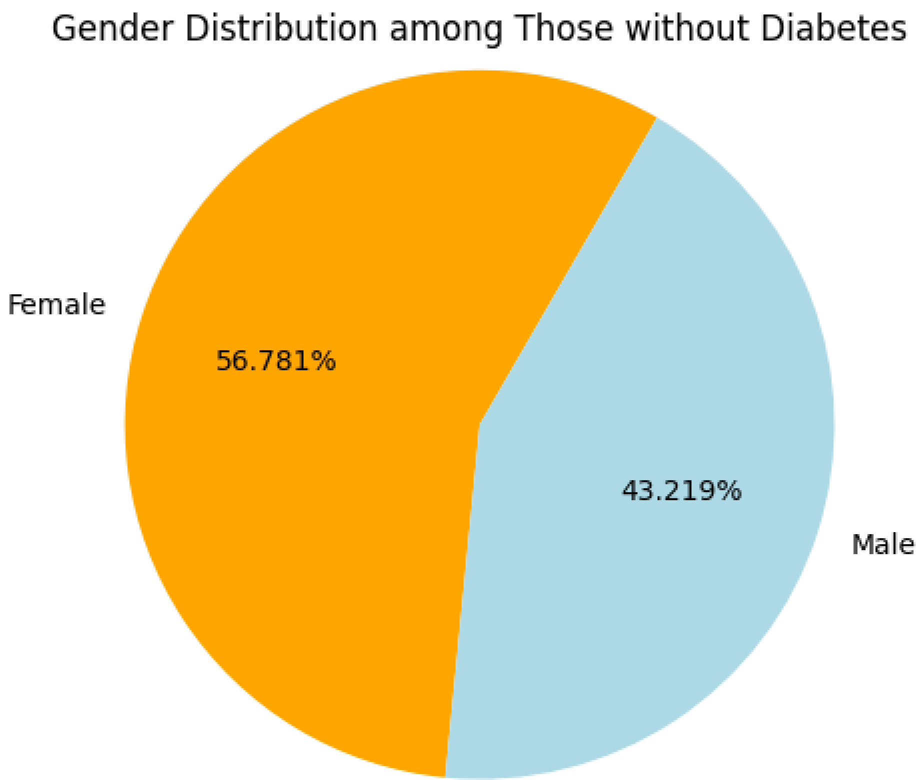


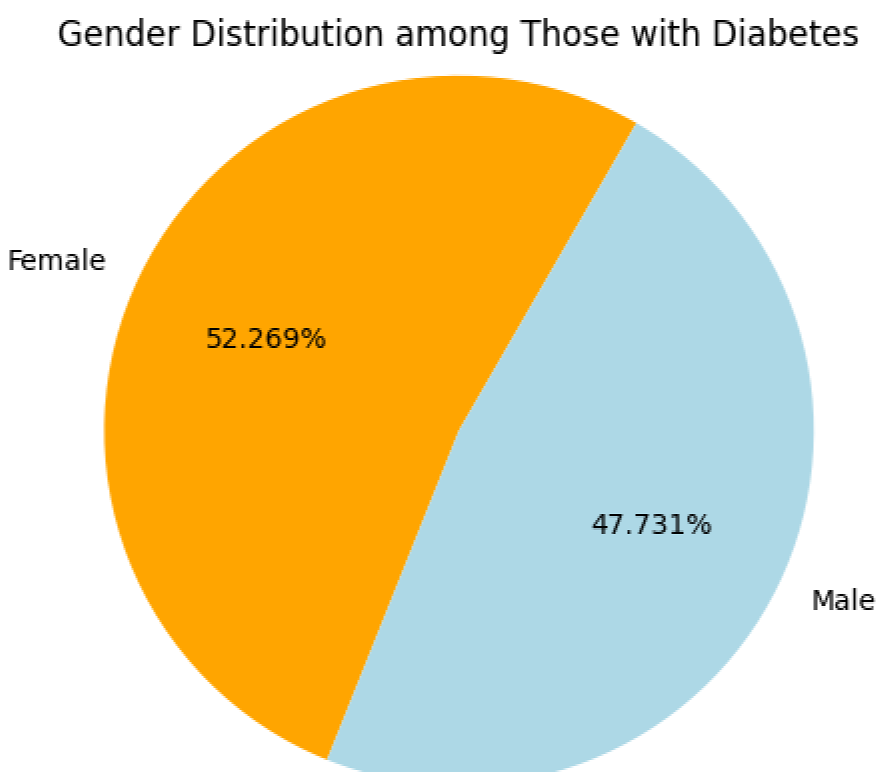
In [14]:
```
# A graph to investigate the relationship between high cholesterol and the occurence of
sn.countplot(x='HighChol', hue='Diabetes_binary', data=Diabetes_df)
plt.title('Relationship between High Cholesterol and Diabetes')
plt.xlabel('Cholesterol')
plt.xticks(ticks=[0.0, 1.0], labels=['Low', 'High'])
plt.ylabel('The number of People')
plt.legend(labels=['No Diabetes', 'Diabetes'] , title = 'Labels')
plt.show()
```



In [15]:
```
distributionGender = Diabetes_df.groupby(['Diabetes_binary', 'Sex']).size().reset_index(

labelGender = ['Female', 'Male']

# Pie chart plotted for people without diabetes
plt.pie(distributionGender[distributionGender['Diabetes_binary'] == 0]['Distribution'],
        labels=labelGender,startangle=60 , autopct='%1.3f%%', radius = 1.2, colors=['ora
```

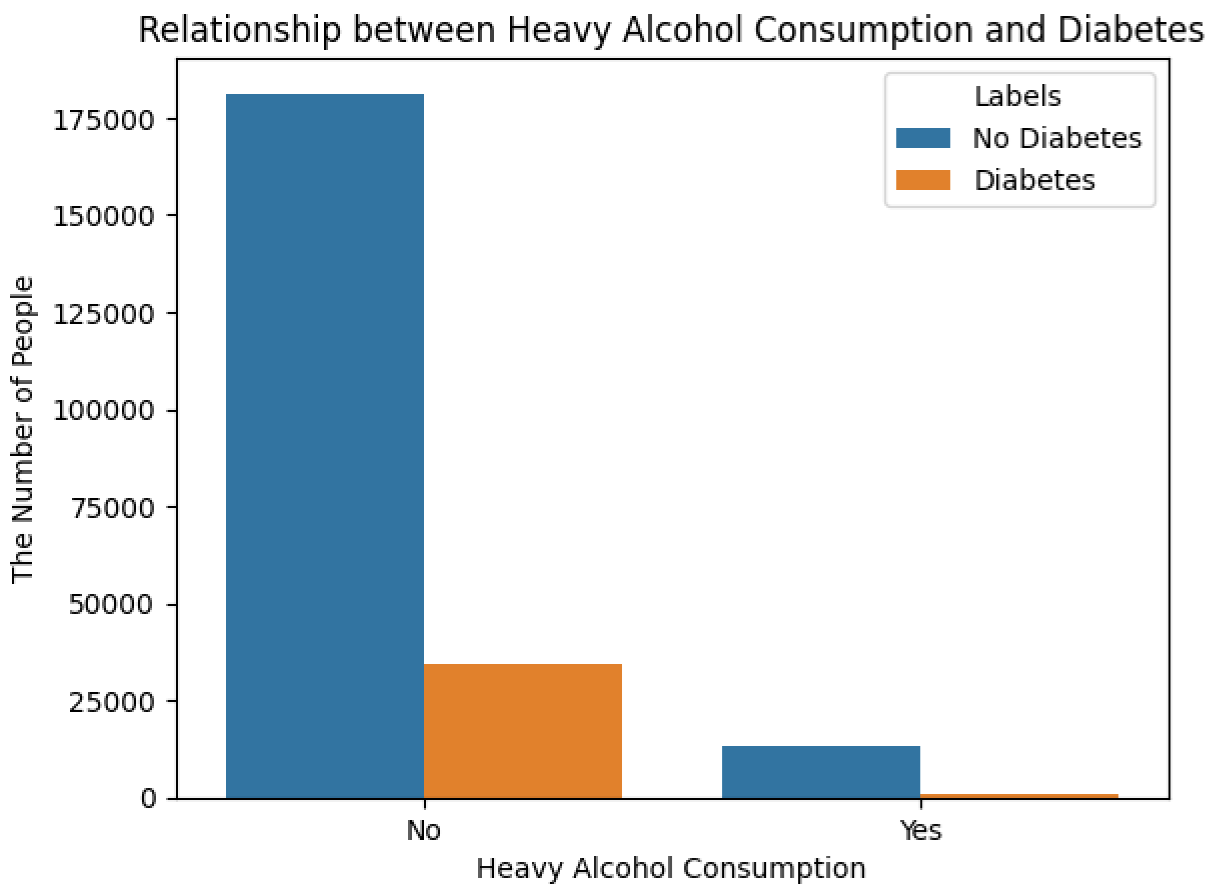
```
plt.title('Gender Distribution among Those without Diabetes')

plt.show()

# Pie chart plotted for people with diabetes
plt.pie(distributionGender[distributionGender['Diabetes_binary'] == 1]['Distribution'],
        labels=labelGender,startangle=60, autopct='%1.3f%%', radius = 1.2, colors=['oran
plt.title('Gender Distribution among Those with Diabetes')

plt.show()
```





In [16]:
```
# A graph to investigate the relationship between high cholesterol and the occurence of
sn.countplot(x='HvyAlcoholConsump', hue='Diabetes_binary', data=Diabetes_df)
```

```
plt.title('Relationship between Heavy Alcohol Consumption and Diabetes')
plt.xlabel('Heavy Alcohol Consumption')
plt.xticks(ticks=[1.0, 0.0], labels=['Yes', 'No'])
plt.ylabel('The Number of People')
plt.legend(labels=['No Diabetes', 'Diabetes'] , title = 'Labels')
plt.show()
```



In [17]:
```
display(Diabetes_df.shape)
```

(229474, 22)

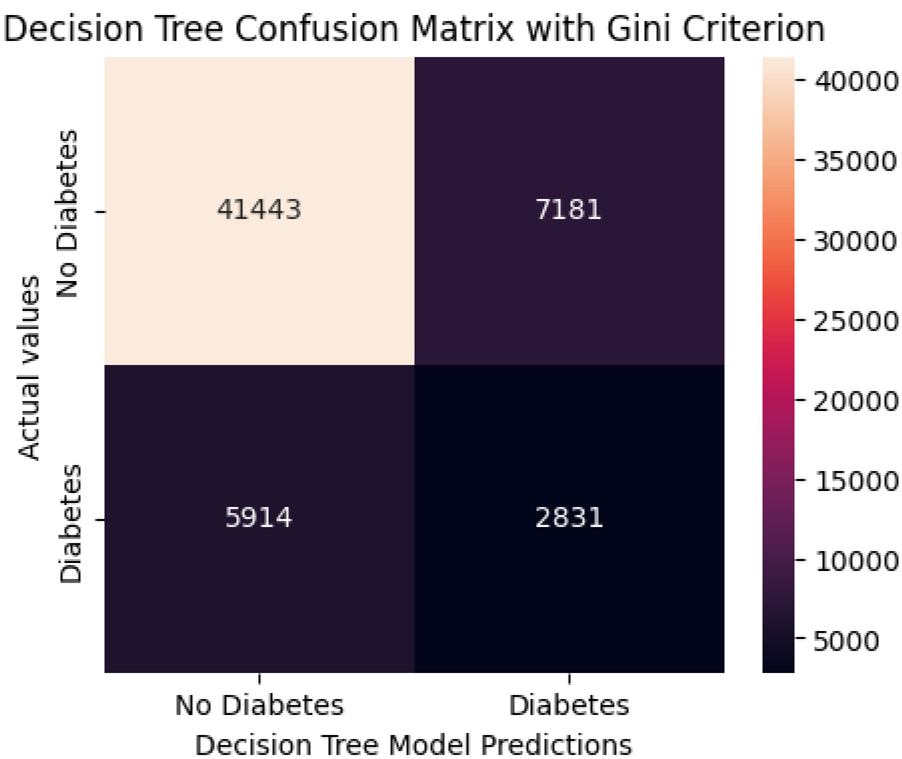
In [18]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,accuracy_score,f1_score,precision_score,rec
from sklearn.metrics import ConfusionMatrixDisplay
```

In [19]:
```
#Decision Tree Classifier

t = Diabetes_df['Diabetes_binary'] #Target Column/variable

f = Diabetes_df.drop('Diabetes_binary', axis=1) #Feature variable

F_train, F_test, T_train, T_test = train_test_split(f, t, test_size=0.25, random_state=

DTmodel = DecisionTreeClassifier(random_state=22) #Model for decision tree initialised.

DTmodel= DTmodel.fit(F_train,T_train) # Training the model on data.
```

In [20]:
```
Gini_pred = DTmodel.predict(F_test) #Prediction variable established.
print(Gini_pred)

# Using multiple metrics for assessing the results.
DTaccuracy = accuracy_score(T_test, Gini_pred)
DTconfusion = confusion_matrix(T_test, Gini_pred)
DTrecall= recall_score(T_test, Gini_pred)
DTf1= f1_score(T_test, Gini_pred)
```

```
DTprecision= precision_score(T_test, Gini_pred)

#Results are printed.
print('Accuracy score: ', DTaccuracy)
print('Confusion Matrix: ', DTconfusion)
print('Recall score: ', DTrecall)
print('F1 score score: ', DTf1)
print('Precision score: ', DTprecision)

plt.figure(figsize=(5, 4))
sn.heatmap(DTconfusion, annot=True, fmt="d",xticklabels=["No Diabetes", "Diabetes"],ytic
plt.title('Decision Tree Confusion Matrix with Gini Criterion') # Title defined
plt.xlabel('Decision Tree Model Predictions')
plt.ylabel('Actual values')
plt.show()
```

```
[0. 0. 0. ... 0. 0. 0.]
Accuracy score:  0.771740835642943
Confusion Matrix:  [[41443  7181]
 [ 5914  2831]]
Recall score:  0.3237278444825615
F1 score score:  0.3018606386948873
Precision score:  0.28276068717538955
```

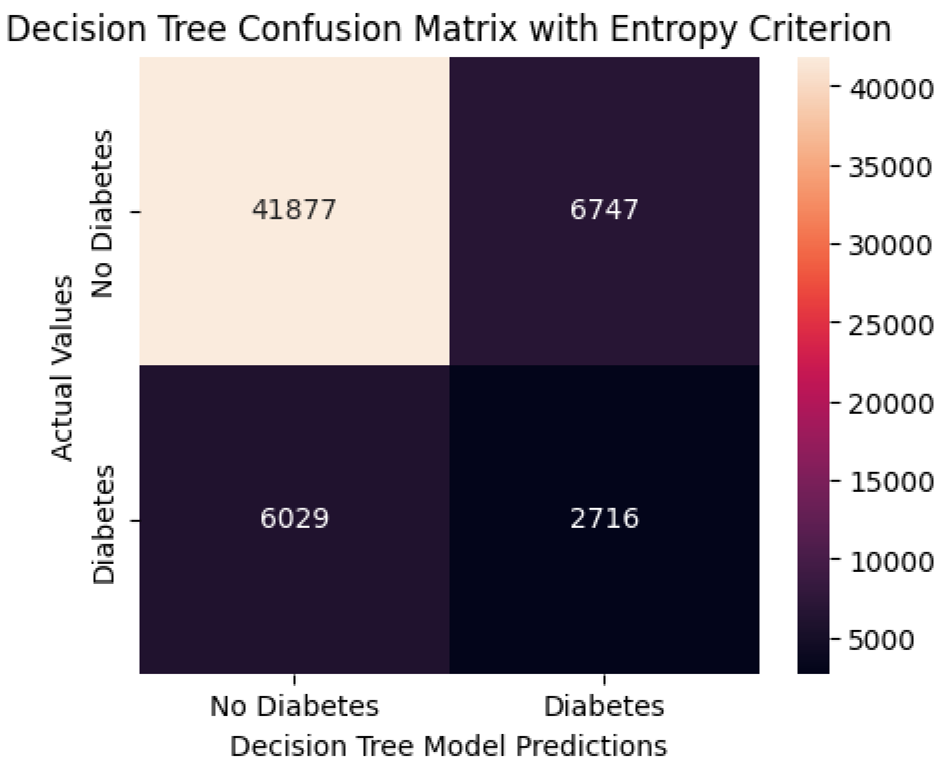


In [21]:

```
# Decision Tree model, this time with entropy criterion to assess their difference.
DTmodel_entropy = DecisionTreeClassifier(criterion='entropy', random_state=22)

# Model is being trained
DTmodel_entropy.fit(F_train, T_train)


Entropy_pred = DTmodel_entropy.predict(F_test)
print(Entropy_pred)

# The results are evaluated using a variety of metrics.
DT_accuracy_entropy = accuracy_score(T_test, Entropy_pred)
DT_confusion_entropy = confusion_matrix(T_test, Entropy_pred)
DT_recall_entropy = recall_score(T_test, Entropy_pred)
DT_f1_entropy = f1_score(T_test, Entropy_pred)
DT_precision_entropy = precision_score(T_test, Entropy_pred)
```

```
print('Accuracy score:', DT_accuracy_entropy)
print('Confusion Matrix:\n', DT_confusion_entropy)
print('Recall score:', DT_recall_entropy)
print('F1 score:', DT_f1_entropy)
print('Precision score:', DT_precision_entropy)

plt.figure(figsize=(5, 4))
sn.heatmap(DT_confusion_entropy, annot=True, fmt="d", xticklabels=["No Diabetes", "Diabe
plt.title('Decision Tree Confusion Matrix with Entropy Criterion') # Title defined
plt.xlabel('Decision Tree Model Predictions')
plt.ylabel('Actual Values')
plt.show()
```

```
[0. 0. 0. ... 1. 0. 1.]
Accuracy score: 0.7773013299865781
Confusion Matrix:
 [[41877  6747]
 [ 6029  2716]]
Recall score: 0.3105774728416238
F1 score: 0.29833040421792617
Precision score: 0.28701257529324736
```



In [22]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [23]:
```
TFmodel = keras.Sequential([                    # The model is initialised, as the data is
    keras.layers.Dense(units = 256, activation='relu'),#specifically 5 layers. Initial l
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(units = 1, activation='sigmoid') #As this is a classification pro
])

  #The model is compiled.
TFmodel.compile(loss='binary_crossentropy', # Binary classification so binary cross entr
                metrics=[keras.metrics.BinaryAccuracy(name = "accuracy"),   #  Various m
                         keras.metrics.Recall(name = "recall"),
                         keras.metrics.Precision(name = "precision"),
                         keras.metrics.F1Score(name = "f1 score" )])
```

```
TFmodel.fit(F_train, T_train, validation_split=0.25, batch_size=128, epochs=25) # 25% of


metrics = TFmodel.evaluate(F_test, T_test) #The model is evaluated using the data availa

print("\n")
print("\n") #The results are printed.
print(f'The results representing loss, accuracy, recall, precision, and F1 values are as
```

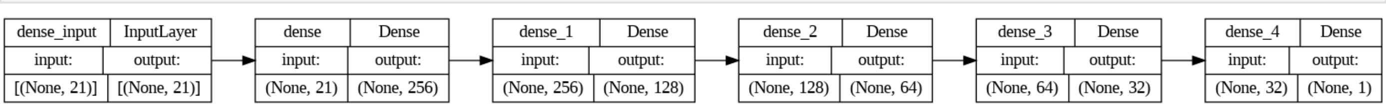
```
Epoch 1/25
1009/1009 [==============================] - 7s 5ms/step - loss: 0.3775 - accuracy: 0.84
66 - recall: 0.0652 - precision: 0.4858 - f1 score: 0.2651 - val_loss: 0.3947 - val_accu
racy: 0.8474 - val_recall: 0.0143 - val_precision: 0.7308 - val_f1 score: 0.2669
Epoch 2/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3567 - accuracy: 0.84
98 - recall: 0.0904 - precision: 0.5522 - f1 score: 0.2651 - val_loss: 0.3565 - val_accu
racy: 0.8494 - val_recall: 0.0714 - val_precision: 0.5935 - val_f1 score: 0.2669
Epoch 3/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3519 - accuracy: 0.85
03 - recall: 0.1003 - precision: 0.5564 - f1 score: 0.2651 - val_loss: 0.3543 - val_accu
racy: 0.8498 - val_recall: 0.2008 - val_precision: 0.5326 - val_f1 score: 0.2669
Epoch 4/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3492 - accuracy: 0.85
10 - recall: 0.1108 - precision: 0.5630 - f1 score: 0.2651 - val_loss: 0.3498 - val_accu
racy: 0.8500 - val_recall: 0.0602 - val_precision: 0.6394 - val_f1 score: 0.2669
Epoch 5/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3477 - accuracy: 0.85
14 - recall: 0.1175 - precision: 0.5658 - f1 score: 0.2651 - val_loss: 0.3454 - val_accu
racy: 0.8515 - val_recall: 0.0943 - val_precision: 0.6176 - val_f1 score: 0.2669
Epoch 6/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3469 - accuracy: 0.85
12 - recall: 0.1198 - precision: 0.5614 - f1 score: 0.2651 - val_loss: 0.3759 - val_accu
racy: 0.8290 - val_recall: 0.4317 - val_precision: 0.4435 - val_f1 score: 0.2669
Epoch 7/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3459 - accuracy: 0.85
13 - recall: 0.1264 - precision: 0.5604 - f1 score: 0.2651 - val_loss: 0.3478 - val_accu
racy: 0.8503 - val_recall: 0.2543 - val_precision: 0.5289 - val_f1 score: 0.2669
Epoch 8/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3447 - accuracy: 0.85
18 - recall: 0.1294 - precision: 0.5665 - f1 score: 0.2651 - val_loss: 0.3436 - val_accu
racy: 0.8534 - val_recall: 0.1326 - val_precision: 0.6117 - val_f1 score: 0.2669
Epoch 9/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3447 - accuracy: 0.85
18 - recall: 0.1307 - precision: 0.5652 - f1 score: 0.2651 - val_loss: 0.3439 - val_accu
racy: 0.8510 - val_recall: 0.0874 - val_precision: 0.6160 - val_f1 score: 0.2669
Epoch 10/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3441 - accuracy: 0.85
21 - recall: 0.1314 - precision: 0.5691 - f1 score: 0.2651 - val_loss: 0.3423 - val_accu
racy: 0.8533 - val_recall: 0.1548 - val_precision: 0.5900 - val_f1 score: 0.2669
Epoch 11/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3438 - accuracy: 0.85
20 - recall: 0.1357 - precision: 0.5654 - f1 score: 0.2651 - val_loss: 0.3443 - val_accu
racy: 0.8513 - val_recall: 0.0742 - val_precision: 0.6525 - val_f1 score: 0.2669
Epoch 12/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3433 - accuracy: 0.85
19 - recall: 0.1318 - precision: 0.5666 - f1 score: 0.2651 - val_loss: 0.3632 - val_accu
racy: 0.8504 - val_recall: 0.0573 - val_precision: 0.6655 - val_f1 score: 0.2669
Epoch 13/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3435 - accuracy: 0.85
27 - recall: 0.1362 - precision: 0.5759 - f1 score: 0.2651 - val_loss: 0.3469 - val_accu
racy: 0.8529 - val_recall: 0.2016 - val_precision: 0.5628 - val_f1 score: 0.2669
Epoch 14/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3432 - accuracy: 0.85
18 - recall: 0.1330 - precision: 0.5645 - f1 score: 0.2651 - val_loss: 0.3484 - val_accu
racy: 0.8520 - val_recall: 0.2411 - val_precision: 0.5439 - val_f1 score: 0.2669
Epoch 15/25
```

```
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3430 - accuracy: 0.85
23 - recall: 0.1369 - precision: 0.5702 - f1 score: 0.2651 - val_loss: 0.3467 - val_accu
racy: 0.8500 - val_recall: 0.0564 - val_precision: 0.6527 - val_f1 score: 0.2669
Epoch 16/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3430 - accuracy: 0.85
25 - recall: 0.1367 - precision: 0.5730 - f1 score: 0.2651 - val_loss: 0.3447 - val_accu
racy: 0.8523 - val_recall: 0.2266 - val_precision: 0.5500 - val_f1 score: 0.2669
Epoch 17/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3427 - accuracy: 0.85
25 - recall: 0.1385 - precision: 0.5713 - f1 score: 0.2651 - val_loss: 0.3449 - val_accu
racy: 0.8533 - val_recall: 0.1501 - val_precision: 0.5933 - val_f1 score: 0.2669
Epoch 18/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3426 - accuracy: 0.85
25 - recall: 0.1403 - precision: 0.5713 - f1 score: 0.2651 - val_loss: 0.3416 - val_accu
racy: 0.8529 - val_recall: 0.1645 - val_precision: 0.5795 - val_f1 score: 0.2669
Epoch 19/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3427 - accuracy: 0.85
22 - recall: 0.1417 - precision: 0.5658 - f1 score: 0.2651 - val_loss: 0.3455 - val_accu
racy: 0.8532 - val_recall: 0.1915 - val_precision: 0.5701 - val_f1 score: 0.2669
Epoch 20/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3425 - accuracy: 0.85
25 - recall: 0.1422 - precision: 0.5701 - f1 score: 0.2651 - val_loss: 0.3421 - val_accu
racy: 0.8527 - val_recall: 0.1121 - val_precision: 0.6212 - val_f1 score: 0.2669
Epoch 21/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3423 - accuracy: 0.85
30 - recall: 0.1435 - precision: 0.5760 - f1 score: 0.2651 - val_loss: 0.3703 - val_accu
racy: 0.8490 - val_recall: 0.0421 - val_precision: 0.6488 - val_f1 score: 0.2670
Epoch 22/25
1009/1009 [==============================] - 6s 6ms/step - loss: 0.3422 - accuracy: 0.85
34 - recall: 0.1440 - precision: 0.5815 - f1 score: 0.2651 - val_loss: 0.3414 - val_accu
racy: 0.8528 - val_recall: 0.1482 - val_precision: 0.5884 - val_f1 score: 0.2669
Epoch 23/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3419 - accuracy: 0.85
25 - recall: 0.1440 - precision: 0.5688 - f1 score: 0.2651 - val_loss: 0.3451 - val_accu
racy: 0.8502 - val_recall: 0.2663 - val_precision: 0.5269 - val_f1 score: 0.2669
Epoch 24/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3420 - accuracy: 0.85
27 - recall: 0.1413 - precision: 0.5732 - f1 score: 0.2651 - val_loss: 0.3426 - val_accu
racy: 0.8529 - val_recall: 0.1818 - val_precision: 0.5700 - val_f1 score: 0.2669
Epoch 25/25
1009/1009 [==============================] - 6s 6ms/step - loss: 0.3417 - accuracy: 0.85
33 - recall: 0.1468 - precision: 0.5791 - f1 score: 0.2651 - val_loss: 0.3437 - val_accu
racy: 0.8533 - val_recall: 0.1848 - val_precision: 0.5735 - val_f1 score: 0.2669
1793/1793 [==============================] - 3s 2ms/step - loss: 0.3429 - accuracy: 0.85
32 - recall: 0.1745 - precision: 0.5594 - f1 score: 0.2645
```

```
The results representing loss, accuracy, recall, precision, and F1 values are as follow
s: [0.3428666293621063, 0.8532134294509888, 0.1744997203350067, 0.5593841671943665, arra
y([0.26454306], dtype=float32)]
```

In [24]:
```python
from keras.utils import plot_model

# The model is plotted.
plot_model(TFmodel,show_shapes=True, show_layer_names=True, rankdir = 'LR')
```

Out[24]:



In [25]:
```python
from sklearn.naive_bayes import GaussianNB
```

In [26]:
```python
NBmodel = GaussianNB() #Naive Bayes model is initialised.
```

```
NBmodel.fit(F_train, T_train) # The model is trained.

NBpred = NBmodel.predict(F_test)

#The model is evaluated against a variety of metrics.
NBaccuracy = accuracy_score(T_test, NBpred)
NBprecision = precision_score(T_test, NBpred)
NBrecall = recall_score(T_test, NBpred)
NBf1_score = f1_score(T_test, NBpred)
NBconfusion = confusion_matrix(T_test, NBpred)

print("Naive Bayes Classifier:")
print('Accuracy score: ', NBaccuracy)
print('Confusion Matrix: ', NBconfusion)
print('Recall score: ', NBrecall)
print('F1 score score: ', NBf1_score)
print('Precision score: ', NBprecision)
```

```
Naive Bayes Classifier:
Accuracy score:  0.7576565741079677
Confusion Matrix:  [[38501 10123]
 [ 3780  4965]]
Recall score:  0.5677530017152659
F1 score score:  0.41664918390467
Precision score:  0.3290694591728526
```

In [27]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler #Standard Scaler imported to use on dat
```

In [28]:
```
#K-Neighbours Classification

SS = StandardScaler().fit(F_train) # Standard Scaler is initialsied and used for standar
SS_F_train = SS.transform(F_train)
SS_F_test = SS.transform(F_test)

KNNmodel = KNeighborsClassifier(n_neighbors=6) #Model is initialised, n_neighbours is 6,
                                              #a range of values from 3 to 479 were eval
#Standardised data is used to fit the model.
KNNmodel.fit(SS_F_train, T_train)


KNNpred = KNNmodel.predict(SS_F_test)

#The model is evaluated.
KNNaccuracy = accuracy_score(T_test, KNNpred)
KNNprecision = precision_score(T_test, KNNpred)
KNNrecall = recall_score(T_test, KNNpred)
KNNf1_score = f1_score(T_test, KNNpred)
KNNconfusion = confusion_matrix(T_test, KNNpred)

#The results are printed.
print("K-Neighbours Classifier with scaled data: ")
print('Accuracy score:', KNNaccuracy)
print('Confusion Matrix:', KNNconfusion)
print('Recall score:', KNNrecall)
print('F1 score:', KNNf1_score)
print('Precision score:', KNNprecision)
```

```
K-Neighbours Classifier with scaled data:
Accuracy score: 0.842772228904112
Confusion Matrix: [[47284  1340]
 [ 7680  1065]]
Recall score: 0.12178387650085763
F1 score: 0.19103139013452916
Precision score: 0.44282744282744285
```

In [29]:
```
#K Neighbours, this time without standardisation of data.
KNNmodel = KNeighborsClassifier(n_neighbors= 6)

KNNmodel.fit(F_train, T_train) #Model is fitted.


KNNpred = KNNmodel.predict(F_test)

#Metrics used for evaluation
KNNaccuracy = accuracy_score(T_test, KNNpred)
KNNprecision = precision_score(T_test, KNNpred)
KNNrecall = recall_score(T_test, KNNpred)
KNNf1_score = f1_score(T_test, KNNpred)
KNNconfusion = confusion_matrix(T_test, KNNpred)

#Results Printed
print("K-Neighbours Classifier without scaled data: ")
print('Accuracy score:', KNNaccuracy)
print('Confusion Matrix:', KNNconfusion)
print('Recall score:', KNNrecall)
print('F1 score:', KNNf1_score)
print('Precision score:', KNNprecision)
```

```
K-Neighbours Classifier without scaled data:
Accuracy score: 0.8434171765239067
Confusion Matrix: [[47411  1213]
 [ 7770   975]]
Recall score: 0.11149228130360206
F1 score: 0.17835909631391197
Precision score: 0.44561243144424134
```

In [30]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [31]:
```
#Random Forest Classification

# The model is initialised
RFmodel = RandomForestClassifier(random_state=22)

RFmodel.fit(F_train, T_train)


RFpred = RFmodel.predict(F_test)

#The model is evaluated.
RFaccuracy = accuracy_score(T_test, RFpred)
RFprecision = precision_score(T_test, RFpred)
RFrecall = recall_score(T_test, RFpred)
RFf1_score = f1_score(T_test, RFpred)
RFconfusion = confusion_matrix(T_test, RFpred)

#The results printed.
print("Random Forest with GINI criterion")
print('Accuracy score:', RFaccuracy)
print('Confusion Matrix:', RFconfusion)
print('Recall score:', RFrecall)
print('F1 score:', RFf1_score)
print('Precision score:', RFprecision)
```

```
Random Forest with GINI criterion
Accuracy score: 0.8445850546462376
Confusion Matrix: [[46950  1674]
 [ 7242  1503]]
Recall score: 0.17186963979416808
F1 score: 0.25213890286864615
Precision score: 0.4730878186968839
```

In [32]:
```
#Random Forest classification, this time with entropy criterion

RFmodel_entropy = RandomForestClassifier(criterion='entropy', random_state=22)


RFmodel_entropy.fit(F_train, T_train)


RFentropy_pred = RFmodel_entropy.predict(F_test)

RF_accuracy_entropy = accuracy_score(T_test, RFentropy_pred)
RF_confusion_entropy = confusion_matrix(T_test, RFentropy_pred)
RF_recall_entropy = recall_score(T_test, RFentropy_pred)
RF_f1_entropy = f1_score(T_test, RFentropy_pred)
RF_precision_entropy = precision_score(T_test, RFentropy_pred)

print("Random Forest with Entropy Criterion: ")
print('Accuracy score:', RF_accuracy_entropy)
print('Confusion Matrix:\n', RF_confusion_entropy)
print('Recall score:', RF_recall_entropy)
print('F1 score:', RF_f1_entropy)
print('Precision score:', RF_precision_entropy)
```

```
Random Forest with Entropy Criterion:
Accuracy score: 0.8443410204117207
Confusion Matrix:
 [[46991  1633]
 [ 7297  1448]]
Recall score: 0.16558033161806746
F1 score: 0.24488415355995263
Precision score: 0.46997728010386236
```

In [33]:
```
from imblearn.over_sampling import SMOTE
```

In [34]:
```
oversample = SMOTE(random_state=22) #SMOTE is initialised to equalise the dsitribution o

F_train_oversampled, T_train_oversampled = oversample.fit_resample(F_train, T_train)
```

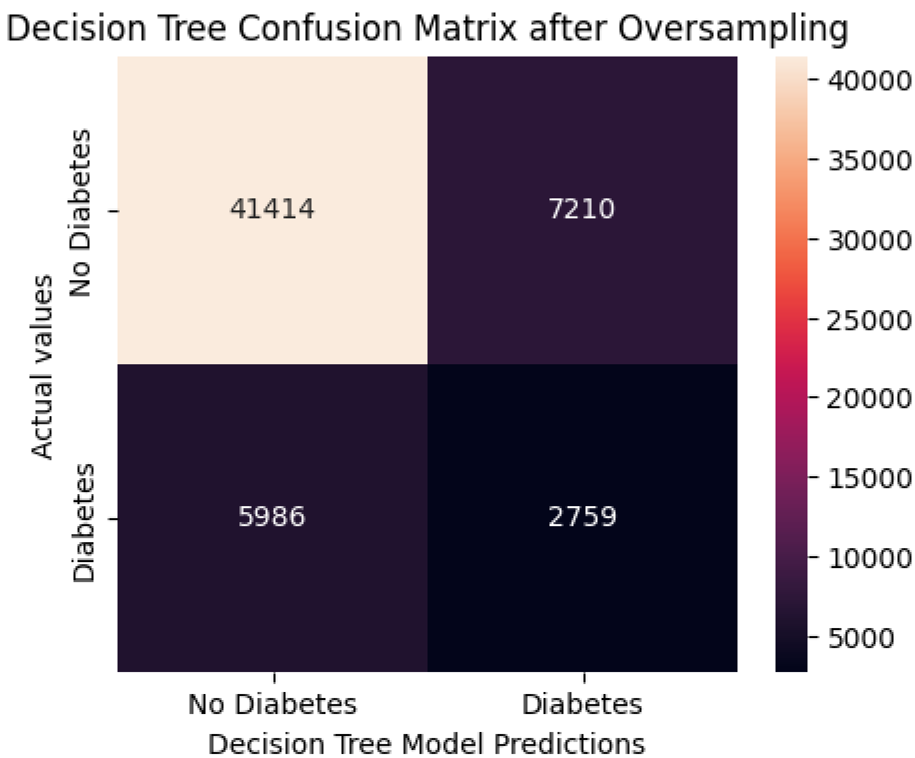
In [35]:
```
print("Class distribution before SMOTE:", T_train.value_counts()) #As you can see, the c
print("Class distribution after SMOTE:", T_train_oversampled.value_counts())
```

```
Class distribution before SMOTE: Diabetes_binary
0.0    145753
1.0     26352
Name: count, dtype: int64
Class distribution after SMOTE: Diabetes_binary
0.0    145753
1.0    145753
Name: count, dtype: int64
```

In [36]:
```
# Decision Tree Classifier after Oversampling
DTmodel_SMOTE = DecisionTreeClassifier(random_state=22) # Model for decision tree initia
DTmodel_SMOTE = DTmodel_SMOTE.fit(F_train_oversampled, T_train_oversampled) # Training t

DTpred_SMOTE = DTmodel_SMOTE.predict(F_test) # Prediction variable established.
print(DTpred_SMOTE)

# Using multiple metrics for assessing the results.
DTaccuracy_SMOTE = accuracy_score(T_test, DTpred_SMOTE)
DTconfusion_SMOTE = confusion_matrix(T_test, DTpred_SMOTE)
DTrecall_SMOTE = recall_score(T_test, DTpred_SMOTE)
DTf1_SMOTE = f1_score(T_test, DTpred_SMOTE)
DTprecision_SMOTE = precision_score(T_test, DTpred_SMOTE)

# Results are printed.
```

```
print('Accuracy score after Oversampling: ', DTaccuracy_SMOTE)
print('Confusion Matrix after Oversampling: ', DTconfusion_SMOTE)
print('Recall score after Oversampling: ', DTrecall_SMOTE)
print('F1 score after Oversampling: ', DTf1_SMOTE)
print('Precision score after Oversampling: ', DTprecision_SMOTE)

plt.figure(figsize=(5, 4))
sn.heatmap(DTconfusion_SMOTE, annot=True, fmt="d", xticklabels=["No Diabetes", "Diabetes
plt.title('Decision Tree Confusion Matrix after Oversampling') # Title defined
plt.xlabel('Decision Tree Model Predictions')
plt.ylabel('Actual values')
plt.show()
```

```
[0. 0. 0. ... 1. 0. 0.]
Accuracy score after Oversampling:  0.7699803029510711
Confusion Matrix after Oversampling:  [[41414  7210]
 [ 5986  2759]]
Recall score after Oversampling:  0.31549456832475703
F1 score after Oversampling:  0.2948594635032596
Precision score after Oversampling:  0.27675794964389605
```



In [37]:

```
# Deep learning model, this time with oversampled training data.

TFmodel_SMOTE = keras.Sequential([                    # The model is initialised, as the d
    keras.layers.Dense(units = 256, activation='relu'),#specifically 5 layers. Initial l
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(units = 1, activation='sigmoid') #As this is a classification pro
])

  #The model is compiled.
TFmodel_SMOTE.compile(loss='binary_crossentropy', # Binary classification so binary cros
                metrics=[keras.metrics.BinaryAccuracy(name = "accuracy"),   #  Various m
                         keras.metrics.Recall(name = "recall"),
                         keras.metrics.Precision(name = "precision"),
                         keras.metrics.F1Score(name = "f1 score" )])

TFmodel_SMOTE.fit(F_train_oversampled, T_train_oversampled, validation_split=0.25, batch
```

```
metrics_oversampled = TFmodel_SMOTE.evaluate(F_test, T_test) #The model is evaluated usi

print("\n")
print("After Oversampling")
print("\n") #The results are printed.
print(f'The results representing loss, accuracy, recall, precision, and F1 values are as
```

```
Epoch 1/25
1709/1709 [==============================] - 9s 4ms/step - loss: 0.5155 - accuracy: 0.73
42 - recall: 0.5314 - precision: 0.6178 - f1 score: 0.5000 - val_loss: 0.6839 - val_accu
racy: 0.6837 - val_recall: 0.6837 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 2/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.4850 - accuracy: 0.75
53 - recall: 0.5843 - precision: 0.6474 - f1 score: 0.5000 - val_loss: 0.9094 - val_accu
racy: 0.6011 - val_recall: 0.6011 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 3/25
1709/1709 [==============================] - 8s 4ms/step - loss: 0.4686 - accuracy: 0.76
66 - recall: 0.6028 - precision: 0.6654 - f1 score: 0.5000 - val_loss: 0.8390 - val_accu
racy: 0.4818 - val_recall: 0.4818 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 4/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.4498 - accuracy: 0.78
09 - recall: 0.6175 - precision: 0.6919 - f1 score: 0.5000 - val_loss: 0.4277 - val_accu
racy: 0.8298 - val_recall: 0.8298 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 5/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.4325 - accuracy: 0.79
11 - recall: 0.6273 - precision: 0.7119 - f1 score: 0.5000 - val_loss: 0.9758 - val_accu
racy: 0.4277 - val_recall: 0.4277 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 6/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.4206 - accuracy: 0.79
84 - recall: 0.6235 - precision: 0.7319 - f1 score: 0.5000 - val_loss: 0.0652 - val_accu
racy: 0.9721 - val_recall: 0.9721 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 7/25
1709/1709 [==============================] - 9s 5ms/step - loss: 0.4132 - accuracy: 0.80
33 - recall: 0.6267 - precision: 0.7429 - f1 score: 0.5000 - val_loss: 0.1527 - val_accu
racy: 0.9424 - val_recall: 0.9424 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 8/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.4066 - accuracy: 0.80
57 - recall: 0.6260 - precision: 0.7498 - f1 score: 0.5000 - val_loss: 0.4965 - val_accu
racy: 0.7522 - val_recall: 0.7522 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 9/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.4009 - accuracy: 0.81
01 - recall: 0.6246 - precision: 0.7628 - f1 score: 0.5000 - val_loss: 0.8102 - val_accu
racy: 0.5428 - val_recall: 0.5428 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 10/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.3971 - accuracy: 0.81
20 - recall: 0.6321 - precision: 0.7632 - f1 score: 0.5000 - val_loss: 0.3035 - val_accu
racy: 0.8691 - val_recall: 0.8691 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 11/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3945 - accuracy: 0.81
35 - recall: 0.6353 - precision: 0.7653 - f1 score: 0.5000 - val_loss: 0.0989 - val_accu
racy: 0.9580 - val_recall: 0.9580 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 12/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.3909 - accuracy: 0.81
59 - recall: 0.6320 - precision: 0.7744 - f1 score: 0.5000 - val_loss: 0.6352 - val_accu
racy: 0.6806 - val_recall: 0.6806 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 13/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.3877 - accuracy: 0.81
77 - recall: 0.6291 - precision: 0.7814 - f1 score: 0.5000 - val_loss: 0.5961 - val_accu
racy: 0.7166 - val_recall: 0.7166 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 14/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3860 - accuracy: 0.81
77 - recall: 0.6246 - precision: 0.7846 - f1 score: 0.5000 - val_loss: 1.2497 - val_accu
racy: 0.3726 - val_recall: 0.3726 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 15/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.3853 - accuracy: 0.81
94 - recall: 0.6202 - precision: 0.7928 - f1 score: 0.5000 - val_loss: 0.3318 - val_accu
```

```
racy: 0.8194 - val_recall: 0.8194 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 16/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3813 - accuracy: 0.82
06 - recall: 0.6177 - precision: 0.7984 - f1 score: 0.5000 - val_loss: 0.5133 - val_accu
racy: 0.6863 - val_recall: 0.6863 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 17/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.3808 - accuracy: 0.82
20 - recall: 0.6223 - precision: 0.7992 - f1 score: 0.5000 - val_loss: 0.2787 - val_accu
racy: 0.8629 - val_recall: 0.8629 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 18/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.3783 - accuracy: 0.82
26 - recall: 0.6180 - precision: 0.8045 - f1 score: 0.5000 - val_loss: 0.4220 - val_accu
racy: 0.7791 - val_recall: 0.7791 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 19/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3762 - accuracy: 0.82
29 - recall: 0.6118 - precision: 0.8106 - f1 score: 0.5000 - val_loss: 0.5953 - val_accu
racy: 0.6609 - val_recall: 0.6609 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 20/25
1709/1709 [==============================] - 8s 5ms/step - loss: 0.3749 - accuracy: 0.82
43 - recall: 0.6157 - precision: 0.8117 - f1 score: 0.5000 - val_loss: 0.9035 - val_accu
racy: 0.4346 - val_recall: 0.4346 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 21/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3718 - accuracy: 0.82
59 - recall: 0.6214 - precision: 0.8122 - f1 score: 0.5000 - val_loss: 0.4440 - val_accu
racy: 0.7233 - val_recall: 0.7233 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 22/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3705 - accuracy: 0.82
64 - recall: 0.6222 - precision: 0.8130 - f1 score: 0.5000 - val_loss: 0.2747 - val_accu
racy: 0.8581 - val_recall: 0.8581 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 23/25
1709/1709 [==============================] - 8s 4ms/step - loss: 0.3690 - accuracy: 0.82
69 - recall: 0.6239 - precision: 0.8133 - f1 score: 0.5001 - val_loss: 0.4110 - val_accu
racy: 0.7788 - val_recall: 0.7788 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 24/25
1709/1709 [==============================] - 6s 3ms/step - loss: 0.3679 - accuracy: 0.82
80 - recall: 0.6294 - precision: 0.8124 - f1 score: 0.5002 - val_loss: 0.7294 - val_accu
racy: 0.5577 - val_recall: 0.5577 - val_precision: 1.0000 - val_f1 score: 1.0000
Epoch 25/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3671 - accuracy: 0.82
85 - recall: 0.6281 - precision: 0.8151 - f1 score: 0.5002 - val_loss: 0.4056 - val_accu
racy: 0.7909 - val_recall: 0.7909 - val_precision: 1.0000 - val_f1 score: 1.0000
1793/1793 [==============================] - 3s 1ms/step - loss: 0.3648 - accuracy: 0.84
61 - recall: 0.2626 - precision: 0.4909 - f1 score: 0.2652


After Oversampling


The results representing loss, accuracy, recall, precision, and F1 values are as follow
s: [0.3647759258747101, 0.8460841178894043, 0.26255002617836, 0.4909129738807678, array
([0.26524115], dtype=float32)]
```

In [38]:

```python
NBmodel_SMOTE = GaussianNB() #Naive Bayes this time with oversampling
NBmodel_SMOTE.fit(F_train_oversampled, T_train_oversampled)


NBpred_SMOTE = NBmodel_SMOTE.predict(F_test)

# metrics for evaluation
NBaccuracy_SMOTE = accuracy_score(T_test, NBpred_SMOTE)
NBprecision_SMOTE = precision_score(T_test, NBpred_SMOTE)
NBrecall_SMOTE = recall_score(T_test, NBpred_SMOTE)
NBf1_score_SMOTE = f1_score(T_test, NBpred_SMOTE)
NBconfusion_SMOTE = confusion_matrix(T_test, NBpred_SMOTE)
```

```python
# Printing the metrics.
print("Naive Bayes Classifier: ")
print('Accuracy score after SMOTE: ', NBaccuracy_SMOTE)
print('Confusion Matrix after SMOTE: ', NBconfusion_SMOTE)
print('Recall score after SMOTE: ', NBrecall_SMOTE)
print('F1 score after SMOTE: ', NBf1_score_SMOTE)
print('Precision score after SMOTE: ', NBprecision_SMOTE)
```

```
Naive Bayes Classifier:
Accuracy score after SMOTE:  0.6464292562185152
Confusion Matrix after SMOTE:  [[30144 18480]
 [ 1804  6941]]
Recall score after SMOTE:  0.7937106918238994
F1 score after SMOTE:  0.4063103670315518
Precision score after SMOTE:  0.2730419731717871
```

In [39]:

```python
# KNN with oversampled data

KNNmodel_SMOTE = KNeighborsClassifier(n_neighbors=6)
KNNmodel_SMOTE.fit(F_train_oversampled, T_train_oversampled)

# from the test data the model is assessed.
KNNpred_SMOTE = KNNmodel_SMOTE.predict(F_test)

# Calculate evaluation metrics
KNNaccuracy_SMOTE = accuracy_score(T_test, KNNpred_SMOTE)
KNNprecision_SMOTE = precision_score(T_test, KNNpred_SMOTE)
KNNrecall_SMOTE = recall_score(T_test, KNNpred_SMOTE)
KNNf1_score_SMOTE = f1_score(T_test, KNNpred_SMOTE)
KNNconfusion_SMOTE = confusion_matrix(T_test, KNNpred_SMOTE)

# Print the evaluation metrics
print("K-Neighbours Classifier: ")
print('Accuracy score after SMOTE:', KNNaccuracy_SMOTE)
print('Confusion Matrix after SMOTE:', KNNconfusion_SMOTE)
print('Recall score after SMOTE:', KNNrecall_SMOTE)
print('F1 score after SMOTE:', KNNf1_score_SMOTE)
print('Precision score after SMOTE:', KNNprecision_SMOTE)
```

```
K-Neighbours Classifier:
Accuracy score after SMOTE: 0.6928654848437309
Confusion Matrix after SMOTE: [[34518 14106]
 [ 3514  5231]]
Recall score after SMOTE: 0.5981703830760434
F1 score after SMOTE: 0.37255181254896375
Precision score after SMOTE: 0.27051766044370895
```

In [40]:

```python
#Random Forest Classification with oversampling

RFmodel_SMOTE = RandomForestClassifier(random_state=22)
RFmodel_SMOTE.fit(F_train_oversampled, T_train_oversampled)  # Oversampled data is used

RFpred_SMOTE = RFmodel_SMOTE.predict(F_test)

# Model evaluated
RFaccuracy_SMOTE = accuracy_score(T_test, RFpred_SMOTE)
RFprecision_SMOTE = precision_score(T_test, RFpred_SMOTE)
RFrecall_SMOTE = recall_score(T_test, RFpred_SMOTE)
RFf1_score_SMOTE = f1_score(T_test, RFpred_SMOTE)
RFconfusion_SMOTE = confusion_matrix(T_test, RFpred_SMOTE)

# Results
print("Random Forest Classifier: ")
print('Accuracy score after SMOTE:', RFaccuracy_SMOTE)
print('Confusion Matrix after SMOTE:', RFconfusion_SMOTE)
print('Recall score after SMOTE:', RFrecall_SMOTE)
```

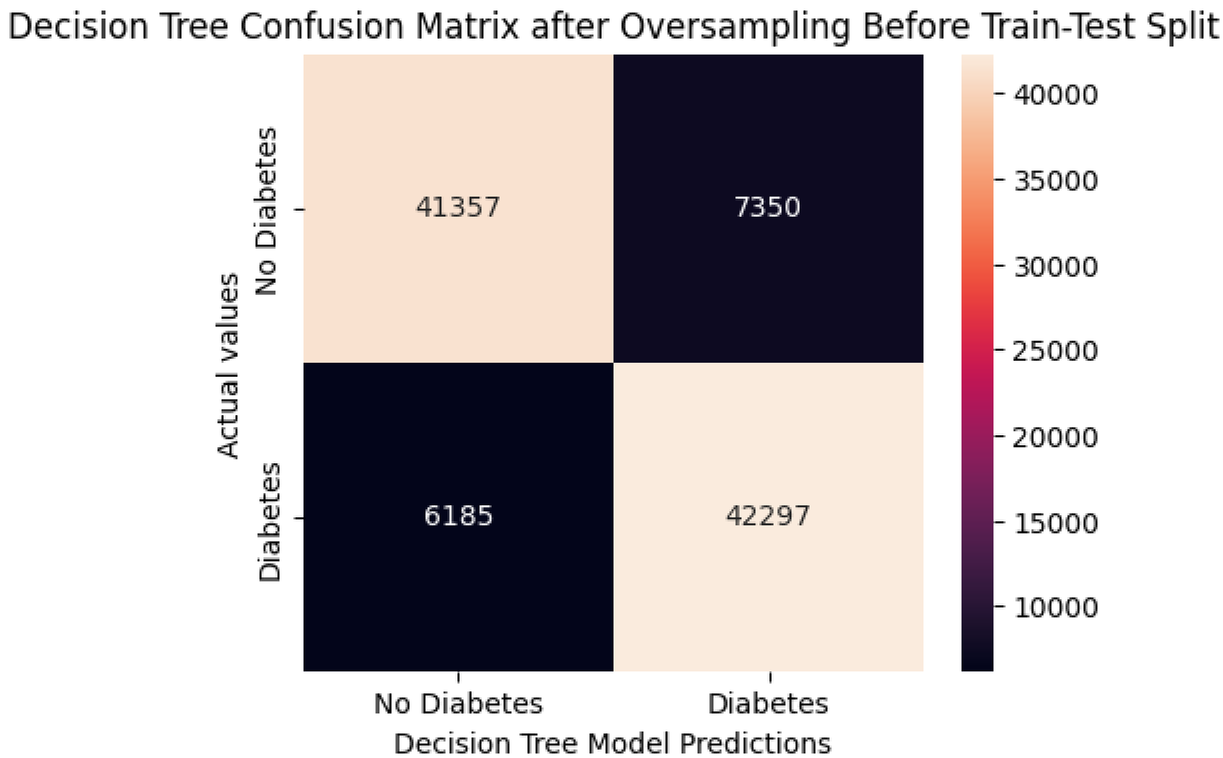
```
print('F1 score after SMOTE:', RFf1_score_SMOTE)
print('Precision score after SMOTE:', RFprecision_SMOTE)
```

```
Random Forest Classifier:
Accuracy score after SMOTE: 0.8403493175756942
Confusion Matrix after SMOTE: [[46513  2111]
 [ 7048  1697]]
Recall score after SMOTE: 0.19405374499714123
F1 score after SMOTE: 0.27037361586871667
Precision score after SMOTE: 0.44564075630252103
```

In [41]:
```
#This time the data will be oversampled before train test split to assess the changes in

t = Diabetes_df['Diabetes_binary'] #Target Column/variable

f = Diabetes_df.drop('Diabetes_binary', axis=1) #Feature variable

oversample = SMOTE(random_state=22)

F_train_oversampled_before, T_train_oversampled_before = oversample.fit_resample(f, t)

F2_train, F2_test, T2_train, T2_test = train_test_split(F_train_oversampled_before, T_tr
```

In [42]:
```
# Decision Tree Classifier after Oversampling before Train Test Split

DTmodel_SMOTE_before = DecisionTreeClassifier(random_state=22) # Model for decision tree
DTmodel_SMOTE_before = DTmodel_SMOTE_before.fit(F2_train, T2_train) # Training the model

DTpred_SMOTE_before = DTmodel_SMOTE_before.predict(F2_test) # Prediction variable establ

# Using multiple metrics for assessing the results.
DTaccuracy_SMOTE_before = accuracy_score(T2_test, DTpred_SMOTE_before)
DTconfusion_SMOTE_before = confusion_matrix(T2_test, DTpred_SMOTE_before)
DTrecall_SMOTE_before = recall_score(T2_test, DTpred_SMOTE_before)
DTf1_SMOTE_before = f1_score(T2_test, DTpred_SMOTE_before)
DTprecision_SMOTE_before = precision_score(T2_test, DTpred_SMOTE_before)

# Results are printed.
print('Accuracy score after Oversampling Before Train-Test Split: ', DTaccuracy_SMOTE_be
print('Confusion Matrix after Oversampling Before Train-Test Split: ', DTconfusion_SMOTE
print('Recall score after Oversampling Before Train-Test Split: ', DTrecall_SMOTE_before
print('F1 score after Oversampling Before Train-Test Split: ', DTf1_SMOTE_before)
print('Precision score after Oversampling Before Train-Test Split: ', DTprecision_SMOTE_



plt.figure(figsize=(5, 4))
sn.heatmap(DTconfusion_SMOTE_before, annot=True, fmt="d", xticklabels=["No Diabetes", "D
plt.title('Decision Tree Confusion Matrix after Oversampling Before Train-Test Split') #
plt.xlabel('Decision Tree Model Predictions')
plt.ylabel('Actual values')
plt.show()
```

```
Accuracy score after Oversampling Before Train-Test Split:  0.8607352683945714
Confusion Matrix after Oversampling Before Train-Test Split:  [[41357  7350]
 [ 6185 42297]]
Recall score after Oversampling Before Train-Test Split:  0.8724268800792047
F1 score after Oversampling Before Train-Test Split:  0.862069316919565
Precision score after Oversampling Before Train-Test Split:  0.8519548008943139
```



In [43]:

```
# Deep learning model, with oversampled training data BEFORE train test split.

TFmodel_SMOTE_before = keras.Sequential([
    keras.layers.Dense(units=256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(units=1, activation='sigmoid')
])

# Compiling the model and incorporating metrics.
TFmodel_SMOTE_before.compile(loss='binary_crossentropy',
                             metrics=[keras.metrics.BinaryAccuracy(name="accuracy"),
                                      keras.metrics.Recall(name="recall"),
                                      keras.metrics.Precision(name="precision"),
                                      keras.metrics.F1Score(name="f1 score")])

# Fitting the model
TFmodel_SMOTE_before.fit(F2_train, T2_train, validation_split=0.25, batch_size=128, epoc

# Evaluating the model on test data
metrics_SMOTE_before = TFmodel_SMOTE_before.evaluate(F2_test, T2_test)

# Printing the evaluation results
print("\n")
print("After Oversampling Before Train-Test Split: ")
print("\n")
print(f'The results representing loss, accuracy, recall, precision, and F1 values are as
```

```
Epoch 1/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.5434 - accuracy: 0.72
25 - recall: 0.7958 - precision: 0.6939 - f1 score: 0.6666 - val_loss: 0.5289 - val_accu
racy: 0.7394 - val_recall: 0.8359 - val_precision: 0.7017 - val_f1 score: 0.6683
Epoch 2/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.5119 - accuracy: 0.74
68 - recall: 0.8031 - precision: 0.7217 - f1 score: 0.6666 - val_loss: 0.4988 - val_accu
racy: 0.7574 - val_recall: 0.8383 - val_precision: 0.7226 - val_f1 score: 0.6683
Epoch 3/25
1709/1709 [==============================] - 6s 4ms/step - loss: 0.4872 - accuracy: 0.76
43 - recall: 0.8061 - precision: 0.7438 - f1 score: 0.6666 - val_loss: 0.5276 - val_accu
```

```
racy: 0.7433 - val_recall: 0.9401 - val_precision: 0.6755 - val_f1 score: 0.6683
Epoch 4/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.4540 - accuracy: 0.78
39 - recall: 0.8107 - precision: 0.7694 - f1 score: 0.6666 - val_loss: 0.4548 - val_accu
racy: 0.7822 - val_recall: 0.7244 - val_precision: 0.8205 - val_f1 score: 0.6683
Epoch 5/25
1709/1709 [==============================] - 6s 4ms/step - loss: 0.4261 - accuracy: 0.79
79 - recall: 0.8081 - precision: 0.7919 - f1 score: 0.6666 - val_loss: 0.4174 - val_accu
racy: 0.8083 - val_recall: 0.7968 - val_precision: 0.8166 - val_f1 score: 0.6683
Epoch 6/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.4074 - accuracy: 0.80
63 - recall: 0.8031 - precision: 0.8083 - f1 score: 0.6666 - val_loss: 0.3850 - val_accu
racy: 0.8165 - val_recall: 0.7730 - val_precision: 0.8479 - val_f1 score: 0.6683
Epoch 7/25
1709/1709 [==============================] - 8s 4ms/step - loss: 0.3962 - accuracy: 0.81
14 - recall: 0.8005 - precision: 0.8182 - f1 score: 0.6666 - val_loss: 0.3893 - val_accu
racy: 0.8117 - val_recall: 0.9006 - val_precision: 0.7656 - val_f1 score: 0.6683
Epoch 8/25
1709/1709 [==============================] - 6s 4ms/step - loss: 0.3868 - accuracy: 0.81
54 - recall: 0.7984 - precision: 0.8265 - f1 score: 0.6666 - val_loss: 0.3905 - val_accu
racy: 0.8154 - val_recall: 0.8608 - val_precision: 0.7902 - val_f1 score: 0.6683
Epoch 9/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3786 - accuracy: 0.81
86 - recall: 0.8001 - precision: 0.8308 - f1 score: 0.6666 - val_loss: 0.3738 - val_accu
racy: 0.8186 - val_recall: 0.7307 - val_precision: 0.8880 - val_f1 score: 0.6683
Epoch 10/25
1709/1709 [==============================] - 6s 4ms/step - loss: 0.3741 - accuracy: 0.82
04 - recall: 0.7970 - precision: 0.8360 - f1 score: 0.6666 - val_loss: 0.3504 - val_accu
racy: 0.8325 - val_recall: 0.8198 - val_precision: 0.8422 - val_f1 score: 0.6683
Epoch 11/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3670 - accuracy: 0.82
47 - recall: 0.7961 - precision: 0.8443 - f1 score: 0.6666 - val_loss: 0.3664 - val_accu
racy: 0.8250 - val_recall: 0.7397 - val_precision: 0.8932 - val_f1 score: 0.6683
Epoch 12/25
1709/1709 [==============================] - 6s 3ms/step - loss: 0.3652 - accuracy: 0.82
39 - recall: 0.7899 - precision: 0.8475 - f1 score: 0.6666 - val_loss: 0.3769 - val_accu
racy: 0.8183 - val_recall: 0.7227 - val_precision: 0.8950 - val_f1 score: 0.6683
Epoch 13/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3616 - accuracy: 0.82
64 - recall: 0.7986 - precision: 0.8456 - f1 score: 0.6666 - val_loss: 0.3680 - val_accu
racy: 0.8206 - val_recall: 0.7225 - val_precision: 0.9003 - val_f1 score: 0.6683
Epoch 14/25
1709/1709 [==============================] - 6s 3ms/step - loss: 0.3595 - accuracy: 0.82
77 - recall: 0.7964 - precision: 0.8494 - f1 score: 0.6666 - val_loss: 0.4835 - val_accu
racy: 0.7980 - val_recall: 0.9406 - val_precision: 0.7327 - val_f1 score: 0.6683
Epoch 15/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3558 - accuracy: 0.82
96 - recall: 0.8022 - precision: 0.8487 - f1 score: 0.6666 - val_loss: 0.3764 - val_accu
racy: 0.8194 - val_recall: 0.7405 - val_precision: 0.8807 - val_f1 score: 0.6683
Epoch 16/25
1709/1709 [==============================] - 6s 4ms/step - loss: 0.3535 - accuracy: 0.83
08 - recall: 0.8043 - precision: 0.8493 - f1 score: 0.6666 - val_loss: 0.3449 - val_accu
racy: 0.8339 - val_recall: 0.7820 - val_precision: 0.8736 - val_f1 score: 0.6683
Epoch 17/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3510 - accuracy: 0.83
32 - recall: 0.8068 - precision: 0.8518 - f1 score: 0.6666 - val_loss: 0.4535 - val_accu
racy: 0.7886 - val_recall: 0.9493 - val_precision: 0.7192 - val_f1 score: 0.6683
Epoch 18/25
1709/1709 [==============================] - 6s 3ms/step - loss: 0.3476 - accuracy: 0.83
42 - recall: 0.8068 - precision: 0.8536 - f1 score: 0.6666 - val_loss: 0.4763 - val_accu
racy: 0.7572 - val_recall: 0.5314 - val_precision: 0.9723 - val_f1 score: 0.6683
Epoch 19/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3484 - accuracy: 0.83
45 - recall: 0.8021 - precision: 0.8576 - f1 score: 0.6666 - val_loss: 0.3332 - val_accu
racy: 0.8388 - val_recall: 0.8059 - val_precision: 0.8638 - val_f1 score: 0.6683
Epoch 20/25
```

```
1709/1709 [==============================] - 6s 4ms/step - loss: 0.3464 - accuracy: 0.83
46 - recall: 0.7970 - precision: 0.8618 - f1 score: 0.6667 - val_loss: 0.3437 - val_accu
racy: 0.8321 - val_recall: 0.7466 - val_precision: 0.9020 - val_f1 score: 0.6683
Epoch 21/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3436 - accuracy: 0.83
66 - recall: 0.7966 - precision: 0.8659 - f1 score: 0.6666 - val_loss: 0.3571 - val_accu
racy: 0.8281 - val_recall: 0.7165 - val_precision: 0.9238 - val_f1 score: 0.6683
Epoch 22/25
1709/1709 [==============================] - 6s 4ms/step - loss: 0.3421 - accuracy: 0.83
65 - recall: 0.7966 - precision: 0.8656 - f1 score: 0.6666 - val_loss: 0.3351 - val_accu
racy: 0.8365 - val_recall: 0.8060 - val_precision: 0.8594 - val_f1 score: 0.6683
Epoch 23/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3377 - accuracy: 0.83
90 - recall: 0.7974 - precision: 0.8697 - f1 score: 0.6666 - val_loss: 0.3458 - val_accu
racy: 0.8372 - val_recall: 0.8040 - val_precision: 0.8624 - val_f1 score: 0.6683
Epoch 24/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3366 - accuracy: 0.83
92 - recall: 0.7988 - precision: 0.8689 - f1 score: 0.6666 - val_loss: 0.4953 - val_accu
racy: 0.7486 - val_recall: 0.5188 - val_precision: 0.9634 - val_f1 score: 0.6683
Epoch 25/25
1709/1709 [==============================] - 7s 4ms/step - loss: 0.3331 - accuracy: 0.84
10 - recall: 0.7997 - precision: 0.8716 - f1 score: 0.6666 - val_loss: 0.4032 - val_accu
racy: 0.8058 - val_recall: 0.9100 - val_precision: 0.7539 - val_f1 score: 0.6683
3038/3038 [==============================] - 5s 2ms/step - loss: 0.4072 - accuracy: 0.80
47 - recall: 0.9122 - precision: 0.7502 - f1 score: 0.6656


After Oversampling Before Train-Test Split:


The results representing loss, accuracy, recall, precision, and F1 values are as follow
s: [0.4071742594242096, 0.8046589493751526, 0.9121941924095154, 0.7501738667488098, arra
y([0.6656369], dtype=float32)]
```

In [44]:

```
#Naive Bayes Classifier oversampled before train-test split

NBmodel_SMOTE_Before = GaussianNB()
NBmodel_SMOTE_Before.fit(F2_train, T2_train)


NBpred_SMOTE_before = NBmodel_SMOTE_Before.predict(F2_test)

# metrics for evaluation
NBaccuracy_SMOTE_before = accuracy_score(T2_test, NBpred_SMOTE_before)
NBprecision_SMOTE_before = precision_score(T2_test, NBpred_SMOTE_before)
NBrecall_SMOTE_before = recall_score(T2_test, NBpred_SMOTE_before)
NBf1_score_SMOTE_before = f1_score(T2_test, NBpred_SMOTE_before)
NBconfusion_SMOTE_before = confusion_matrix(T2_test, NBpred_SMOTE_before)

# Printing the metrics.
print("Naive Bayes Classifier: ")
print('Accuracy score after SMOTE before train-test split: ', NBaccuracy_SMOTE_before)
print('Confusion Matrix after SMOTE before train-test split: ', NBconfusion_SMOTE_before
print('Recall score after SMOTE before train-test split: ', NBrecall_SMOTE_before)
print('F1 score after SMOTE before train-test split: ', NBf1_score_SMOTE_before)
print('Precision score after SMOTE before train-test split: ', NBprecision_SMOTE_before)
```

```
Naive Bayes Classifier:
Accuracy score after SMOTE before train-test split:  0.7192069061313523
Confusion Matrix after SMOTE before train-test split:  [[30095 18612]
 [ 8678 39804]]
Recall score after SMOTE before train-test split:  0.8210057340868776
F1 score after SMOTE before train-test split:  0.7447099103818594
Precision score after SMOTE before train-test split:  0.6813886606409203
```

In [45]:

```
# KNN with oversampled data before train test split
```

```python
KNNmodel_SMOTE_before = KNeighborsClassifier(n_neighbors=6)
KNNmodel_SMOTE_before.fit(F2_train, T2_train)


KNNpred_SMOTE_before = KNNmodel_SMOTE_before.predict(F2_test)

# Metrics to assess performance initialised here
KNNaccuracy_SMOTE_before = accuracy_score(T2_test, KNNpred_SMOTE_before)
KNNprecision_SMOTE_before = precision_score(T2_test, KNNpred_SMOTE_before)
KNNrecall_SMOTE_before = recall_score(T2_test, KNNpred_SMOTE_before)
KNNf1_score_SMOTE_before = f1_score(T2_test, KNNpred_SMOTE_before)
KNNconfusion_SMOTE_before = confusion_matrix(T2_test, KNNpred_SMOTE_before)

# Print the evaluation metrics
print("K-Neighbours Classifier: ")
print('Accuracy score after SMOTE before train-test split:', KNNaccuracy_SMOTE_before)
print('Confusion Matrix after SMOTE before train-test split:', KNNconfusion_SMOTE_before
print('Recall score after SMOTE before train-test split:', KNNrecall_SMOTE_before)
print('F1 score after SMOTE before train-test split:', KNNf1_score_SMOTE_before)
print('Precision score after SMOTE before train-test split:', KNNprecision_SMOTE_before)
```

```
K-Neighbours Classifier:
Accuracy score after SMOTE before train-test split: 0.8304437745012295
Confusion Matrix after SMOTE before train-test split: [[34035 14672]
 [ 1807 46675]]
Recall score after SMOTE before train-test split: 0.9627284352955736
F1 score after SMOTE before train-test split: 0.8499576614555354
Precision score after SMOTE before train-test split: 0.7608359006960406
```

In [46]:

```python
#Random Forest Classification with oversampling before train test split

RFmodel_SMOTE_before = RandomForestClassifier(random_state=22)
RFmodel_SMOTE_before.fit(F2_train, T2_train)

RFpred_SMOTE_before = RFmodel_SMOTE_before.predict(F2_test)

# Model evaluated
RFaccuracy_SMOTE_before = accuracy_score(T2_test, RFpred_SMOTE_before)
RFprecision_SMOTE_before = precision_score(T2_test, RFpred_SMOTE_before)
RFrecall_SMOTE_before = recall_score(T2_test, RFpred_SMOTE_before)
RFf1_score_SMOTE_before = f1_score(T2_test, RFpred_SMOTE_before)
RFconfusion_SMOTE_before = confusion_matrix(T2_test, RFpred_SMOTE_before)

# Results
print("Random Forest Classifier: ")
print('Accuracy score after SMOTE before train-test split:', RFaccuracy_SMOTE_before)
print('Confusion Matrix after SMOTE before train-test split:', RFconfusion_SMOTE_before)
print('Recall score after SMOTE before train-test split:', RFrecall_SMOTE_before)
print('F1 score after SMOTE before train-test split:', RFf1_score_SMOTE_before)
print('Precision score after SMOTE before train-test split:', RFprecision_SMOTE_before)
```

```
Random Forest Classifier:
Accuracy score after SMOTE before train-test split: 0.9095062198396938
Confusion Matrix after SMOTE before train-test split: [[46461  2246]
 [ 6549 41933]]
Recall score after SMOTE before train-test split: 0.8649189389876655
F1 score after SMOTE before train-test split: 0.9050841238493001
Precision score after SMOTE before train-test split: 0.9491613662599878
```

In [47]:

```python
#Feature Selection
from sklearn.feature_selection import SelectKBest, f_classif
```

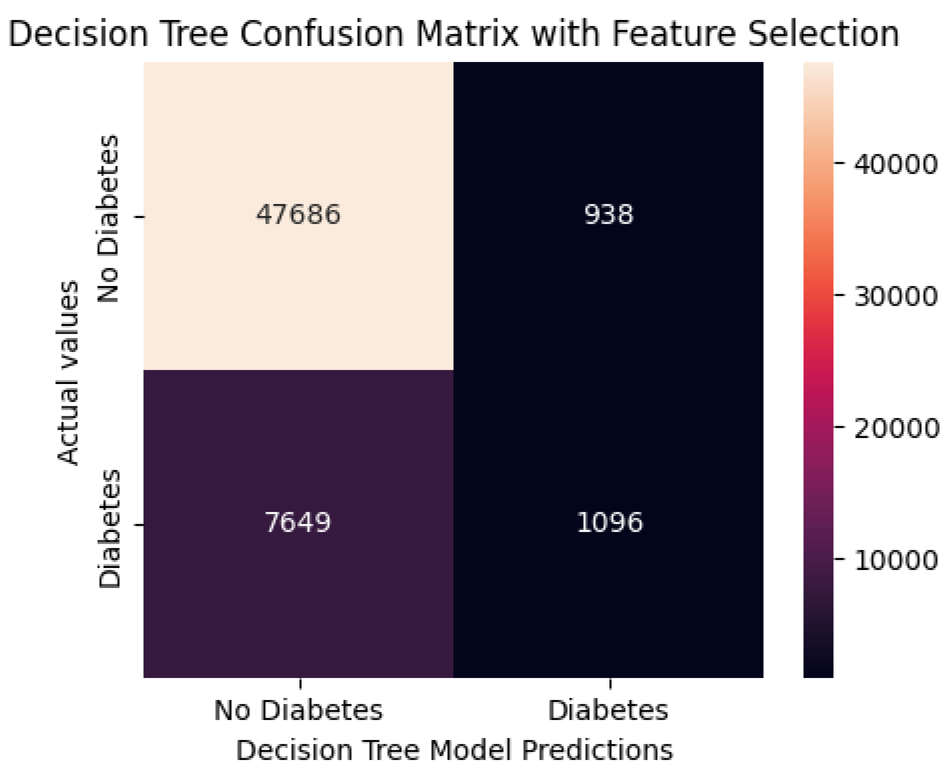
In [48]:

```python
feature_selection = SelectKBest(k=5).fit(F_train,T_train)  #Top 5 features are selected,
```

```
F_train_feature = feature_selection.transform(F_train)
F_test_feature = feature_selection.transform(F_test)
```

In [49]:
```
# Decision Tree Classifier with feature selection

DTmodel_feature = DecisionTreeClassifier(random_state=22) # Model for decision tree init


DTmodel_feature = DTmodel_feature.fit(F_train_feature, T_train) # Training the model on


DTpred_feature = DTmodel_feature.predict(F_test_feature)

# Using multiple metrics for assessing the results
DTaccuracy_feature = accuracy_score(T_test, DTpred_feature)
DTconfusion_feature = confusion_matrix(T_test, DTpred_feature)
DTrecall_feature = recall_score(T_test, DTpred_feature)
DTf1_feature = f1_score(T_test, DTpred_feature)
DTprecision_feature = precision_score(T_test, DTpred_feature)

# Results are printed
print('Accuracy score: ', DTaccuracy_feature)
print('Confusion Matrix: ', DTconfusion_feature)
print('Recall score: ', DTrecall_feature)
print('F1 score: ', DTf1_feature)
print('Precision score: ', DTprecision_feature)

# Confusion matrix is plotted
plt.figure(figsize=(5, 4))
sn.heatmap(DTconfusion_feature, annot=True, fmt="d", xticklabels=["No Diabetes", "Diabet
plt.title('Decision Tree Confusion Matrix with Feature Selection')
plt.xlabel('Decision Tree Model Predictions')
plt.ylabel('Actual values')
plt.show()
```

```
Accuracy score:  0.8503198591573846
Confusion Matrix:  [[47686   938]
 [ 7649  1096]]
Recall score:  0.12532875929102344
F1 score:  0.2033583820391502
Precision score:  0.5388397246804326
```



In [50]:

```
TFmodel_feature = tf.keras.Sequential([
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])


TFmodel_feature.compile(loss='binary_crossentropy',
                        metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                                 tf.keras.metrics.Recall(name='recall'),
                                 tf.keras.metrics.Precision(name='precision'),
                                 tf.keras.metrics.F1Score(name='f1_score')])

TFmodel_feature.fit(F_train_feature, T_train, validation_split=0.25, batch_size=128, epo

metrics_feature = TFmodel_feature.evaluate(F_test_feature, T_test) #Testing data is used

# Results are printed
print("\n")
print("For feature selection")
print("\n")
print(f'The results representing loss, accuracy, recall, precision, and F1 score values
```

```
Epoch 1/25
1009/1009 [==============================] - 6s 5ms/step - loss: 0.3996 - accuracy: 0.84
71 - recall: 0.0179 - precision: 0.4910 - f1_score: 0.2651 - val_loss: 0.3595 - val_accu
racy: 0.8504 - val_recall: 0.1251 - val_precision: 0.5651 - val_f1_score: 0.2669
Epoch 2/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3638 - accuracy: 0.84
91 - recall: 0.0717 - precision: 0.5478 - f1_score: 0.2651 - val_loss: 0.3568 - val_accu
racy: 0.8489 - val_recall: 0.0448 - val_precision: 0.6346 - val_f1_score: 0.2669
Epoch 3/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3594 - accuracy: 0.85
00 - recall: 0.0963 - precision: 0.5531 - f1_score: 0.2651 - val_loss: 0.3625 - val_accu
racy: 0.8480 - val_recall: 0.0237 - val_precision: 0.6886 - val_f1_score: 0.2669
Epoch 4/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3576 - accuracy: 0.85
07 - recall: 0.1037 - precision: 0.5621 - f1_score: 0.2651 - val_loss: 0.3569 - val_accu
racy: 0.8496 - val_recall: 0.0569 - val_precision: 0.6315 - val_f1_score: 0.2669
Epoch 5/25
1009/1009 [==============================] - 4s 3ms/step - loss: 0.3569 - accuracy: 0.85
05 - recall: 0.1069 - precision: 0.5566 - f1_score: 0.2651 - val_loss: 0.3592 - val_accu
racy: 0.8477 - val_recall: 0.0160 - val_precision: 0.7571 - val_f1_score: 0.2669
Epoch 6/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3565 - accuracy: 0.85
13 - recall: 0.1110 - precision: 0.5699 - f1_score: 0.2651 - val_loss: 0.3612 - val_accu
racy: 0.8520 - val_recall: 0.1338 - val_precision: 0.5859 - val_f1_score: 0.2669
Epoch 7/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3562 - accuracy: 0.85
12 - recall: 0.1121 - precision: 0.5665 - f1_score: 0.2651 - val_loss: 0.3552 - val_accu
racy: 0.8513 - val_recall: 0.1762 - val_precision: 0.5546 - val_f1_score: 0.2669
Epoch 8/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3562 - accuracy: 0.85
12 - recall: 0.1071 - precision: 0.5702 - f1_score: 0.2651 - val_loss: 0.3555 - val_accu
racy: 0.8501 - val_recall: 0.2233 - val_precision: 0.5318 - val_f1_score: 0.2669
Epoch 9/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3560 - accuracy: 0.85
11 - recall: 0.1104 - precision: 0.5647 - f1_score: 0.2651 - val_loss: 0.3562 - val_accu
racy: 0.8493 - val_recall: 0.0480 - val_precision: 0.6437 - val_f1_score: 0.2669
Epoch 10/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3557 - accuracy: 0.85
18 - recall: 0.1097 - precision: 0.5802 - f1_score: 0.2651 - val_loss: 0.3571 - val_accu
racy: 0.8520 - val_recall: 0.1427 - val_precision: 0.5800 - val_f1_score: 0.2669
```

```
Epoch 11/25
1009/1009 [==============================] - 4s 3ms/step - loss: 0.3556 - accuracy: 0.85
13 - recall: 0.1107 - precision: 0.5688 - f1_score: 0.2651 - val_loss: 0.3599 - val_accu
racy: 0.8494 - val_recall: 0.0471 - val_precision: 0.6541 - val_f1_score: 0.2669
Epoch 12/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3557 - accuracy: 0.85
09 - recall: 0.1108 - precision: 0.5624 - f1_score: 0.2651 - val_loss: 0.3561 - val_accu
racy: 0.8515 - val_recall: 0.0937 - val_precision: 0.6185 - val_f1_score: 0.2669
Epoch 13/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3555 - accuracy: 0.85
11 - recall: 0.1114 - precision: 0.5641 - f1_score: 0.2651 - val_loss: 0.3556 - val_accu
racy: 0.8514 - val_recall: 0.0880 - val_precision: 0.6262 - val_f1_score: 0.2669
Epoch 14/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3553 - accuracy: 0.85
14 - recall: 0.1144 - precision: 0.5678 - f1_score: 0.2651 - val_loss: 0.3654 - val_accu
racy: 0.8483 - val_recall: 0.0276 - val_precision: 0.6906 - val_f1_score: 0.2669
Epoch 15/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3555 - accuracy: 0.85
15 - recall: 0.1103 - precision: 0.5729 - f1_score: 0.2651 - val_loss: 0.3558 - val_accu
racy: 0.8505 - val_recall: 0.0706 - val_precision: 0.6324 - val_f1_score: 0.2669
Epoch 16/25
1009/1009 [==============================] - 4s 3ms/step - loss: 0.3551 - accuracy: 0.85
15 - recall: 0.1110 - precision: 0.5724 - f1_score: 0.2651 - val_loss: 0.3546 - val_accu
racy: 0.8513 - val_recall: 0.1127 - val_precision: 0.5914 - val_f1_score: 0.2669
Epoch 17/25
1009/1009 [==============================] - 5s 4ms/step - loss: 0.3553 - accuracy: 0.85
12 - recall: 0.1152 - precision: 0.5639 - f1_score: 0.2651 - val_loss: 0.3534 - val_accu
racy: 0.8524 - val_recall: 0.1482 - val_precision: 0.5821 - val_f1_score: 0.2669
Epoch 18/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3549 - accuracy: 0.85
12 - recall: 0.1142 - precision: 0.5643 - f1_score: 0.2651 - val_loss: 0.3553 - val_accu
racy: 0.8517 - val_recall: 0.1340 - val_precision: 0.5804 - val_f1_score: 0.2669
Epoch 19/25
1009/1009 [==============================] - 4s 3ms/step - loss: 0.3552 - accuracy: 0.85
15 - recall: 0.1139 - precision: 0.5703 - f1_score: 0.2651 - val_loss: 0.3567 - val_accu
racy: 0.8506 - val_recall: 0.0665 - val_precision: 0.6466 - val_f1_score: 0.2669
Epoch 20/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3553 - accuracy: 0.85
14 - recall: 0.1150 - precision: 0.5674 - f1_score: 0.2651 - val_loss: 0.3538 - val_accu
racy: 0.8514 - val_recall: 0.0914 - val_precision: 0.6203 - val_f1_score: 0.2669
Epoch 21/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3551 - accuracy: 0.85
15 - recall: 0.1158 - precision: 0.5688 - f1_score: 0.2651 - val_loss: 0.3595 - val_accu
racy: 0.8498 - val_recall: 0.2138 - val_precision: 0.5311 - val_f1_score: 0.2669
Epoch 22/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3551 - accuracy: 0.85
13 - recall: 0.1126 - precision: 0.5679 - f1_score: 0.2651 - val_loss: 0.3565 - val_accu
racy: 0.8498 - val_recall: 0.2156 - val_precision: 0.5304 - val_f1_score: 0.2669
Epoch 23/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3551 - accuracy: 0.85
13 - recall: 0.1145 - precision: 0.5663 - f1_score: 0.2651 - val_loss: 0.3571 - val_accu
racy: 0.8518 - val_recall: 0.1385 - val_precision: 0.5792 - val_f1_score: 0.2669
Epoch 24/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3552 - accuracy: 0.85
12 - recall: 0.1137 - precision: 0.5657 - f1_score: 0.2651 - val_loss: 0.3593 - val_accu
racy: 0.8491 - val_recall: 0.2138 - val_precision: 0.5246 - val_f1_score: 0.2669
Epoch 25/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3551 - accuracy: 0.85
15 - recall: 0.1126 - precision: 0.5718 - f1_score: 0.2651 - val_loss: 0.3618 - val_accu
racy: 0.8466 - val_recall: 0.2594 - val_precision: 0.5038 - val_f1_score: 0.2669
1793/1793 [==============================] - 3s 2ms/step - loss: 0.3601 - accuracy: 0.84
66 - recall: 0.2504 - precision: 0.4935 - f1_score: 0.2645
```

For feature selection

```
The results representing loss, accuracy, recall, precision, and F1 score values are as f
ollows: [0.3600764274597168, 0.8465547561645508, 0.25042882561683655, 0.4934655129909515
4, array([0.26454306], dtype=float32)]
```

In [51]:
```
NBmodel_feature = GaussianNB()
NBmodel_feature.fit(F_train_feature, T_train)

NBpred_feature = NBmodel_feature.predict(F_test_feature)

NBaccuracy_feature = accuracy_score(T_test, NBpred_feature)
NBconfusion_feature = confusion_matrix(T_test, NBpred_feature)
NBrecall_feature = recall_score(T_test, NBpred_feature)
NBf1_feature = f1_score(T_test, NBpred_feature)
NBprecision_feature = precision_score(T_test, NBpred_feature)


print("Naive Bayes Classifier after feature selection :")
print('Accuracy score: ', NBaccuracy_feature)
print('Confusion Matrix: ', NBconfusion_feature)
print('Recall score: ', NBrecall_feature)
print('F1 score: ', NBf1_feature)
print('Precision score: ', NBprecision_feature)
```

```
Naive Bayes Classifier after feature selection :
Accuracy score:  0.8171486342798375
Confusion Matrix:  [[43497  5127]
 [ 5363  3382]]
Recall score:  0.386735277301315
F1 score:  0.39202503767242375
Precision score:  0.3974615113409331
```

In [52]:
```
KNNmodel_feature = KNeighborsClassifier(n_neighbors = 6)

KNNmodel_feature.fit(F_train_feature, T_train)
KNNpred_feature = KNNmodel_feature.predict(F_test_feature)


KNNaccuracy_feature = accuracy_score(T_test, KNNpred_feature)
KNNconfusion_feature = confusion_matrix(T_test, KNNpred_feature)
KNNrecall_feature = recall_score(T_test, KNNpred_feature)
KNNf1_feature = f1_score(T_test, KNNpred_feature)
KNNprecision_feature = precision_score(T_test, KNNpred_feature)

print("K-Nearest Neighbors Classifier after feature selection:")
print('Accuracy score: ', KNNaccuracy_feature)
print('Confusion Matrix: ', KNNconfusion_feature)
print('Recall score: ', KNNrecall_feature)
print('F1 score: ', KNNf1_feature)
print('Precision score: ', KNNprecision_feature)
```

```
K-Nearest Neighbors Classifier after feature selection:
Accuracy score:  0.8433125904233993
Confusion Matrix:  [[47324  1300]
 [ 7689  1056]]
Recall score:  0.12075471698113208
F1 score:  0.19025313034861727
Precision score:  0.44821731748726656
```

In [53]:
```
RFmodel_feature = RandomForestClassifier(random_state=22)

RFmodel_feature.fit(F_train_feature, T_train)
RFpred_feature = RFmodel_feature.predict(F_test_feature)

RFaccuracy_feature = accuracy_score(T_test, RFpred_feature)
RFconfusion_feature = confusion_matrix(T_test, RFpred_feature)
```

```python
RFrecall_feature = recall_score(T_test, RFpred_feature)
RFf1_feature = f1_score(T_test, RFpred_feature)
RFprecision_feature = precision_score(T_test, RFpred_feature)

print("Random Forest Classifier after feature selection:")
print('Accuracy score: ', RFaccuracy_feature)
print('Confusion Matrix: ', RFconfusion_feature)
print('Recall score: ', RFrecall_feature)
print('F1 score: ', RFf1_feature)
print('Precision score: ', RFprecision_feature)
```

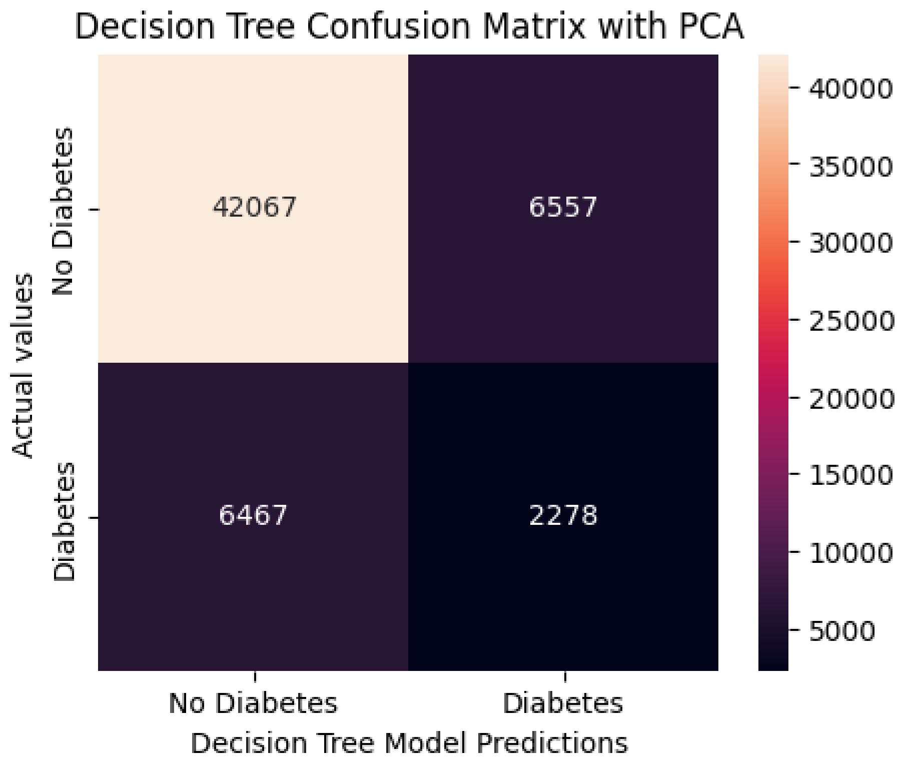
```
Random Forest Classifier after feature selection:
Accuracy score:  0.8500409628893654
Confusion Matrix:  [[47637   987]
 [ 7616  1129]]
Recall score:  0.12910234419668382
F1 score:  0.20789982506214894
Precision score:  0.5335538752362949
```

In [54]:
```python
from sklearn.decomposition import PCA #Dimensinality Reduction
```

In [55]:
```python
pca = PCA(n_components=3).fit(SS_F_train) #Using the training data we standarised earlie

F_train_3D = pca.transform(SS_F_train)
F_test_3D = pca.transform(SS_F_test)
```

In [56]:
```python
DTmodel_3D = DecisionTreeClassifier(random_state=22)


DTmodel_3D.fit(F_train_3D, T_train)


DTpred_3D = DTmodel_3D.predict(F_test_3D)


DTaccuracy_3D = accuracy_score(T_test, DTpred_3D)
DTconfusion_3D = confusion_matrix(T_test, DTpred_3D)
DTrecall_3D = recall_score(T_test, DTpred_3D)
DTf1_3D = f1_score(T_test, DTpred_3D)
DTprecision_3D = precision_score(T_test, DTpred_3D)

# Results are printed
print('Accuracy score: ', DTaccuracy_3D)
print('Confusion Matrix: ', DTconfusion_3D)
print('Recall score: ', DTrecall_3D)
print('F1 score: ', DTf1_3D)
print('Precision score: ', DTprecision_3D)

# Confusion matrix plotted
plt.figure(figsize=(5, 4))
sn.heatmap(DTconfusion_3D, annot=True, fmt="d", xticklabels=["No Diabetes", "Diabetes"],
plt.title('Decision Tree Confusion Matrix with PCA')
plt.xlabel('Decision Tree Model Predictions')
plt.ylabel('Actual values')
plt.show()
```

```
Accuracy score:  0.7729784378322787
Confusion Matrix:  [[42067  6557]
 [ 6467  2278]]
Recall score:  0.2604917095483133
F1 score:  0.25915813424345846
Precision score:  0.2578381437464629
```



In [57]:

```
#For Dimensionality Reduction
TFmodel_3D = tf.keras.Sequential([
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

TFmodel_3D.compile(loss='binary_crossentropy',
                        metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                                 tf.keras.metrics.Recall(name='recall'),
                                 tf.keras.metrics.Precision(name='precision'),
                                 tf.keras.metrics.F1Score(name='f1_score')])


TFmodel_3D.fit(F_train_3D, T_train, validation_split=0.25, batch_size=128, epochs=25)


metrics_3D = TFmodel_3D.evaluate(F_test_3D, T_test)


print('\n')
print ("For Dimensionality Reduction using PCA")
print('\n')
print(f'The results representing loss, accuracy, recall, precision, and F1 values are as
```

```
Epoch 1/25
1009/1009 [==============================] - 5s 4ms/step - loss: 0.3633 - accuracy: 0.84
65 - recall: 0.0262 - precision: 0.4599 - f1_score: 0.2651 - val_loss: 0.3672 - val_accu
racy: 0.8386 - val_recall: 0.2019 - val_precision: 0.4467 - val_f1_score: 0.2669
Epoch 2/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3602 - accuracy: 0.84
78 - recall: 0.0427 - precision: 0.5266 - f1_score: 0.2651 - val_loss: 0.3619 - val_accu
racy: 0.8473 - val_recall: 0.0368 - val_precision: 0.5635 - val_f1_score: 0.2669
Epoch 3/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3599 - accuracy: 0.84
81 - recall: 0.0424 - precision: 0.5372 - f1_score: 0.2651 - val_loss: 0.3618 - val_accu
racy: 0.8473 - val_recall: 0.0324 - val_precision: 0.5733 - val_f1_score: 0.2669
Epoch 4/25
```

```
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3597 - accuracy: 0.84
77 - recall: 0.0454 - precision: 0.5194 - f1_score: 0.2651 - val_loss: 0.3613 - val_accu
racy: 0.8472 - val_recall: 0.0496 - val_precision: 0.5438 - val_f1_score: 0.2669
Epoch 5/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3595 - accuracy: 0.84
80 - recall: 0.0461 - precision: 0.5310 - f1_score: 0.2651 - val_loss: 0.3614 - val_accu
racy: 0.8471 - val_recall: 0.0332 - val_precision: 0.5641 - val_f1_score: 0.2669
Epoch 6/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3593 - accuracy: 0.84
83 - recall: 0.0470 - precision: 0.5405 - f1_score: 0.2651 - val_loss: 0.3620 - val_accu
racy: 0.8470 - val_recall: 0.0217 - val_precision: 0.5926 - val_f1_score: 0.2669
Epoch 7/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3595 - accuracy: 0.84
80 - recall: 0.0463 - precision: 0.5299 - f1_score: 0.2651 - val_loss: 0.3614 - val_accu
racy: 0.8468 - val_recall: 0.0647 - val_precision: 0.5206 - val_f1_score: 0.2669
Epoch 8/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3594 - accuracy: 0.84
80 - recall: 0.0480 - precision: 0.5306 - f1_score: 0.2651 - val_loss: 0.3614 - val_accu
racy: 0.8462 - val_recall: 0.0928 - val_precision: 0.5037 - val_f1_score: 0.2669
Epoch 9/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3593 - accuracy: 0.84
80 - recall: 0.0492 - precision: 0.5292 - f1_score: 0.2651 - val_loss: 0.3621 - val_accu
racy: 0.8456 - val_recall: 0.0996 - val_precision: 0.4936 - val_f1_score: 0.2669
Epoch 10/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3592 - accuracy: 0.84
80 - recall: 0.0499 - precision: 0.5293 - f1_score: 0.2651 - val_loss: 0.3613 - val_accu
racy: 0.8470 - val_recall: 0.0211 - val_precision: 0.5932 - val_f1_score: 0.2669
Epoch 11/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3591 - accuracy: 0.84
82 - recall: 0.0481 - precision: 0.5356 - f1_score: 0.2651 - val_loss: 0.3617 - val_accu
racy: 0.8471 - val_recall: 0.0314 - val_precision: 0.5683 - val_f1_score: 0.2669
Epoch 12/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3592 - accuracy: 0.84
79 - recall: 0.0454 - precision: 0.5255 - f1_score: 0.2651 - val_loss: 0.3630 - val_accu
racy: 0.8468 - val_recall: 0.0166 - val_precision: 0.5914 - val_f1_score: 0.2669
Epoch 13/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3591 - accuracy: 0.84
83 - recall: 0.0474 - precision: 0.5411 - f1_score: 0.2651 - val_loss: 0.3618 - val_accu
racy: 0.8467 - val_recall: 0.0661 - val_precision: 0.5196 - val_f1_score: 0.2669
Epoch 14/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3591 - accuracy: 0.84
83 - recall: 0.0509 - precision: 0.5378 - f1_score: 0.2651 - val_loss: 0.3617 - val_accu
racy: 0.8467 - val_recall: 0.0582 - val_precision: 0.5202 - val_f1_score: 0.2669
Epoch 15/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3589 - accuracy: 0.84
82 - recall: 0.0506 - precision: 0.5368 - f1_score: 0.2651 - val_loss: 0.3614 - val_accu
racy: 0.8473 - val_recall: 0.0380 - val_precision: 0.5625 - val_f1_score: 0.2669
Epoch 16/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3590 - accuracy: 0.84
81 - recall: 0.0460 - precision: 0.5363 - f1_score: 0.2651 - val_loss: 0.3618 - val_accu
racy: 0.8467 - val_recall: 0.0736 - val_precision: 0.5159 - val_f1_score: 0.2669
Epoch 17/25
1009/1009 [==============================] - 4s 3ms/step - loss: 0.3588 - accuracy: 0.84
83 - recall: 0.0514 - precision: 0.5365 - f1_score: 0.2651 - val_loss: 0.3616 - val_accu
racy: 0.8471 - val_recall: 0.0373 - val_precision: 0.5538 - val_f1_score: 0.2669
Epoch 18/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3589 - accuracy: 0.84
81 - recall: 0.0490 - precision: 0.5337 - f1_score: 0.2651 - val_loss: 0.3623 - val_accu
racy: 0.8472 - val_recall: 0.0412 - val_precision: 0.5515 - val_f1_score: 0.2669
Epoch 19/25
1009/1009 [==============================] - 4s 4ms/step - loss: 0.3590 - accuracy: 0.84
81 - recall: 0.0474 - precision: 0.5355 - f1_score: 0.2651 - val_loss: 0.3619 - val_accu
racy: 0.8466 - val_recall: 0.0118 - val_precision: 0.6000 - val_f1_score: 0.2669
Epoch 20/25
1009/1009 [==============================] - 4s 3ms/step - loss: 0.3589 - accuracy: 0.84
81 - recall: 0.0486 - precision: 0.5340 - f1_score: 0.2651 - val_loss: 0.3619 - val_accu
```

```
racy: 0.8466 - val_recall: 0.0750 - val_precision: 0.5134 - val_f1_score: 0.2669
Epoch 21/25
1009/1009 [==============================] - 4s 3ms/step - loss: 0.3588 - accuracy: 0.84
82 - recall: 0.0514 - precision: 0.5343 - f1_score: 0.2651 - val_loss: 0.3615 - val_accu
racy: 0.8467 - val_recall: 0.0653 - val_precision: 0.5179 - val_f1_score: 0.2669
Epoch 22/25
1009/1009 [==============================] - 5s 5ms/step - loss: 0.3588 - accuracy: 0.84
84 - recall: 0.0510 - precision: 0.5438 - f1_score: 0.2651 - val_loss: 0.3613 - val_accu
racy: 0.8472 - val_recall: 0.0358 - val_precision: 0.5616 - val_f1_score: 0.2669
Epoch 23/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3588 - accuracy: 0.84
82 - recall: 0.0482 - precision: 0.5379 - f1_score: 0.2651 - val_loss: 0.3614 - val_accu
racy: 0.8468 - val_recall: 0.0616 - val_precision: 0.5217 - val_f1_score: 0.2669
Epoch 24/25
1009/1009 [==============================] - 3s 3ms/step - loss: 0.3587 - accuracy: 0.84
79 - recall: 0.0496 - precision: 0.5261 - f1_score: 0.2651 - val_loss: 0.3619 - val_accu
racy: 0.8472 - val_recall: 0.0418 - val_precision: 0.5518 - val_f1_score: 0.2669
Epoch 25/25
1009/1009 [==============================] - 5s 4ms/step - loss: 0.3588 - accuracy: 0.84
82 - recall: 0.0503 - precision: 0.5368 - f1_score: 0.2651 - val_loss: 0.3611 - val_accu
racy: 0.8472 - val_recall: 0.0333 - val_precision: 0.5667 - val_f1_score: 0.2669
1793/1793 [==============================] - 3s 2ms/step - loss: 0.3573 - accuracy: 0.84
82 - recall: 0.0310 - precision: 0.5366 - f1_score: 0.2645


For Dimensionality Reduction using PCA


The results representing loss, accuracy, recall, precision, and F1 values are as follow
s: [0.3573031723499298, 0.8482106924057007, 0.030989136546850204, 0.5366336703300476, ar
ray([0.26454306], dtype=float32)]
```

In [58]:

```python
NBmodel_3D = GaussianNB()


NBmodel_3D.fit(F_train_3D, T_train)


NBpred_3D = NBmodel_3D.predict(F_test_3D)


NBaccuracy_3D = accuracy_score(T_test, NBpred_3D)
NBconfusion_3D = confusion_matrix(T_test, NBpred_3D)
NBrecall_3D = recall_score(T_test, NBpred_3D)
NBf1_3D = f1_score(T_test, NBpred_3D)
NBprecision_3D = precision_score(T_test, NBpred_3D)

# Results are printed
print("Naive Bayes Classifier after dimensionality reduction")
print('Accuracy score: ', NBaccuracy_3D)
print('Confusion Matrix: ', NBconfusion_3D)
print('Recall score: ', NBrecall_3D)
print('F1 score: ', NBf1_3D)
print('Precision score: ', NBprecision_3D)
```

```
Naive Bayes Classifier after dimensionality reduction
Accuracy score:  0.8411162823127473
Confusion Matrix:  [[46773  1851]
 [ 7264  1481]]
Recall score:  0.16935391652372783
F1 score:  0.24525958433385772
Precision score:  0.4444777911164466
```

In [59]:

```python
KNNmodel_3D = KNeighborsClassifier(n_neighbors = 6 )

# Training the KNN classifier on the dimension reduced data
```

```python
KNNmodel_3D.fit(F_train_3D, T_train)

KNNpred_3D = KNNmodel_3D.predict(F_test_3D)

# Assessing the model's performance using various metrics
KNNaccuracy_3D = accuracy_score(T_test, KNNpred_3D)
KNNconfusion_3D = confusion_matrix(T_test, KNNpred_3D)
KNNrecall_3D = recall_score(T_test, KNNpred_3D)
KNNf1_3D = f1_score(T_test, KNNpred_3D)
KNNprecision_3D = precision_score(T_test, KNNpred_3D)

# Results are printed
print("K-Neighbours After Dimensionality Reduction")
print('Accuracy score: ', KNNaccuracy_3D)
print('Confusion Matrix: ', KNNconfusion_3D)
print('Recall score: ', KNNrecall_3D)
print('F1 score: ', KNNf1_3D)
print('Precision score: ', KNNprecision_3D)
```

```
K-Neighbours After Dimensionality Reduction
Accuracy score:  0.839320887587373
Confusion Matrix:  [[47354  1270]
 [ 7948   797]]
Recall score:  0.09113779302458548
F1 score:  0.1474287828338883
Precision score:  0.3855829704886309
```

In [60]:
```python
RFmodel_3D = RandomForestClassifier(random_state=22)

# Training the Random Forest classifier on PCA data.
RFmodel_3D.fit(F_train_3D, T_train)

# The model is tested.
RFpred_3D = RFmodel_3D.predict(F_test_3D)


RFaccuracy_3D = accuracy_score(T_test, RFpred_3D)
RFconfusion_3D = confusion_matrix(T_test, RFpred_3D)
RFrecall_3D = recall_score(T_test, RFpred_3D)
RFf1_3D = f1_score(T_test, RFpred_3D)
RFprecision_3D = precision_score(T_test, RFpred_3D)

# Results are printed

print("Random Forest after dimensionality reduction: ")
print('Accuracy score: ', RFaccuracy_3D)
print('Confusion Matrix: ', RFconfusion_3D)
print('Recall score: ', RFrecall_3D)
print('F1 score: ', RFf1_3D)
print('Precision score: ', RFprecision_3D)
```

```
Random Forest after dimensionality reduction:
Accuracy score:  0.8341613066290157
Confusion Matrix:  [[46877  1747]
 [ 7767   978]]
Recall score:  0.1118353344768439
F1 score:  0.17053182214472537
Precision score:  0.3588990825688073
```

In [60]: