Bank Churn Prediction

Problem Statement

Context

Businesses like banks which provide service have to worry about problem of 'Customer Churn' i.e. customers leaving and joining another service provider. It is important to understand which aspects of the service influence a customer's decision in this regard. Management can concentrate efforts on improvement of service, keeping in mind these priorities.

Objective

You as a Data scientist with the bank need to build a neural network based classifier that can determine whether a customer will leave the bank or not in the next 6 months.

Data Dictionary

- **CustomerId: Unique ID which is assigned to each customer**
- **Surname: Last name of the customer**
- **CreditScore: It defines the credit history of the customer.**
- **Geography: A customer's location**
- **Gender: It defines the Gender of the customer**
- **Age: Age of the customer**
- **Tenure: Number of years for which the customer has been with the bank**
- **NumOfProducts: refers to the number of products that a customer has purchased through the bank.**
- **Balance: Account balance**
- **HasCrCard: It is a categorical variable which decides whether the customer has credit card or not.**
- **EstimatedSalary: Estimated salary**
- **isActiveMember: Is is a categorical variable which decides whether the customer is active member of the bank or not (Active member in the sense, using bank products regularly, making transactions etc)**
- **Exited : whether or not the customer left the bank within six month. It can take two values 0=No (Customer did not leave the bank) 1=Yes (Customer left the bank)**

Importing necessary libraries

In [2]:

```
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Library to split data
from sklearn.model_selection import train_test_split

# library to import to standardize the data
from sklearn.preprocessing import StandardScaler, LabelEncoder

# importing different functions to build models
import tensorflow as tf
```

```python
from tensorflow import keras
from keras import backend
from keras.models import Sequential
from keras.layers import Dense, Dropout

# importing SMOTE
from imblearn.over_sampling import SMOTE

# importing metrics
from sklearn.metrics import confusion_matrix,roc_curve,classification_report,recall_score

import random

# Library to avoid the warnings
import warnings
warnings.filterwarnings("ignore")
```

Loading the dataset

In [3]:

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [5]:

```python
# Importing the dataset
data = pd.read_csv('drive/My Drive/Colab Notebooks/AIML ANN/AIMLPR4/Churn.csv')

# Check the top five records of the data
data.head()
```

Out[5]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCar
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	

Data Overview

In [6]:

```python
data.shape
```

Out[6]:

```
(10000, 14)
```

In [7]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
```

```
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [9]:

```
data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
RowNumber	10000.0	5.000500e+03	2886.895680	1.00	2500.75	5.000500e+03	7.500250e+03	10000.00
CustomerId	10000.0	1.569094e+07	71936.186123	15565701.00	15628528.25	1.569074e+07	1.575323e+07	15815690.00
CreditScore	10000.0	6.505288e+02	96.653299	350.00	584.00	6.520000e+02	7.180000e+02	850.00
Age	10000.0	3.892180e+01	10.487806	18.00	32.00	3.700000e+01	4.400000e+01	92.00
Tenure	10000.0	5.012800e+00	2.892174	0.00	3.00	5.000000e+00	7.000000e+00	10.00
Balance	10000.0	7.648589e+04	62397.405202	0.00	0.00	9.719854e+04	1.276442e+05	250898.09
NumOfProducts	10000.0	1.530200e+00	0.581654	1.00	1.00	1.000000e+00	2.000000e+00	4.00
HasCrCard	10000.0	7.055000e-01	0.455840	0.00	0.00	1.000000e+00	1.000000e+00	1.00
IsActiveMember	10000.0	5.151000e-01	0.499797	0.00	0.00	1.000000e+00	1.000000e+00	1.00
EstimatedSalary	10000.0	1.000902e+05	57510.492818	11.58	51002.11	1.001939e+05	1.493882e+05	199992.48
Exited	10000.0	2.037000e-01	0.402769	0.00	0.00	0.000000e+00	0.000000e+00	1.00

In [10]:

```
duplicate_entries = data.duplicated().sum()
print(f"Number of duplicate entries: {duplicate_entries}")
```

```
Number of duplicate entries: 0
```

In [11]:

```
# categorical columns
categorical_columns = data.select_dtypes(include=['object']).columns

if len(categorical_columns) > 0:
    for column in categorical_columns:
        print(f"Unique values in '{column}' are:")
        # count of unique values
        print(data[column].value_counts())
        # line separator
        print("*" * 50)
else:
    print("No categorical (object type) columns found in the DataFrame.")
```

```
Unique values in 'Surname' are:
Surname
Smith       32
Scott       29
Martin      29
Walker      28
Brown       26
            ..
```

```
Izmailov      1
Bold          1
Bonham        1
Poninski      1
Burbidge      1
Name: count, Length: 2932, dtype: int64
**************************************************
Unique values in 'Geography' are:
Geography
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
**************************************************
Unique values in 'Gender' are:
Gender
Male      5457
Female    4543
Name: count, dtype: int64
**************************************************
```

In [12]:

```
missing_values = data.isnull().sum()
print(missing_values)
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [13]:

```
data.shape
```

Out[13]:

```
(10000, 14)
```

In [14]:

```
data.head()
```

Out[14]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCar
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	

Observations

Dataset Overview

- **Number of Entries**: 10,000
- **Number of Features**: 14
- **Non-Null Count**: All columns have 10,000 non-null entries.

Feature Summary

- `RowNumber`, `CustomerId`, and `Surname` are unique identifiers for each customer.
- Numerical features include `CreditScore`, `Age`, `Tenure`, `Balance`, `NumOfProducts`, `HasCrCard`, `IsActiveMember`, and `EstimatedSalary`.
- Categorical features are `Geography` and `Gender`.
- The target variable is `Exited` indicating churn.

Statistical Summary

- `CreditScore` ranges from 350 to 850 with a mean of approximately 650.52.
- `Age` varies between 18 to 92 years, with an average age of around 38.92.
- The average `Balance` is around 76,485.89 with a significant standard deviation, indicating varied customer account balances.
- The majority of customers (approximately 70.55%) possess a credit card (`HasCrCard`).
- Around 51.51% of customers are active members (`IsActiveMember`).

Class Distribution

- Churn Rate (`Exited`): 20.37% of customers have exited.

Categorical Variable Distribution

- `Geography`: Most customers are from France (50.14%), followed by Germany (25.09%) and Spain (24.77%).
- `Gender`: 54.57% Male and 45.43% Female.

Unique Value Count

- `Surname`: 2,932 unique surnames with 'Smith' being the most common (32 occurrences).

Drop unnecessary features

In [15]:

```
# Drop ID column and Surname as it has too many unique values and we will drop it as it i
s not significant in churn analysis
data.drop(["CustomerId","RowNumber","Surname"], axis=1, inplace=True)
```

Exploratory Data Analysis

Submitted in a separate notebook

Data Preprocessing

Train-validation-test Split

In [16]:

```
df = data.copy()
```

In [17]:

```
X = df.drop(['Exited'],axis=1) # Credit Score through Estimated Salary
y = df['Exited'] # Exited
```

In [18]:

```
# Splitting the dataset into the Training and Testing set.

X_large, X_test, y_large, y_test = train_test_split(X, y, test_size = 0.20, random_state
= 42,stratify=y,shuffle = True)
```

In [19]:

```
# Splitting the dataset into the Training and Testing set.

X_train, X_val, y_train, y_val = train_test_split(X_large, y_large, test_size = 0.25, ra
ndom_state = 42,stratify=y_large, shuffle = True)
```

In [20]:

```
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(6000, 10) (2000, 10) (2000, 10)
```

In [21]:

```
print(y_train.shape, y_val.shape, y_test.shape)
```

```
(6000,) (2000,) (2000,)
```

Dummy Variable Creation

In [22]:

```
X_train.head()
```

Out[22]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalar
1995	584	France	Female	44	5	95671.75	2	1	1	106564.8
2724	453	Germany	Female	38	8	120623.21	1	1	0	129697.9
5224	803	Spain	Male	43	3	0.00	1	1	0	72051.4
7697	601	Spain	Female	41	3	0.00	2	1	0	54342.8
1226	531	Germany	Female	42	6	88324.31	2	1	0	75248.7

In [23]:

```
X_train = pd.get_dummies(X_train, columns=['Geography', 'Gender'], drop_first=True)
X_test = pd.get_dummies(X_test, columns=['Geography', 'Gender'], drop_first=True)
X_val = pd.get_dummies(X_val, columns=['Geography', 'Gender'], drop_first=True)
```

In [24]:

```
# Convert boolean columns to integers
X_train['Geography_Germany'] = X_train['Geography_Germany'].astype(int)
X_train['Geography_Spain'] = X_train['Geography_Spain'].astype(int)
X_train['Gender_Male'] = X_train['Gender_Male'].astype(int)

# And similarly for X_test and X_val if needed
X_test['Geography_Germany'] = X_test['Geography_Germany'].astype(int)
X_test['Geography_Spain'] = X_test['Geography_Spain'].astype(int)
X_test['Gender_Male'] = X_test['Gender_Male'].astype(int)

X_val['Geography_Germany'] = X_val['Geography_Germany'].astype(int)
```

```
X_val['Geography_Spain'] = X_val['Geography_Spain'].astype(int)
X_val['Gender_Male'] = X_val['Gender_Male'].astype(int)

# Verify the changes
X_train.head()
```

Out[24]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_Germa
1995	584	44	5	95671.75	2	1	1	106564.88	
2724	453	38	8	120623.21	1	1	0	129697.99	
5224	803	43	3	0.00	1	1	0	72051.44	
7697	601	41	3	0.00	2	1	0	54342.83	
1226	531	42	6	88324.31	2	1	0	75248.75	

In [25]:

```
X_val.head()
```

Out[25]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_German
6263	445	37	3	0.00	2	1	1	180012.39	
7644	675	28	9	0.00	1	1	0	134110.93	
429	568	40	1	99282.63	1	0	0	134600.94	
647	578	38	7	82259.29	1	1	0	8996.97	
8353	524	32	6	0.00	1	1	1	132861.90	

Data Normalization

In [26]:

```
# defining the list of columns to normalize
cols_list = ['CreditScore', 'Age', 'Tenure', 'Balance', 'EstimatedSalary']

# creating an instance of the standard scaler
sc = StandardScaler()

# Normalizing the training set
X_train[cols_list] = sc.fit_transform(X_train[cols_list])

# Transforming test sets
X_test[cols_list] = sc.transform(X_test[cols_list])
# Transforming val sets
X_val[cols_list] = sc.transform(X_val[cols_list])
```

In [27]:

```
X_train.head()
```

Out[27]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_G
1995	-0.694374	0.480890	-0.009572	0.295612	2	1	1	0.124178	
2724	-2.049957	-0.091560	1.022171	0.696248	1	1	0	0.528050	
5224	1.571829	0.385481	-	-	1	1	0	-0.478379	

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_G
7697	-0.518459	0.194665	-0.697401	-1.240550	2	1	0	-0.787547	
1226	-1.242816	0.290073	0.334342	0.177637	2	1	0	-0.422558	

Model Building

Model Evaluation Criterion

Write down the logic for choosing the metric that would be the best metric for this business scenario.

For a bank's churn prediction, Recall might be the most critical metric because:

- **The cost of false negatives (failing to identify a customer who will churn) can be higher than the cost of false positives (wrongly identifying a customer as a churn risk).**
- **The bank can target retention strategies at customers identified as likely to churn, which is less costly than acquiring new customers.**
- **A high recall ensures that the bank captures a broad segment of at-risk customers, maximizing the opportunity to retain them through interventions.**
- **Thus, optimizing for recall (while keeping an eye on precision to avoid too many false positives) can be considered a suitable approach for a churn prediction model in banking**

Function to plot confusion matrix

In [28]:

```
def make_confusion_matrix(actual_targets, predicted_targets):
    """
    To plot the confusion_matrix with percentages

    actual_targets: actual target (dependent) variable values
    predicted_targets: predicted target (dependent) variable values
    """
    cm = confusion_matrix(actual_targets, predicted_targets)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(cm.shape[0], cm.shape[1])

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Let's create two blank dataframes that will store the recall values for all the models we build.

In [29]:

```
train_metric_df = pd.DataFrame(columns=["recall"])
valid_metric_df = pd.DataFrame(columns=["recall"])
```

Neural Network with SGD Optimizer

In [30]:

```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the same o
utput everytime
```

```
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [31]:

```
#Initializing the neural network
model_0 = Sequential()
# Adding the input layer with 64 neurons and relu as activation function
model_0.add(Dense(64, activation='relu', input_dim = X_train.shape[1]))
# Add a hidden layer (specify the # of neurons and the activation function)
model_0.add(Dense(32, activation='relu'))
# Add the output layer with the number of neurons required.
model_0.add(Dense(1, activation = 'sigmoid'))
```

In [32]:

```
#Complete the code to use SGD as the optimizer.
optimizer = tf.keras.optimizers.SGD(0.001)

# uncomment one of the following lines to define the metric to be used
# metric = 'accuracy'
metric = keras.metrics.Recall()
# metric = keras.metrics.Precision()
# metric = keras.metrics.F1Score()
```

In [33]:

```
## Complete the code to compile the model with binary cross entropy as loss function and
recall as the metric.
model_0.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [34]:

```
model_0.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                768

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 2881 (11.25 KB)
Trainable params: 2881 (11.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [35]:

```
# Fitting the ANN

history_0 = model_0.fit(
    X_train, y_train,
    batch_size=32,
    validation_data=(X_val,y_val),
    epochs=100,
    verbose=1
)
```

```
Epoch 1/100
188/188 [==============================] - 1s 4ms/step - loss: 0.6151 - recall: 0.0720 -
val_loss: 0.5833 - val_recall: 0.0000e+00
Epoch 2/100
188/188 [==============================] - 0s 3ms/step - loss: 0.5628 - recall: 0.0041 -
val_loss: 0.5462 - val_recall: 0.0000e+00
Epoch 3/100
188/188 [==============================] - 1s 3ms/step - loss: 0.5341 - recall: 0.0000e+0
```

```
188/188 [==============================] - 1s 3ms/step - loss: 0.5341 - recall: 0.0000e+0
0 - val_loss: 0.5254 - val_recall: 0.0000e+00
Epoch 4/100
188/188 [==============================] - 1s 3ms/step - loss: 0.5176 - recall: 0.0000e+0
0 - val_loss: 0.5130 - val_recall: 0.0000e+00
Epoch 5/100
188/188 [==============================] - 1s 3ms/step - loss: 0.5074 - recall: 0.0000e+0
0 - val_loss: 0.5049 - val_recall: 0.0000e+00
Epoch 6/100
188/188 [==============================] - 1s 3ms/step - loss: 0.5004 - recall: 0.0000e+0
0 - val_loss: 0.4991 - val_recall: 0.0000e+00
Epoch 7/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4952 - recall: 0.0000e+0
0 - val_loss: 0.4946 - val_recall: 0.0000e+00
Epoch 8/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4909 - recall: 0.0000e+0
0 - val_loss: 0.4908 - val_recall: 0.0000e+00
Epoch 9/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4873 - recall: 0.0000e+0
0 - val_loss: 0.4875 - val_recall: 0.0000e+00
Epoch 10/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4840 - recall: 0.0000e+0
0 - val_loss: 0.4845 - val_recall: 0.0000e+00
Epoch 11/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4811 - recall: 0.0000e+0
0 - val_loss: 0.4817 - val_recall: 0.0000e+00
Epoch 12/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4783 - recall: 0.0000e+0
0 - val_loss: 0.4792 - val_recall: 0.0000e+00
Epoch 13/100
188/188 [==============================] - 1s 5ms/step - loss: 0.4757 - recall: 0.0000e+0
0 - val_loss: 0.4767 - val_recall: 0.0000e+00
Epoch 14/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4732 - recall: 0.0000e+0
0 - val_loss: 0.4744 - val_recall: 0.0000e+00
Epoch 15/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4708 - recall: 8.1766e-0
4 - val_loss: 0.4723 - val_recall: 0.0000e+00
Epoch 16/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4686 - recall: 8.1766e-0
4 - val_loss: 0.4702 - val_recall: 0.0000e+00
Epoch 17/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4665 - recall: 0.0016 -
val_loss: 0.4683 - val_recall: 0.0000e+00
Epoch 18/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4645 - recall: 0.0025 -
val_loss: 0.4664 - val_recall: 0.0000e+00
Epoch 19/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4625 - recall: 0.0025 -
val_loss: 0.4647 - val_recall: 0.0025
Epoch 20/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4607 - recall: 0.0025 -
val_loss: 0.4630 - val_recall: 0.0025
Epoch 21/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4589 - recall: 0.0041 -
val_loss: 0.4614 - val_recall: 0.0025
Epoch 22/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4573 - recall: 0.0049 -
val_loss: 0.4599 - val_recall: 0.0025
Epoch 23/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4556 - recall: 0.0057 -
val_loss: 0.4585 - val_recall: 0.0074
Epoch 24/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4541 - recall: 0.0065 -
val_loss: 0.4571 - val_recall: 0.0074
Epoch 25/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4526 - recall: 0.0098 -
val_loss: 0.4558 - val_recall: 0.0074
Epoch 26/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4512 - recall: 0.0131 -
val_loss: 0.4545 - val_recall: 0.0123
Epoch 27/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4499 - recall: 0.0155 -
```

```
188/188 [==============================] - 0s 2ms/step - loss: 0.4499 - recall: 0.0155 -
val_loss: 0.4533 - val_recall: 0.0172
Epoch 28/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4486 - recall: 0.0164 -
val_loss: 0.4522 - val_recall: 0.0197
Epoch 29/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4474 - recall: 0.0237 -
val_loss: 0.4511 - val_recall: 0.0270
Epoch 30/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4462 - recall: 0.0253 -
val_loss: 0.4501 - val_recall: 0.0270
Epoch 31/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4451 - recall: 0.0278 -
val_loss: 0.4491 - val_recall: 0.0393
Epoch 32/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4440 - recall: 0.0319 -
val_loss: 0.4481 - val_recall: 0.0418
Epoch 33/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4429 - recall: 0.0376 -
val_loss: 0.4472 - val_recall: 0.0467
Epoch 34/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4419 - recall: 0.0409 -
val_loss: 0.4464 - val_recall: 0.0491
Epoch 35/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4410 - recall: 0.0433 -
val_loss: 0.4455 - val_recall: 0.0541
Epoch 36/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4401 - recall: 0.0466 -
val_loss: 0.4448 - val_recall: 0.0541
Epoch 37/100
188/188 [==============================] - 1s 5ms/step - loss: 0.4392 - recall: 0.0523 -
val_loss: 0.4440 - val_recall: 0.0541
Epoch 38/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4383 - recall: 0.0572 -
val_loss: 0.4433 - val_recall: 0.0590
Epoch 39/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4375 - recall: 0.0630 -
val_loss: 0.4427 - val_recall: 0.0614
Epoch 40/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4368 - recall: 0.0703 -
val_loss: 0.4420 - val_recall: 0.0639
Epoch 41/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4360 - recall: 0.0752 -
val_loss: 0.4414 - val_recall: 0.0663
Epoch 42/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4353 - recall: 0.0801 -
val_loss: 0.4408 - val_recall: 0.0688
Epoch 43/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4346 - recall: 0.0834 -
val_loss: 0.4403 - val_recall: 0.0713
Epoch 44/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4339 - recall: 0.0891 -
val_loss: 0.4398 - val_recall: 0.0737
Epoch 45/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4333 - recall: 0.0932 -
val_loss: 0.4393 - val_recall: 0.0737
Epoch 46/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4327 - recall: 0.0957 -
val_loss: 0.4388 - val_recall: 0.0762
Epoch 47/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4321 - recall: 0.1047 -
val_loss: 0.4384 - val_recall: 0.0762
Epoch 48/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4315 - recall: 0.1096 -
val_loss: 0.4379 - val_recall: 0.0835
Epoch 49/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4310 - recall: 0.1104 -
val_loss: 0.4375 - val_recall: 0.0885
Epoch 50/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4304 - recall: 0.1169 -
val_loss: 0.4371 - val_recall: 0.0958
Epoch 51/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4299 - recall: 0.1202 -
```

```
188/188 [==============================] - 0s 2ms/step - loss: 0.4299 - recall: 0.1202 -
val_loss: 0.4367 - val_recall: 0.0983
Epoch 52/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4294 - recall: 0.1226 -
val_loss: 0.4364 - val_recall: 0.0983
Epoch 53/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4289 - recall: 0.1259 -
val_loss: 0.4360 - val_recall: 0.1007
Epoch 54/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4284 - recall: 0.1300 -
val_loss: 0.4356 - val_recall: 0.1007
Epoch 55/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4280 - recall: 0.1357 -
val_loss: 0.4353 - val_recall: 0.1057
Epoch 56/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4275 - recall: 0.1382 -
val_loss: 0.4350 - val_recall: 0.1106
Epoch 57/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4270 - recall: 0.1374 -
val_loss: 0.4347 - val_recall: 0.1106
Epoch 58/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4266 - recall: 0.1406 -
val_loss: 0.4344 - val_recall: 0.1130
Epoch 59/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4262 - recall: 0.1447 -
val_loss: 0.4341 - val_recall: 0.1179
Epoch 60/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4258 - recall: 0.1480 -
val_loss: 0.4338 - val_recall: 0.1204
Epoch 61/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4254 - recall: 0.1472 -
val_loss: 0.4335 - val_recall: 0.1302
Epoch 62/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4250 - recall: 0.1504 -
val_loss: 0.4332 - val_recall: 0.1327
Epoch 63/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4246 - recall: 0.1513 -
val_loss: 0.4330 - val_recall: 0.1351
Epoch 64/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4242 - recall: 0.1554 -
val_loss: 0.4327 - val_recall: 0.1351
Epoch 65/100
188/188 [==============================] - 1s 5ms/step - loss: 0.4238 - recall: 0.1586 -
val_loss: 0.4325 - val_recall: 0.1327
Epoch 66/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4235 - recall: 0.1619 -
val_loss: 0.4322 - val_recall: 0.1327
Epoch 67/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4231 - recall: 0.1619 -
val_loss: 0.4320 - val_recall: 0.1351
Epoch 68/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4227 - recall: 0.1635 -
val_loss: 0.4317 - val_recall: 0.1351
Epoch 69/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4224 - recall: 0.1643 -
val_loss: 0.4315 - val_recall: 0.1425
Epoch 70/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4221 - recall: 0.1709 -
val_loss: 0.4313 - val_recall: 0.1425
Epoch 71/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4217 - recall: 0.1742 -
val_loss: 0.4311 - val_recall: 0.1450
Epoch 72/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4214 - recall: 0.1774 -
val_loss: 0.4309 - val_recall: 0.1474
Epoch 73/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4211 - recall: 0.1742 -
val_loss: 0.4306 - val_recall: 0.1523
Epoch 74/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4208 - recall: 0.1807 -
val_loss: 0.4304 - val_recall: 0.1548
Epoch 75/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4205 - recall: 0.1807
```

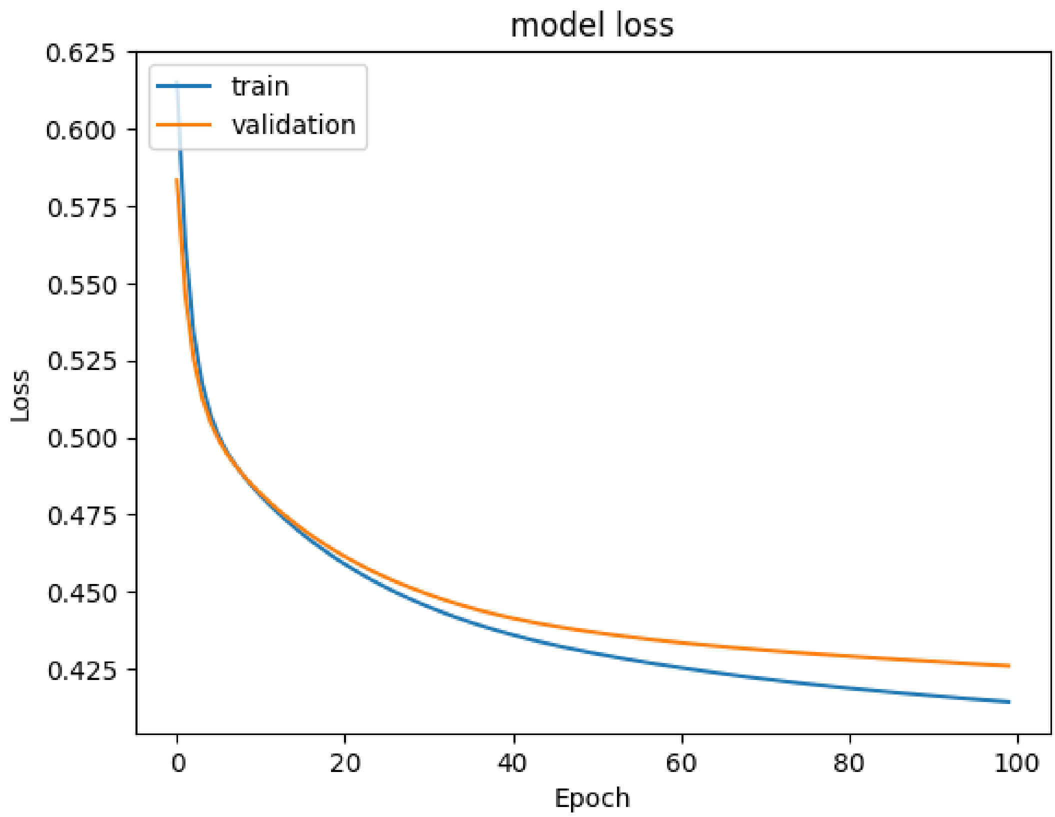
```
188/188 [==============================] - 1s 3ms/step - loss: 0.4205 - recall: 0.1807 -
val_loss: 0.4302 - val_recall: 0.1622
Epoch 76/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4202 - recall: 0.1856 -
val_loss: 0.4300 - val_recall: 0.1622
Epoch 77/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4199 - recall: 0.1856 -
val_loss: 0.4298 - val_recall: 0.1695
Epoch 78/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4196 - recall: 0.1897 -
val_loss: 0.4297 - val_recall: 0.1720
Epoch 79/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4193 - recall: 0.1856 -
val_loss: 0.4295 - val_recall: 0.1769
Epoch 80/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4190 - recall: 0.1897 -
val_loss: 0.4293 - val_recall: 0.1769
Epoch 81/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4187 - recall: 0.1938 -
val_loss: 0.4291 - val_recall: 0.1794
Epoch 82/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4185 - recall: 0.1889 -
val_loss: 0.4289 - val_recall: 0.1794
Epoch 83/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4182 - recall: 0.1938 -
val_loss: 0.4287 - val_recall: 0.1794
Epoch 84/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4180 - recall: 0.1946 -
val_loss: 0.4286 - val_recall: 0.1794
Epoch 85/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4177 - recall: 0.1987 -
val_loss: 0.4284 - val_recall: 0.1794
Epoch 86/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4175 - recall: 0.1995 -
val_loss: 0.4282 - val_recall: 0.1769
Epoch 87/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4172 - recall: 0.1971 -
val_loss: 0.4280 - val_recall: 0.1892
Epoch 88/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4170 - recall: 0.2044 -
val_loss: 0.4279 - val_recall: 0.1818
Epoch 89/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4167 - recall: 0.2011 -
val_loss: 0.4277 - val_recall: 0.1843
Epoch 90/100
188/188 [==============================] - 1s 4ms/step - loss: 0.4165 - recall: 0.2028 -
val_loss: 0.4276 - val_recall: 0.1867
Epoch 91/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4163 - recall: 0.2134 -
val_loss: 0.4274 - val_recall: 0.1818
Epoch 92/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4160 - recall: 0.2052 -
val_loss: 0.4272 - val_recall: 0.1843
Epoch 93/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4158 - recall: 0.2052 -
val_loss: 0.4271 - val_recall: 0.1867
Epoch 94/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4156 - recall: 0.2118 -
val_loss: 0.4269 - val_recall: 0.1867
Epoch 95/100
188/188 [==============================] - 0s 3ms/step - loss: 0.4154 - recall: 0.2110 -
val_loss: 0.4267 - val_recall: 0.1892
Epoch 96/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4152 - recall: 0.2126 -
val_loss: 0.4266 - val_recall: 0.1941
Epoch 97/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4149 - recall: 0.2126 -
val_loss: 0.4264 - val_recall: 0.1941
Epoch 98/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4147 - recall: 0.2167 -
val_loss: 0.4263 - val_recall: 0.1966
Epoch 99/100
188/188 [==============================] - 0s 2ms/step - loss: 0.4145 - recall: 0.2191
```

```
188/188 [==============================] - 0s 2ms/step - loss: 0.4145 - recall: 0.2191 -
val_loss: 0.4261 - val_recall: 0.1941
Epoch 100/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4143 - recall: 0.2175 -
val_loss: 0.4260 - val_recall: 0.1941
```

Loss

In [36]:

```
#Plotting Train Loss vs Validation Loss
plt.plot(history_0.history['loss'])
plt.plot(history_0.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



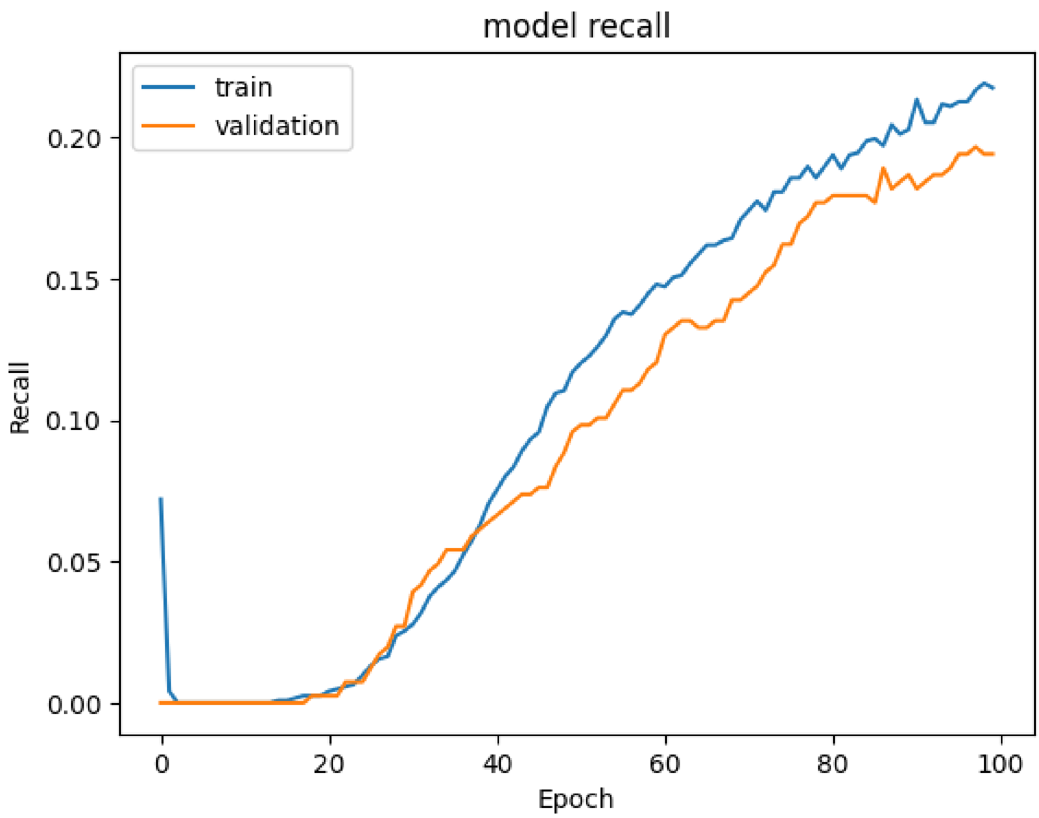
- **The model shows a rapid decrease in loss for both training and validation datasets in the initial epochs.**
- **As the epochs increase, both curves plateau, indicating that the model is converging.**
- **The training and validation loss are closely aligned throughout the training process.**
- **There is no sign of overfitting, as the validation loss does not increase as the epochs go by. Instead, it closely tracks the training loss, which is desirable.**
- **The model could potentially benefit from early stopping, as the loss doesn't show significant improvement after around 20 epochs.**
- **Given the stability of the validation loss, the learning rate and model complexity appear to be well-configured for this dataset.**

Recall

In [38]:

```
#Plotting Train recall vs Validation recall
plt.plot(history_0.history['recall'])
plt.plot(history_0.history['val_recall'])
plt.title('model recall')
plt.ylabel('Recall')
plt.xlabel('Epoch')
```

```
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



- **The recall for both training and validation data shows an upward trend, suggesting an improvement in the model's ability to correctly identify positive cases (customers likely to churn) as training progresses.**
- **There is a noticeable volatility in recall, especially in the initial epochs, which then smoothens out indicating that the model starts to stabilize.**
- **The recall for the validation set closely follows the training recall, which is a good sign that the model generalizes well.**
- **The model does not show perfect convergence as the recall continues to slowly increase, suggesting that further training or hyperparameter tuning might yield better results.**
- **The recall metric has not plateaued by the 100th epoch, implying that either the model could benefit from more training epochs or additional techniques to enhance recall could be applied (e.g., adjusting class weights or the decision threshold).**

In [39]:

```
#Predicting the results using best as a threshold
y_train_pred = model_0.predict(X_train)
y_train_pred = (y_train_pred > 0.5)
y_train_pred
```

```
188/188 [==============================] - 0s 1ms/step
```

Out[39]:

```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [40]:

```
#Predicting the results using best as a threshold
y_val_pred = model_0.predict(X_val)     ## Complete the code to make prediction on the val
idation set
y_val_pred = (y_val_pred > 0.5)
y_val_pred
```

```
63/63 [==============================] - 0s 1ms/step
```

Out[40]:

```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [41]:

```
model_name = "NN with SGD"

train_metric_df.loc[model_name] = recall_score(y_train, y_train_pred)
valid_metric_df.loc[model_name] = recall_score(y_val, y_val_pred)

print(train_metric_df)
```

```
               recall
NN with SGD  0.219133
```

In [42]:

```
#Classification report
cr = classification_report(y_train, y_train_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.83      0.97      0.90      4777
           1       0.67      0.22      0.33      1223

    accuracy                           0.82      6000
   macro avg       0.75      0.60      0.61      6000
weighted avg       0.80      0.82      0.78      6000
```

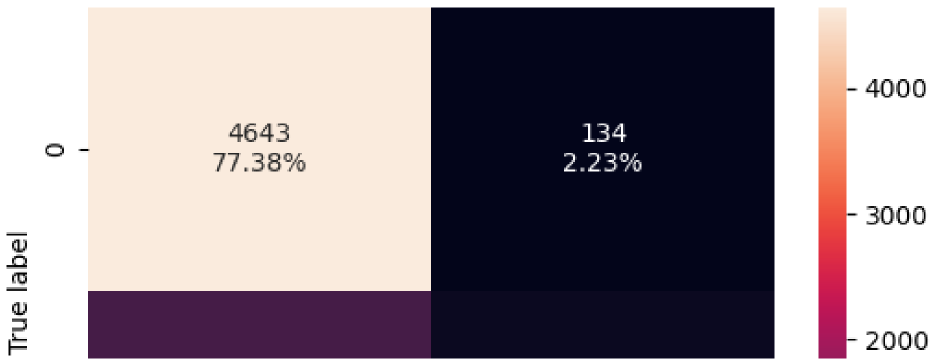
In [43]:

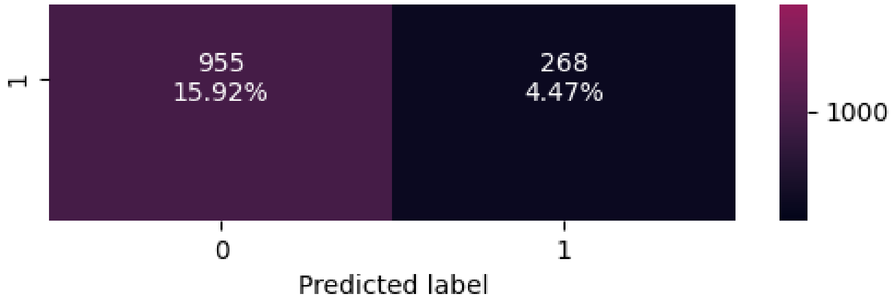
```
#Classification report
cr = classification_report(y_val, y_val_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      1593
           1       0.61      0.19      0.29       407

    accuracy                           0.81      2000
   macro avg       0.72      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000
```

In [44]:

```
make_confusion_matrix(y_train, y_train_pred)
```

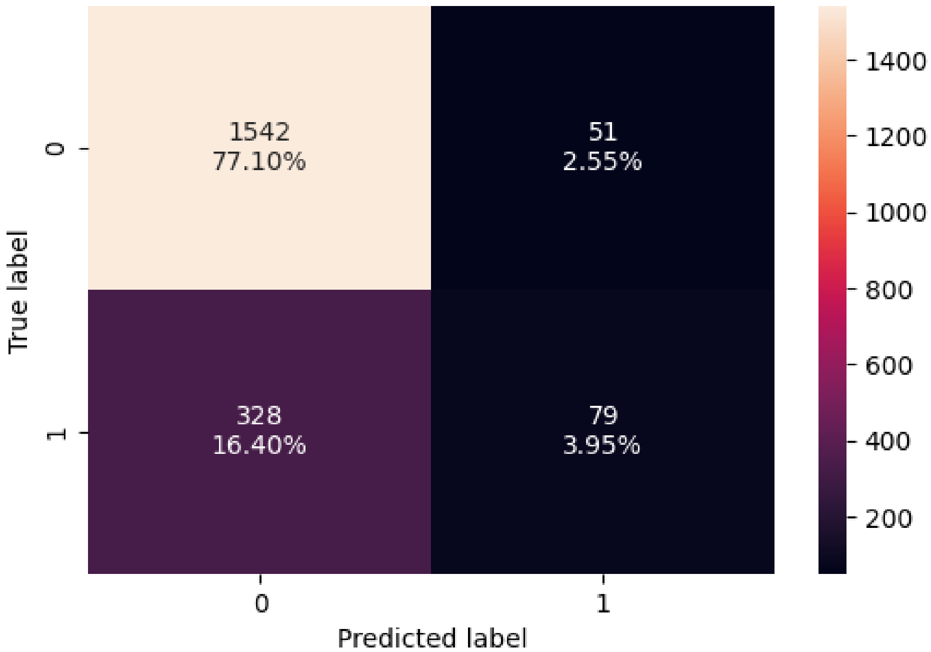




- The confusion matrix shows the distribution of the true and predicted labels.
- True negatives (correctly predicted non-churn): 4643, making up 77.38% of the total predictions.
- False positives (incorrectly predicted churn): 134, making up 2.23% of the total predictions.
- False negatives (incorrectly predicted non-churn): 955, making up 15.92% of the total predictions.
- True positives (correctly predicted churn): 268, making up 4.47% of the total predictions.
- The model shows a relatively high rate of false negatives, which is critical in churn prediction as it represents customers who are likely to churn but were not identified by the model.
- The recall (true positives / (true positives + false negatives)) can be considered lower than desired for churn prediction, indicating potential room for improvement in the model's ability to identify all positive (churn) cases.

In [47]:

```
make_confusion_matrix(y_val, y_val_pred)
```



- The matrix shows a considerable number of true negatives (TN): 1542 (77.10%), indicating the model's effectiveness in identifying customers who did not churn.
- The true positives (TP) count is 79, representing 3.95% of predictions, showing the model's ability to identify actual churn cases.
- However, there is a relatively high count of false negatives (FN): 328 (16.40%), which could represent a significant missed opportunity in terms of customer retention efforts.
- False positives (FP) are comparatively lower at 51, translating to 2.55% of predictions, suggesting that the model is conservative in predicting churn.
- The higher number of false negatives could indicate that the model may benefit from measures to improve recall, ensuring that fewer actual churn cases are missed.

Model Performance Improvement

Neural Network with Adam Optimizer

In [48]:

```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the same o
utput everytime
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [49]:

```
#Initializing the neural network
model_1 = Sequential()
# Add a input layer (specify the # of neurons and activation function)
model_1.add(Dense(64,activation='relu',input_dim = X_train.shape[1]))
# Add a hidden layer (specify the # of neurons and activation function)
model_1.add(Dense(32,activation='relu'))
# Add a output layer with the required number of neurons and relu as activation function
model_1.add(Dense(1, activation = 'sigmoid'))
```

In [50]:

```
#Complete the code to use Adam as the optimizer.
optimizer = tf.keras.optimizers.Adam()

# uncomment one of the following lines to define the metric to be used
# metric = 'accuracy'
metric = keras.metrics.Recall()
# metric = keras.metrics.Precision()
# metric = keras.metrics.F1Score()
```

In [51]:

```
# Complete the code to compile the model with binary cross entropy as loss function and r
ecall as the metric
model_1.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [52]:

```
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                768

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 2881 (11.25 KB)
Trainable params: 2881 (11.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [53]:

```
#Fitting the ANN
history_1 = model_1.fit(
    X_train,y_train,
    batch_size=32, ## Complete the code to specify the batch size to use
    validation_data=(X_val,y_val),
    epochs=100, ## Complete the code to specify the number of epochs
    verbose=1
)
```

```
Epoch 1/100
188/188 [==============================] - 2s 4ms/step - loss: 0.4575 - recall: 0.0981 -
```

```
val_loss: 0.4252 - val_recall: 0.2088
Epoch 2/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4122 - recall: 0.2608 -
val_loss: 0.4150 - val_recall: 0.2948
Epoch 3/100
188/188 [==============================] - 0s 2ms/step - loss: 0.3981 - recall: 0.3189 -
val_loss: 0.4081 - val_recall: 0.3538
Epoch 4/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3866 - recall: 0.3475 -
val_loss: 0.4016 - val_recall: 0.3735
Epoch 5/100
188/188 [==============================] - 0s 2ms/step - loss: 0.3768 - recall: 0.3810 -
val_loss: 0.3934 - val_recall: 0.3391
Epoch 6/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3701 - recall: 0.3982 -
val_loss: 0.3867 - val_recall: 0.3661
Epoch 7/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3613 - recall: 0.4088 -
val_loss: 0.3822 - val_recall: 0.3636
Epoch 8/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3546 - recall: 0.4276 -
val_loss: 0.3791 - val_recall: 0.3833
Epoch 9/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3483 - recall: 0.4407 -
val_loss: 0.3754 - val_recall: 0.4324
Epoch 10/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3437 - recall: 0.4554 -
val_loss: 0.3700 - val_recall: 0.4103
Epoch 11/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3397 - recall: 0.4587 -
val_loss: 0.3663 - val_recall: 0.4275
Epoch 12/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3332 - recall: 0.4620 -
val_loss: 0.3760 - val_recall: 0.5135
Epoch 13/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3301 - recall: 0.4702 -
val_loss: 0.3649 - val_recall: 0.3833
Epoch 14/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3266 - recall: 0.4865 -
val_loss: 0.3633 - val_recall: 0.3907
Epoch 15/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3225 - recall: 0.4857 -
val_loss: 0.3617 - val_recall: 0.4128
Epoch 16/100
188/188 [==============================] - 0s 2ms/step - loss: 0.3202 - recall: 0.4865 -
val_loss: 0.3702 - val_recall: 0.5233
Epoch 17/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3187 - recall: 0.5020 -
val_loss: 0.3595 - val_recall: 0.3907
Epoch 18/100
188/188 [==============================] - 0s 3ms/step - loss: 0.3160 - recall: 0.4947 -
val_loss: 0.3635 - val_recall: 0.4472
Epoch 19/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3147 - recall: 0.5094 -
val_loss: 0.3650 - val_recall: 0.3808
Epoch 20/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3109 - recall: 0.5151 -
val_loss: 0.3604 - val_recall: 0.4349
Epoch 21/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3094 - recall: 0.5094 -
val_loss: 0.3672 - val_recall: 0.3415
Epoch 22/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3078 - recall: 0.5078 -
val_loss: 0.3642 - val_recall: 0.4644
Epoch 23/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3104 - recall: 0.5258 -
val_loss: 0.3649 - val_recall: 0.3857
Epoch 24/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3080 - recall: 0.5086 -
val_loss: 0.3616 - val_recall: 0.4177
Epoch 25/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3048 - recall: 0.5217 -
```

```
val_loss: 0.3669 - val_recall: 0.4914
Epoch 26/100
188/188 [==============================] - 0s 3ms/step - loss: 0.3054 - recall: 0.5225 -
val_loss: 0.3669 - val_recall: 0.4472
Epoch 27/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3029 - recall: 0.5192 -
val_loss: 0.3592 - val_recall: 0.4521
Epoch 28/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3016 - recall: 0.5307 -
val_loss: 0.3640 - val_recall: 0.4398
Epoch 29/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3017 - recall: 0.5298 -
val_loss: 0.3614 - val_recall: 0.4570
Epoch 30/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2990 - recall: 0.5323 -
val_loss: 0.3630 - val_recall: 0.4300
Epoch 31/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2975 - recall: 0.5258 -
val_loss: 0.3660 - val_recall: 0.4398
Epoch 32/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2994 - recall: 0.5298 -
val_loss: 0.3702 - val_recall: 0.4054
Epoch 33/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2984 - recall: 0.5331 -
val_loss: 0.3676 - val_recall: 0.5086
Epoch 34/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2959 - recall: 0.5364 -
val_loss: 0.3621 - val_recall: 0.5061
Epoch 35/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2952 - recall: 0.5380 -
val_loss: 0.3631 - val_recall: 0.4717
Epoch 36/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2947 - recall: 0.5421 -
val_loss: 0.3657 - val_recall: 0.5332
Epoch 37/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2954 - recall: 0.5339 -
val_loss: 0.3652 - val_recall: 0.4963
Epoch 38/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2923 - recall: 0.5511 -
val_loss: 0.3659 - val_recall: 0.4668
Epoch 39/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2914 - recall: 0.5405 -
val_loss: 0.3731 - val_recall: 0.4128
Epoch 40/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2908 - recall: 0.5446 -
val_loss: 0.3628 - val_recall: 0.4963
Epoch 41/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2902 - recall: 0.5536 -
val_loss: 0.3823 - val_recall: 0.3735
Epoch 42/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2881 - recall: 0.5397 -
val_loss: 0.3741 - val_recall: 0.3956
Epoch 43/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2904 - recall: 0.5658 -
val_loss: 0.3672 - val_recall: 0.4889
Epoch 44/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2872 - recall: 0.5568 -
val_loss: 0.3664 - val_recall: 0.4889
Epoch 45/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2859 - recall: 0.5536 -
val_loss: 0.3702 - val_recall: 0.5012
Epoch 46/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2879 - recall: 0.5511 -
val_loss: 0.3681 - val_recall: 0.5307
Epoch 47/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2837 - recall: 0.5666 -
val_loss: 0.3759 - val_recall: 0.4791
Epoch 48/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2832 - recall: 0.5699 -
val_loss: 0.3815 - val_recall: 0.4079
Epoch 49/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2831 - recall: 0.5478 -
```

```
val_loss: 0.3805 - val_recall: 0.5479
Epoch 50/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2818 - recall: 0.5634 -
val_loss: 0.3810 - val_recall: 0.4103
Epoch 51/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2803 - recall: 0.5650 -
val_loss: 0.3835 - val_recall: 0.4054
Epoch 52/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2803 - recall: 0.5707 -
val_loss: 0.3725 - val_recall: 0.5160
Epoch 53/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2804 - recall: 0.5732 -
val_loss: 0.3787 - val_recall: 0.4668
Epoch 54/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2794 - recall: 0.5634 -
val_loss: 0.3899 - val_recall: 0.5749
Epoch 55/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2779 - recall: 0.5724 -
val_loss: 0.3857 - val_recall: 0.4054
Epoch 56/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2774 - recall: 0.5666 -
val_loss: 0.3816 - val_recall: 0.4545
Epoch 57/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2761 - recall: 0.5715 -
val_loss: 0.3848 - val_recall: 0.4668
Epoch 58/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2760 - recall: 0.5773 -
val_loss: 0.3786 - val_recall: 0.4496
Epoch 59/100
188/188 [==============================] - 1s 5ms/step - loss: 0.2745 - recall: 0.5724 -
val_loss: 0.3833 - val_recall: 0.4300
Epoch 60/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2747 - recall: 0.5805 -
val_loss: 0.3841 - val_recall: 0.4619
Epoch 61/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2740 - recall: 0.5699 -
val_loss: 0.3847 - val_recall: 0.4914
Epoch 62/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2741 - recall: 0.5756 -
val_loss: 0.3913 - val_recall: 0.5430
Epoch 63/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2725 - recall: 0.5846 -
val_loss: 0.3823 - val_recall: 0.4767
Epoch 64/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2695 - recall: 0.5715 -
val_loss: 0.3886 - val_recall: 0.5455
Epoch 65/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2728 - recall: 0.5977 -
val_loss: 0.4025 - val_recall: 0.3907
Epoch 66/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2709 - recall: 0.5871 -
val_loss: 0.3891 - val_recall: 0.4914
Epoch 67/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2692 - recall: 0.5871 -
val_loss: 0.3894 - val_recall: 0.4226
Epoch 68/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2675 - recall: 0.5863 -
val_loss: 0.3820 - val_recall: 0.5037
Epoch 69/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2667 - recall: 0.5863 -
val_loss: 0.3913 - val_recall: 0.4742
Epoch 70/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2659 - recall: 0.5912 -
val_loss: 0.3957 - val_recall: 0.4128
Epoch 71/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2670 - recall: 0.5944 -
val_loss: 0.3894 - val_recall: 0.4644
Epoch 72/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2653 - recall: 0.5969 -
val_loss: 0.3930 - val_recall: 0.5111
Epoch 73/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2641 - recall: 0.6043 -
```

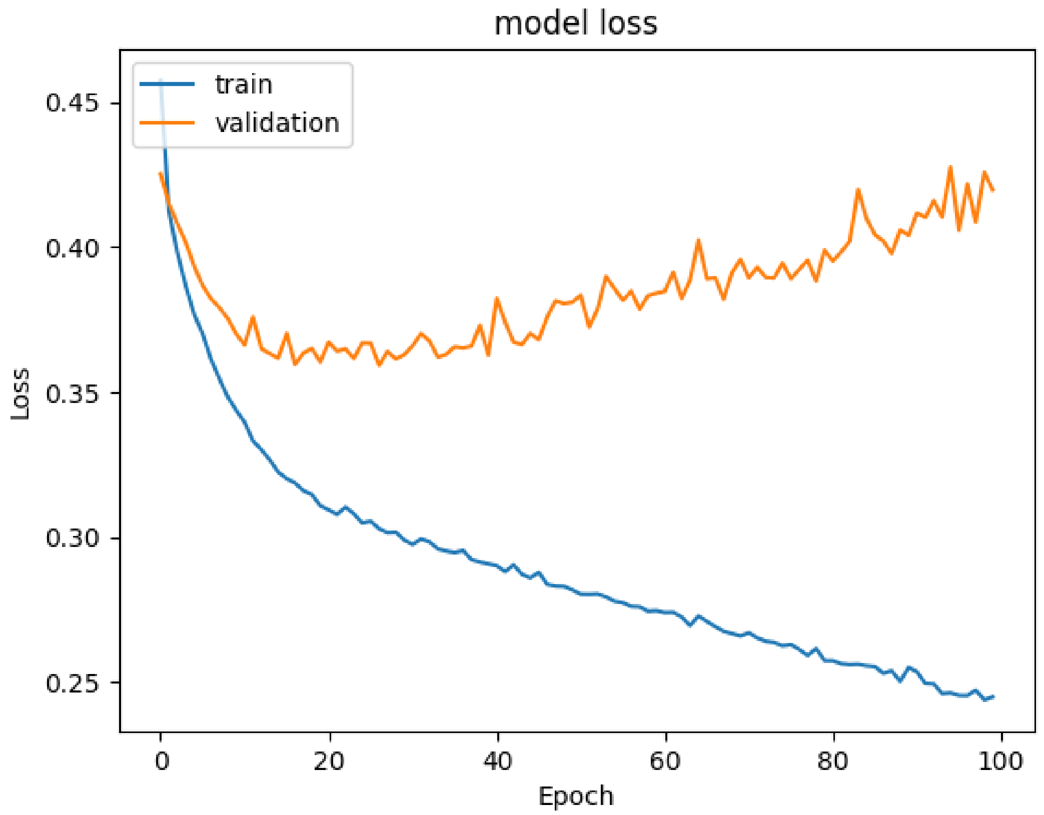
```
val_loss: 0.3896 - val_recall: 0.5381
Epoch 74/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2637 - recall: 0.6034 -
val_loss: 0.3894 - val_recall: 0.5160
Epoch 75/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2626 - recall: 0.5961 -
val_loss: 0.3944 - val_recall: 0.5086
Epoch 76/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2630 - recall: 0.5928 -
val_loss: 0.3891 - val_recall: 0.4717
Epoch 77/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2613 - recall: 0.6100 -
val_loss: 0.3923 - val_recall: 0.5012
Epoch 78/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2592 - recall: 0.6002 -
val_loss: 0.3954 - val_recall: 0.4840
Epoch 79/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2616 - recall: 0.5985 -
val_loss: 0.3883 - val_recall: 0.5233
Epoch 80/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2574 - recall: 0.6083 -
val_loss: 0.3989 - val_recall: 0.4226
Epoch 81/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2574 - recall: 0.5993 -
val_loss: 0.3952 - val_recall: 0.4840
Epoch 82/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2563 - recall: 0.6092 -
val_loss: 0.3984 - val_recall: 0.4717
Epoch 83/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2560 - recall: 0.6157 -
val_loss: 0.4021 - val_recall: 0.5184
Epoch 84/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2562 - recall: 0.6214 -
val_loss: 0.4199 - val_recall: 0.3956
Epoch 85/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2556 - recall: 0.6083 -
val_loss: 0.4098 - val_recall: 0.4423
Epoch 86/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2553 - recall: 0.6083 -
val_loss: 0.4042 - val_recall: 0.4742
Epoch 87/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2531 - recall: 0.6206 -
val_loss: 0.4021 - val_recall: 0.5160
Epoch 88/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2539 - recall: 0.6092 -
val_loss: 0.3978 - val_recall: 0.4767
Epoch 89/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2503 - recall: 0.6173 -
val_loss: 0.4059 - val_recall: 0.5405
Epoch 90/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2551 - recall: 0.6034 -
val_loss: 0.4040 - val_recall: 0.4496
Epoch 91/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2536 - recall: 0.6247 -
val_loss: 0.4118 - val_recall: 0.5528
Epoch 92/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2496 - recall: 0.6345 -
val_loss: 0.4102 - val_recall: 0.4693
Epoch 93/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2494 - recall: 0.6108 -
val_loss: 0.4160 - val_recall: 0.4324
Epoch 94/100
188/188 [==============================] - 0s 3ms/step - loss: 0.2460 - recall: 0.6361 -
val_loss: 0.4103 - val_recall: 0.4742
Epoch 95/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2463 - recall: 0.6247 -
val_loss: 0.4275 - val_recall: 0.4349
Epoch 96/100
188/188 [==============================] - 0s 2ms/step - loss: 0.2454 - recall: 0.6198 -
val_loss: 0.4058 - val_recall: 0.4988
Epoch 97/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2454 - recall: 0.6386 -
```

```
val_loss: 0.4218 - val_recall: 0.4226
Epoch 98/100
188/188 [==============================] - 0s 2ms/step - loss: 0.2472 - recall: 0.6247 -
val_loss: 0.4086 - val_recall: 0.5233
Epoch 99/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2439 - recall: 0.6353 -
val_loss: 0.4258 - val_recall: 0.4693
Epoch 100/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2449 - recall: 0.6386 -
val_loss: 0.4198 - val_recall: 0.4939
```

Loss function

In [54]:

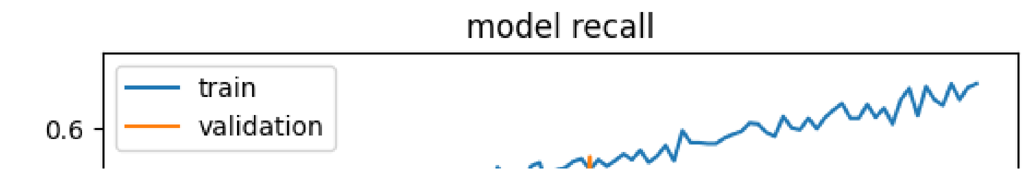
```
#Plotting Train Loss vs Validation Loss
plt.plot(history_1.history['loss'])
plt.plot(history_1.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

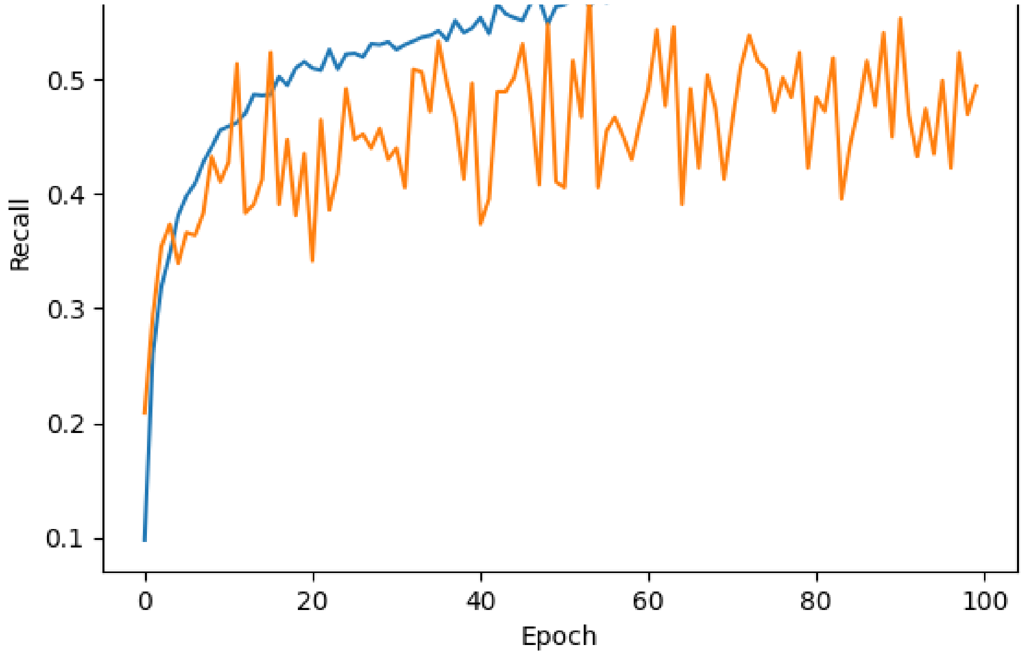


Recall

In [55]:

```
#Plotting Train recall vs Validation recall
plt.plot(history_1.history['recall'])
plt.plot(history_1.history['val_recall'])
plt.title('model recall')
plt.ylabel('Recall')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





In [56]:

```
#Predicting the results using 0.5 as the threshold
y_train_pred = model_1.predict(X_train)
y_train_pred = (y_train_pred > 0.5)
y_train_pred
```

```
188/188 [==============================] - 0s 1ms/step
```

Out[56]:

```
array([[False],
       [False],
       [ True],
       ...,
       [ True],
       [False],
       [ True]])
```

In [57]:

```
#Predicting the results using 0.5 as the threshold
y_val_pred = model_1.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
y_val_pred
```

```
63/63 [==============================] - 0s 1ms/step
```

Out[57]:

```
array([[False],
       [False],
       [ True],
       ...,
       [False],
       [False],
       [False]])
```

In [58]:

```
model_name = "NN with Adam"

train_metric_df.loc[model_name] = recall_score(y_train,y_train_pred)
valid_metric_df.loc[model_name] = recall_score(y_val,y_val_pred)

print(train_metric_df)
```

```
                recall
NN with SGD   0.219133
NN with Adam  0.678659
```

Classification report

In [59]:

```
#lassification report
cr=classification_report(y_train,y_train_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      4777
           1       0.83      0.68      0.75      1223

    accuracy                           0.91      6000
   macro avg       0.88      0.82      0.85      6000
weighted avg       0.90      0.91      0.90      6000
```

In [60]:

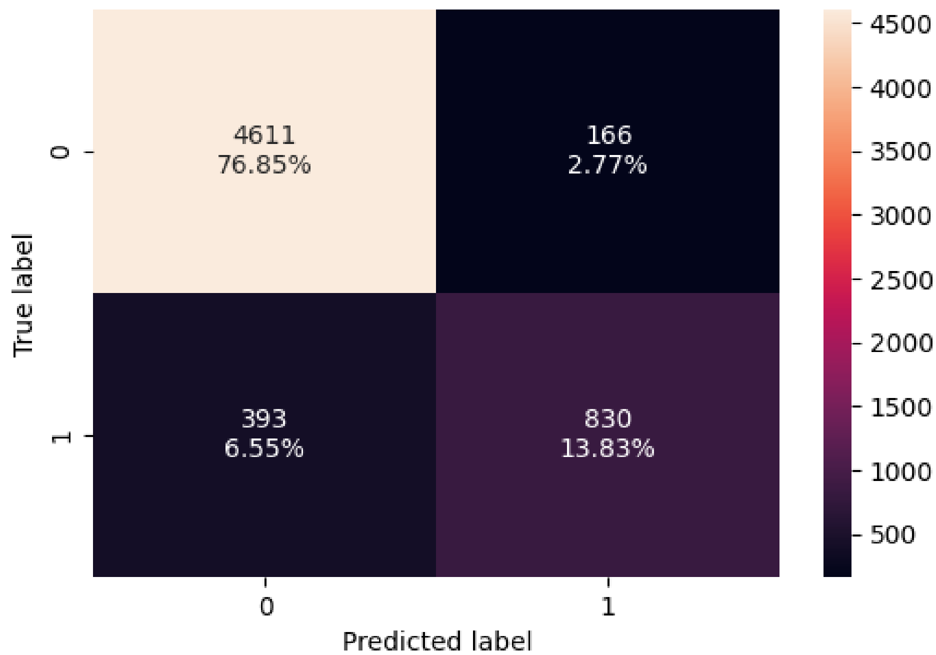
```
#classification report
cr=classification_report(y_val,y_val_pred)  ## Complete the code to check the model's per
formance on the validation set
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.91      1593
           1       0.66      0.49      0.56       407

    accuracy                           0.84      2000
   macro avg       0.77      0.71      0.74      2000
weighted avg       0.83      0.84      0.84      2000
```

Confusion matrix

In [61]:

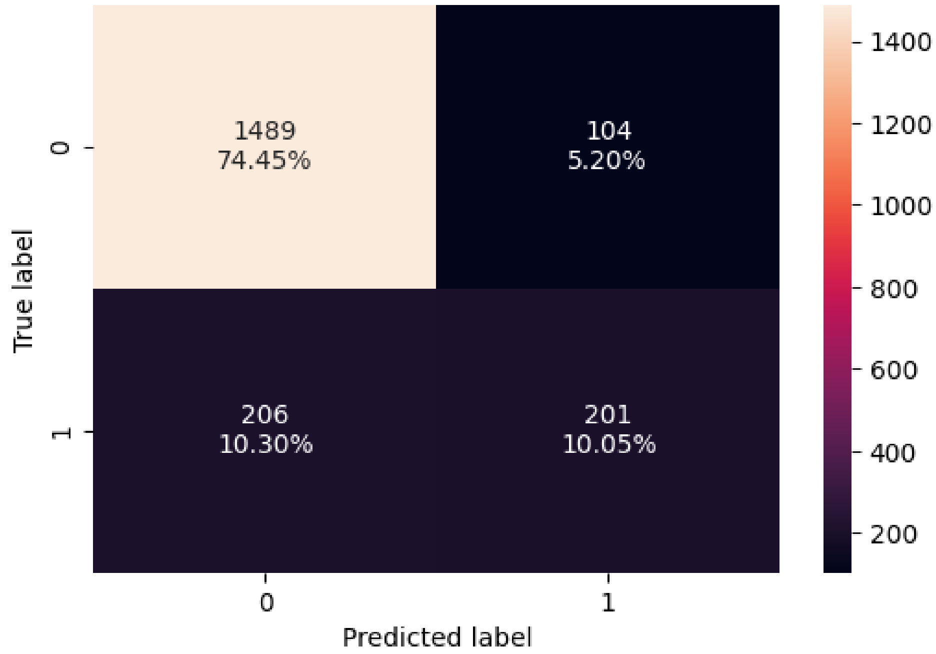
```
#Calculating the confusion matrix
make_confusion_matrix(y_train, y_train_pred)
```



In [62]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)  ## Complete the code to check the model's perfor
```

```
mance on the validation set
```



Neural Network with Adam Optimizer and Dropout

In [71]:

```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the same o
utput everytime
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [72]:

```
#Initializing the neural network
model_2 = Sequential()
#Adding the input layer with 32 neurons and relu as activation function
model_2.add(Dense(32,activation='relu',input_dim = X_train.shape[1]))
# Add dropout with ratio of 0.2 or any suitable value.
model_2.add(Dropout(0.2))
# Add a hidden layer (specify the # of neurons and the activation function)
model_2.add(Dense(64,activation='relu'))
# Add a hidden layer (specify the # of neurons and the activation function)
model_2.add(Dense(64,activation='relu'))
# Add dropout with ratio of 0.1 or any suitable value.
model_2.add(Dropout(0.1))
# Add a hidden layer (specify the # of neurons and the activation function)
model_2.add(Dense(32,activation='relu'))
# Add the number of neurons required in the output layer.
model_2.add(Dense(1, activation = 'sigmoid'))
```

In [73]:

```
#Complete the code to use Adam as the optimizer.
optimizer = tf.keras.optimizers.Adam()

# uncomment one of the following lines to define the metric to be used
# metric = 'accuracy'
metric = keras.metrics.Recall()
# metric = keras.metrics.Precision()
# metric = keras.metrics.F1Score()
```

In [74]:

```
# Complete the code to compile the model with binary cross entropy as loss function and r
ecall as the metric
```

```
model_2.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [75]:

```
model_2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 32)                384       
                                                                 
 dropout (Dropout)           (None, 32)                0         
                                                                 
 dense_1 (Dense)             (None, 64)                2112      
                                                                 
 dense_2 (Dense)             (None, 64)                4160      
                                                                 
 dropout_1 (Dropout)         (None, 64)                0         
                                                                 
 dense_3 (Dense)             (None, 32)                2080      
                                                                 
 dense_4 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 8769 (34.25 KB)
Trainable params: 8769 (34.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [76]:

```
#Fitting the ANN
history_2 = model_2.fit(
    X_train,y_train,
    batch_size=32, ## Complete the code to specify the batch size to use
    validation_data=(X_val,y_val),
    epochs=100, ## Complete the code to specify the number of epochs
    verbose=1
)
```

```
Epoch 1/100
188/188 [==============================] - 2s 5ms/step - loss: 0.4733 - recall: 0.0164 -
val_loss: 0.4366 - val_recall: 0.0270
Epoch 2/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4377 - recall: 0.1226 -
val_loss: 0.4284 - val_recall: 0.1867
Epoch 3/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4300 - recall: 0.1881 -
val_loss: 0.4179 - val_recall: 0.2776
Epoch 4/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4202 - recall: 0.2731 -
val_loss: 0.4128 - val_recall: 0.3415
Epoch 5/100
188/188 [==============================] - 1s 3ms/step - loss: 0.4094 - recall: 0.3230 -
val_loss: 0.4075 - val_recall: 0.2826
Epoch 6/100
188/188 [==============================] - 1s 5ms/step - loss: 0.4065 - recall: 0.3066 -
val_loss: 0.4080 - val_recall: 0.3538
Epoch 7/100
188/188 [==============================] - 1s 5ms/step - loss: 0.4018 - recall: 0.3246 -
val_loss: 0.3987 - val_recall: 0.3120
Epoch 8/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3962 - recall: 0.3393 -
val_loss: 0.3950 - val_recall: 0.3219
Epoch 9/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3911 - recall: 0.3500 -
val_loss: 0.3941 - val_recall: 0.3342
Epoch 10/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3917 - recall: 0.3500 -
val_loss: 0.3930 - val_recall: 0.3661
Epoch 11/100
```

```
188/188 [==============================] - 1s 3ms/step - loss: 0.3897 - recall: 0.3573 -
val_loss: 0.3885 - val_recall: 0.4226
Epoch 12/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3832 - recall: 0.3704 -
val_loss: 0.3824 - val_recall: 0.4619
Epoch 13/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3772 - recall: 0.3917 -
val_loss: 0.3744 - val_recall: 0.3980
Epoch 14/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3716 - recall: 0.4047 -
val_loss: 0.3703 - val_recall: 0.4177
Epoch 15/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3661 - recall: 0.4146 -
val_loss: 0.3649 - val_recall: 0.4079
Epoch 16/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3588 - recall: 0.4399 -
val_loss: 0.3591 - val_recall: 0.4447
Epoch 17/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3589 - recall: 0.4252 -
val_loss: 0.3620 - val_recall: 0.3759
Epoch 18/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3556 - recall: 0.4186 -
val_loss: 0.3578 - val_recall: 0.4103
Epoch 19/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3503 - recall: 0.4252 -
val_loss: 0.3585 - val_recall: 0.3563
Epoch 20/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3453 - recall: 0.4407 -
val_loss: 0.3550 - val_recall: 0.4201
Epoch 21/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3453 - recall: 0.4424 -
val_loss: 0.3577 - val_recall: 0.3563
Epoch 22/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3457 - recall: 0.4276 -
val_loss: 0.3488 - val_recall: 0.4496
Epoch 23/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3417 - recall: 0.4440 -
val_loss: 0.3558 - val_recall: 0.4349
Epoch 24/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3415 - recall: 0.4399 -
val_loss: 0.3521 - val_recall: 0.4054
Epoch 25/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3419 - recall: 0.4448 -
val_loss: 0.3507 - val_recall: 0.4619
Epoch 26/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3406 - recall: 0.4464 -
val_loss: 0.3498 - val_recall: 0.4914
Epoch 27/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3363 - recall: 0.4538 -
val_loss: 0.3491 - val_recall: 0.4373
Epoch 28/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3368 - recall: 0.4497 -
val_loss: 0.3468 - val_recall: 0.4496
Epoch 29/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3344 - recall: 0.4595 -
val_loss: 0.3437 - val_recall: 0.4521
Epoch 30/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3348 - recall: 0.4530 -
val_loss: 0.3473 - val_recall: 0.4275
Epoch 31/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3310 - recall: 0.4636 -
val_loss: 0.3475 - val_recall: 0.4693
Epoch 32/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3306 - recall: 0.4603 -
val_loss: 0.3468 - val_recall: 0.4398
Epoch 33/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3266 - recall: 0.4644 -
val_loss: 0.3503 - val_recall: 0.4595
Epoch 34/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3338 - recall: 0.4497 -
val_loss: 0.3496 - val_recall: 0.4767
Epoch 35/100
```

```
188/188 [==============================] - 1s 3ms/step - loss: 0.3313 - recall: 0.4522 -
val_loss: 0.3455 - val_recall: 0.4939
Epoch 36/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3268 - recall: 0.4620 -
val_loss: 0.3480 - val_recall: 0.4742
Epoch 37/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3302 - recall: 0.4587 -
val_loss: 0.3508 - val_recall: 0.4447
Epoch 38/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3288 - recall: 0.4669 -
val_loss: 0.3523 - val_recall: 0.4619
Epoch 39/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3239 - recall: 0.4546 -
val_loss: 0.3487 - val_recall: 0.4275
Epoch 40/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3230 - recall: 0.4636 -
val_loss: 0.3461 - val_recall: 0.4840
Epoch 41/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3280 - recall: 0.4775 -
val_loss: 0.3467 - val_recall: 0.4447
Epoch 42/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3288 - recall: 0.4546 -
val_loss: 0.3514 - val_recall: 0.4275
Epoch 43/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3258 - recall: 0.4693 -
val_loss: 0.3507 - val_recall: 0.4889
Epoch 44/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3191 - recall: 0.4857 -
val_loss: 0.3510 - val_recall: 0.4668
Epoch 45/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3218 - recall: 0.4857 -
val_loss: 0.3478 - val_recall: 0.4423
Epoch 46/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3229 - recall: 0.4685 -
val_loss: 0.3489 - val_recall: 0.4398
Epoch 47/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3200 - recall: 0.4677 -
val_loss: 0.3522 - val_recall: 0.4717
Epoch 48/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3204 - recall: 0.4783 -
val_loss: 0.3491 - val_recall: 0.4447
Epoch 49/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3214 - recall: 0.4628 -
val_loss: 0.3568 - val_recall: 0.5037
Epoch 50/100
188/188 [==============================] - 1s 6ms/step - loss: 0.3230 - recall: 0.4791 -
val_loss: 0.3528 - val_recall: 0.3882
Epoch 51/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3254 - recall: 0.4644 -
val_loss: 0.3545 - val_recall: 0.3489
Epoch 52/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3188 - recall: 0.4702 -
val_loss: 0.3494 - val_recall: 0.4742
Epoch 53/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3203 - recall: 0.4783 -
val_loss: 0.3500 - val_recall: 0.4300
Epoch 54/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3146 - recall: 0.4718 -
val_loss: 0.3616 - val_recall: 0.5209
Epoch 55/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3160 - recall: 0.4914 -
val_loss: 0.3549 - val_recall: 0.4128
Epoch 56/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3179 - recall: 0.4759 -
val_loss: 0.3517 - val_recall: 0.4373
Epoch 57/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3176 - recall: 0.4816 -
val_loss: 0.3538 - val_recall: 0.4914
Epoch 58/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3133 - recall: 0.4890 -
val_loss: 0.3516 - val_recall: 0.4079
Epoch 59/100
```

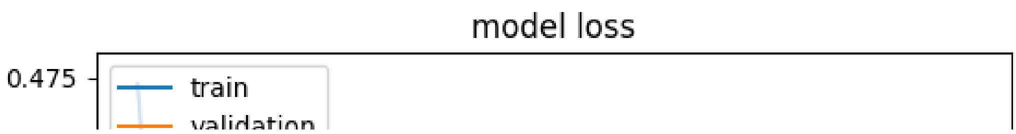
```
188/188 [==============================] - 1s 4ms/step - loss: 0.3149 - recall: 0.4816 -
val_loss: 0.3541 - val_recall: 0.4619
Epoch 60/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3153 - recall: 0.4930 -
val_loss: 0.3505 - val_recall: 0.4324
Epoch 61/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3114 - recall: 0.4849 -
val_loss: 0.3555 - val_recall: 0.4595
Epoch 62/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3145 - recall: 0.4791 -
val_loss: 0.3523 - val_recall: 0.4349
Epoch 63/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3130 - recall: 0.4783 -
val_loss: 0.3552 - val_recall: 0.4226
Epoch 64/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3132 - recall: 0.4726 -
val_loss: 0.3573 - val_recall: 0.4251
Epoch 65/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3148 - recall: 0.4783 -
val_loss: 0.3543 - val_recall: 0.4201
Epoch 66/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3121 - recall: 0.4849 -
val_loss: 0.3508 - val_recall: 0.4103
Epoch 67/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3166 - recall: 0.4685 -
val_loss: 0.3542 - val_recall: 0.4496
Epoch 68/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3089 - recall: 0.4783 -
val_loss: 0.3593 - val_recall: 0.4226
Epoch 69/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3121 - recall: 0.4890 -
val_loss: 0.3513 - val_recall: 0.4275
Epoch 70/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3065 - recall: 0.4898 -
val_loss: 0.3584 - val_recall: 0.3882
Epoch 71/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3052 - recall: 0.4922 -
val_loss: 0.3588 - val_recall: 0.4275
Epoch 72/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3054 - recall: 0.4955 -
val_loss: 0.3592 - val_recall: 0.3686
Epoch 73/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3071 - recall: 0.5029 -
val_loss: 0.3559 - val_recall: 0.4644
Epoch 74/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3052 - recall: 0.4922 -
val_loss: 0.3617 - val_recall: 0.4521
Epoch 75/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3049 - recall: 0.4963 -
val_loss: 0.3682 - val_recall: 0.4939
Epoch 76/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3073 - recall: 0.5061 -
val_loss: 0.3618 - val_recall: 0.4079
Epoch 77/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3049 - recall: 0.4857 -
val_loss: 0.3635 - val_recall: 0.4717
Epoch 78/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3050 - recall: 0.5061 -
val_loss: 0.3649 - val_recall: 0.3931
Epoch 79/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3009 - recall: 0.4963 -
val_loss: 0.3650 - val_recall: 0.4619
Epoch 80/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3092 - recall: 0.5012 -
val_loss: 0.3583 - val_recall: 0.4668
Epoch 81/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3005 - recall: 0.5045 -
val_loss: 0.3631 - val_recall: 0.3931
Epoch 82/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3037 - recall: 0.4980 -
val_loss: 0.3705 - val_recall: 0.5258
Epoch 83/100
```

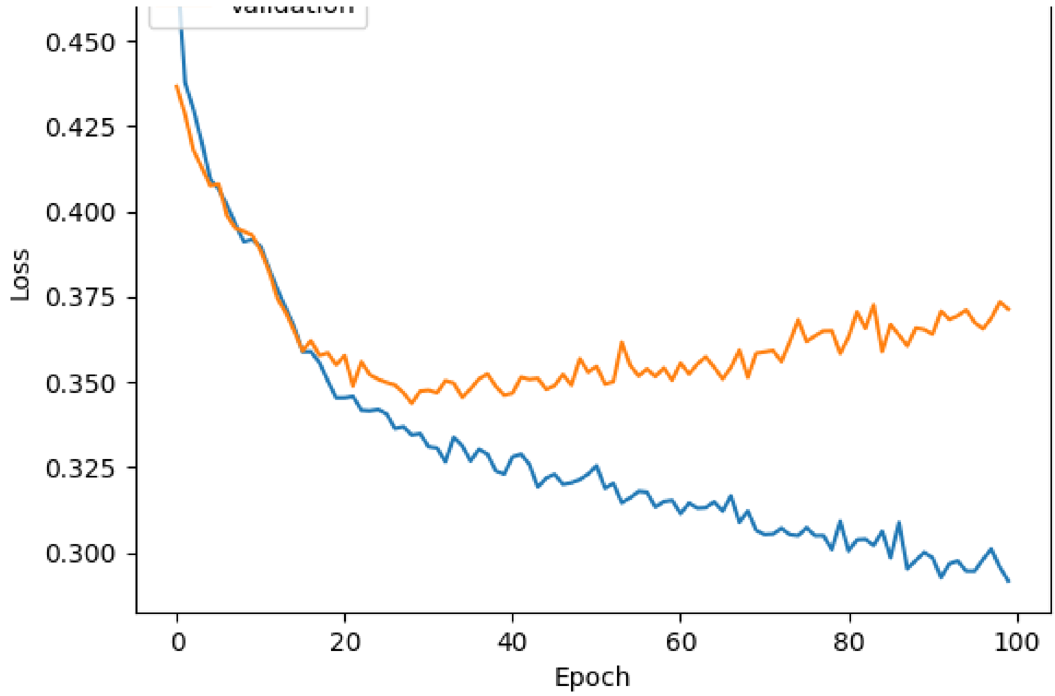
```
188/188 [==============================] - 1s 3ms/step - loss: 0.3039 - recall: 0.4963 -
val_loss: 0.3657 - val_recall: 0.4644
Epoch 84/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3021 - recall: 0.5143 -
val_loss: 0.3726 - val_recall: 0.3931
Epoch 85/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3062 - recall: 0.4930 -
val_loss: 0.3589 - val_recall: 0.4373
Epoch 86/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2985 - recall: 0.5086 -
val_loss: 0.3668 - val_recall: 0.4865
Epoch 87/100
188/188 [==============================] - 1s 4ms/step - loss: 0.3088 - recall: 0.5045 -
val_loss: 0.3637 - val_recall: 0.4177
Epoch 88/100
188/188 [==============================] - 1s 5ms/step - loss: 0.2952 - recall: 0.5102 -
val_loss: 0.3605 - val_recall: 0.3931
Epoch 89/100
188/188 [==============================] - 1s 5ms/step - loss: 0.2977 - recall: 0.4939 -
val_loss: 0.3658 - val_recall: 0.4742
Epoch 90/100
188/188 [==============================] - 1s 5ms/step - loss: 0.3000 - recall: 0.5070 -
val_loss: 0.3654 - val_recall: 0.4103
Epoch 91/100
188/188 [==============================] - 1s 4ms/step - loss: 0.2985 - recall: 0.5102 -
val_loss: 0.3639 - val_recall: 0.4447
Epoch 92/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2926 - recall: 0.5127 -
val_loss: 0.3707 - val_recall: 0.4840
Epoch 93/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2967 - recall: 0.5200 -
val_loss: 0.3683 - val_recall: 0.4742
Epoch 94/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2975 - recall: 0.5110 -
val_loss: 0.3693 - val_recall: 0.4521
Epoch 95/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2945 - recall: 0.5217 -
val_loss: 0.3711 - val_recall: 0.4275
Epoch 96/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2945 - recall: 0.5127 -
val_loss: 0.3674 - val_recall: 0.4496
Epoch 97/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2980 - recall: 0.5184 -
val_loss: 0.3656 - val_recall: 0.4103
Epoch 98/100
188/188 [==============================] - 1s 3ms/step - loss: 0.3010 - recall: 0.5045 -
val_loss: 0.3686 - val_recall: 0.4521
Epoch 99/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2957 - recall: 0.5249 -
val_loss: 0.3734 - val_recall: 0.4029
Epoch 100/100
188/188 [==============================] - 1s 3ms/step - loss: 0.2917 - recall: 0.5233 -
val_loss: 0.3713 - val_recall: 0.4373
```

Loss function

In [77]:

```
#Plotting Train Loss vs Validation Loss
plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

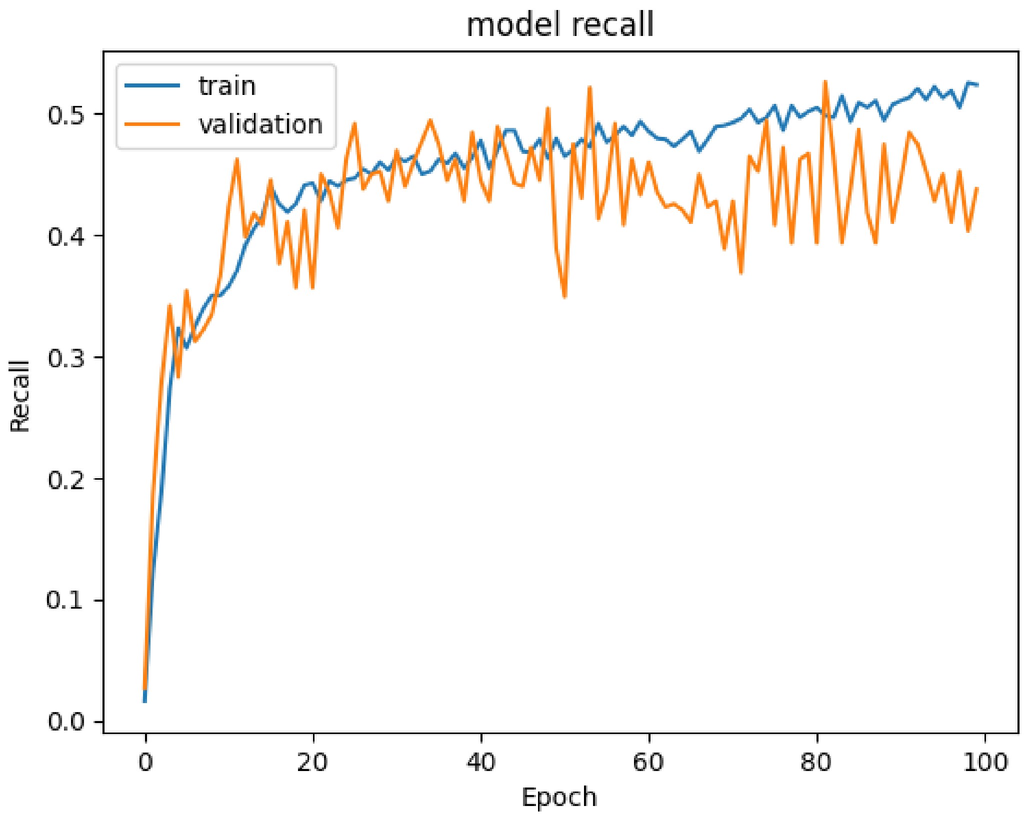




Recall

In [78]:

```
#Plotting Train recall vs Validation recall
plt.plot(history_2.history['recall'])
plt.plot(history_2.history['val_recall'])
plt.title('model recall')
plt.ylabel('Recall')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



In [79]:

```
#Predicting the results using 0.5 as the threshold
y_train_pred = model_2.predict(X_train)
y_train_pred = (y_train_pred > 0.5)
y_train_pred
```

```
188/188 [==============================] - 1s 2ms/step
```

Out[79]:

```
array([[False],
       [False],
       [ True],
       ...,
       [ True],
       [False],
       [ True]])
```

In [80]:

```
#Predicting the results using 0.5 as the threshold
y_val_pred = model_2.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
y_val_pred
```

```
63/63 [==============================] - 0s 2ms/step
```

Out[80]:

```
array([[False],
       [False],
       [ True],
       ...,
       [False],
       [False],
       [False]])
```

In [81]:

```
model_name = "NN with Adam DropOut"

train_metric_df.loc[model_name] = recall_score(y_train,y_train_pred)
valid_metric_df.loc[model_name] = recall_score(y_val,y_val_pred)

print(train_metric_df)
```

```
                        recall
NN with SGD           0.219133
NN with Adam          0.678659
NN with Adam DropOut  0.538021
```

Classification report

In [82]:

```
#lassification report
cr=classification_report(y_train,y_train_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.89      0.98      0.94      4777
           1       0.89      0.54      0.67      1223

    accuracy                           0.89      6000
   macro avg       0.89      0.76      0.80      6000
weighted avg       0.89      0.89      0.88      6000
```

In [83]:

```
#classification report
cr=classification_report(y_val,y_val_pred)  ## Complete the code to check the model's per
formance on the validation set
print(cr)
```

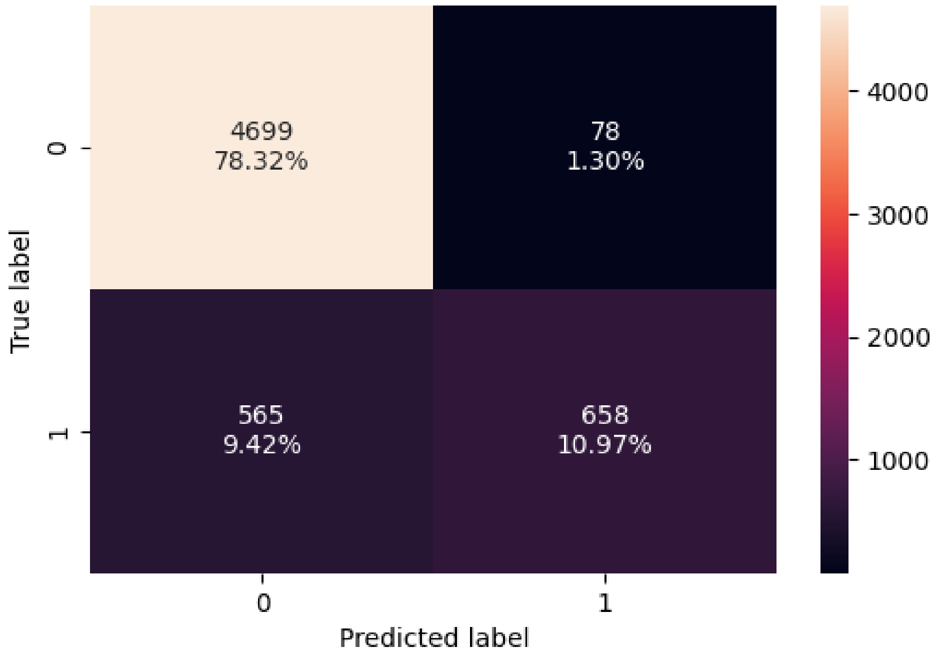
```
              precision    recall  f1-score   support
```

```
              0       0.87      0.96      0.91      1593
              1       0.75      0.44      0.55       407

       accuracy                           0.86      2000
      macro avg       0.81      0.70      0.73      2000
   weighted avg       0.85      0.86      0.84      2000
```

Confusion matrix

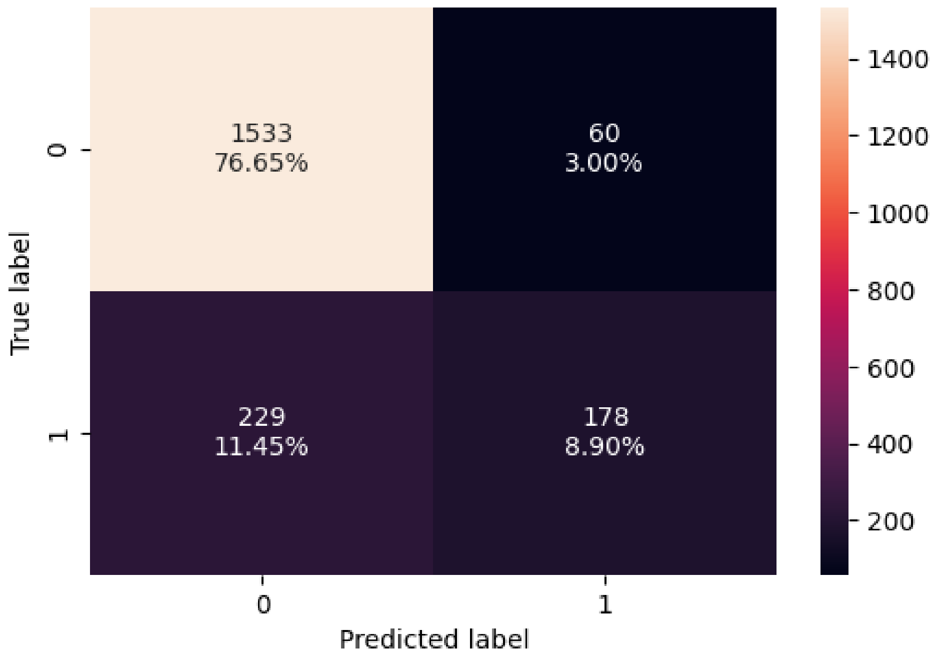
In [84]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_train, y_train_pred)
```



In [85]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)   ## Complete the code to check the model's perfor
mance on the validation set
```



Observations-

Neural Network with Balanced Data (by applying SMOTE) and SGD Optimizer

Let's try to apply SMOTE to balance this dataset and then again apply hyperparamter tuning accordingly.

In [86]:

```
sm  = SMOTE(random_state=42)
#Fit SMOTE on the training data.
X_train_smote, y_train_smote= sm.fit_resample(X_train,y_train)
print('After UpSampling, the shape of train_X: {}'.format(X_train_smote.shape))
print('After UpSampling, the shape of train_y: {} \n'.format(y_train_smote.shape))
```

```
After UpSampling, the shape of train_X: (9554, 11)
After UpSampling, the shape of train_y: (9554,)
```

In [87]:

```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the same o
utput everytime
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [88]:

```
#Initializing the neural network
model_3 = Sequential()
# Adding the input layer with 32 neurons and relu as activation function
model_3.add(Dense(32,activation='relu',input_dim = X_train_smote.shape[1]))
# Add a hidden layer (specify the # of neurons and the activation function)
model_3.add(Dense(16,activation='relu'))
# Add a hidden layer (specify the # of neurons and the activation function)
model_3.add(Dense(8,activation='relu'))
# Add a hidden layer (specify the # of neurons and the activation function)
model_3.add(Dense(1, activation = 'sigmoid'))
```

In [89]:

```
#Complete the code to use Adam as the optimizer.
optimizer = tf.keras.optimizers.SGD(0.001)

# uncomment one of the following lines to define the metric to be used
# metric = 'accuracy'
metric = keras.metrics.Recall()
# metric = keras.metrics.Precision()
# metric = keras.metrics.F1Score()
```

In [90]:

```
# Complete the code to compile the model with binary cross entropy as loss function and r
ecall as the metric
model_3.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [91]:

```
model_3.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                384

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 8)                 136

 dense_3 (Dense)             (None, 1)                 9
```

```
=================================================================
Total params: 1057 (4.13 KB)
Trainable params: 1057 (4.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [92]:

```python
#Fitting the ANN
history_3 = model_3.fit(
    X_train_smote,y_train_smote,
    batch_size=32,
    validation_data=(X_val,y_val),
    epochs=100,
    verbose=1
)
```

```
Epoch 1/100
299/299 [==============================] - 2s 3ms/step - loss: 0.7034 - recall: 0.9376 -
val_loss: 0.7339 - val_recall: 0.9189
Epoch 2/100
299/299 [==============================] - 1s 3ms/step - loss: 0.6998 - recall: 0.9288 -
val_loss: 0.7241 - val_recall: 0.9115
Epoch 3/100
299/299 [==============================] - 1s 3ms/step - loss: 0.6968 - recall: 0.9150 -
val_loss: 0.7158 - val_recall: 0.8943
Epoch 4/100
299/299 [==============================] - 1s 4ms/step - loss: 0.6941 - recall: 0.8882 -
val_loss: 0.7076 - val_recall: 0.8698
Epoch 5/100
299/299 [==============================] - 1s 3ms/step - loss: 0.6910 - recall: 0.8321 -
val_loss: 0.6979 - val_recall: 0.7740
Epoch 6/100
299/299 [==============================] - 1s 3ms/step - loss: 0.6867 - recall: 0.7593 -
val_loss: 0.6862 - val_recall: 0.7076
Epoch 7/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6816 - recall: 0.7017 -
val_loss: 0.6744 - val_recall: 0.6536
Epoch 8/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6767 - recall: 0.6762 -
val_loss: 0.6642 - val_recall: 0.6192
Epoch 9/100
299/299 [==============================] - 1s 3ms/step - loss: 0.6723 - recall: 0.6556 -
val_loss: 0.6551 - val_recall: 0.5995
Epoch 10/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6680 - recall: 0.6492 -
val_loss: 0.6466 - val_recall: 0.5921
Epoch 11/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6638 - recall: 0.6339 -
val_loss: 0.6394 - val_recall: 0.5872
Epoch 12/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6597 - recall: 0.6362 -
val_loss: 0.6326 - val_recall: 0.5897
Epoch 13/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6557 - recall: 0.6320 -
val_loss: 0.6265 - val_recall: 0.5749
Epoch 14/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6517 - recall: 0.6293 -
val_loss: 0.6211 - val_recall: 0.5725
Epoch 15/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6478 - recall: 0.6255 -
val_loss: 0.6162 - val_recall: 0.5823
Epoch 16/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6439 - recall: 0.6295 -
val_loss: 0.6114 - val_recall: 0.5700
Epoch 17/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6400 - recall: 0.6263 -
val_loss: 0.6077 - val_recall: 0.5700
Epoch 18/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6361 - recall: 0.6376 -
val_loss: 0.6037 - val_recall: 0.5749
Epoch 19/100
```

```
Epoch 19/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6321 - recall: 0.6404 -
val_loss: 0.6003 - val_recall: 0.5749
Epoch 20/100
299/299 [==============================] - 1s 4ms/step - loss: 0.6281 - recall: 0.6454 -
val_loss: 0.5972 - val_recall: 0.5774
Epoch 21/100
299/299 [==============================] - 1s 4ms/step - loss: 0.6241 - recall: 0.6525 -
val_loss: 0.5944 - val_recall: 0.5823
Epoch 22/100
299/299 [==============================] - 1s 3ms/step - loss: 0.6201 - recall: 0.6584 -
val_loss: 0.5918 - val_recall: 0.5946
Epoch 23/100
299/299 [==============================] - 1s 3ms/step - loss: 0.6161 - recall: 0.6688 -
val_loss: 0.5887 - val_recall: 0.6044
Epoch 24/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6120 - recall: 0.6741 -
val_loss: 0.5856 - val_recall: 0.6044
Epoch 25/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6080 - recall: 0.6757 -
val_loss: 0.5835 - val_recall: 0.6069
Epoch 26/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6039 - recall: 0.6885 -
val_loss: 0.5802 - val_recall: 0.6118
Epoch 27/100
299/299 [==============================] - 1s 2ms/step - loss: 0.6000 - recall: 0.6847 -
val_loss: 0.5785 - val_recall: 0.6143
Epoch 28/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5961 - recall: 0.6944 -
val_loss: 0.5759 - val_recall: 0.6192
Epoch 29/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5923 - recall: 0.6977 -
val_loss: 0.5733 - val_recall: 0.6265
Epoch 30/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5887 - recall: 0.6971 -
val_loss: 0.5723 - val_recall: 0.6290
Epoch 31/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5851 - recall: 0.7048 -
val_loss: 0.5704 - val_recall: 0.6290
Epoch 32/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5817 - recall: 0.7036 -
val_loss: 0.5696 - val_recall: 0.6339
Epoch 33/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5784 - recall: 0.7122 -
val_loss: 0.5671 - val_recall: 0.6388
Epoch 34/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5753 - recall: 0.7109 -
val_loss: 0.5661 - val_recall: 0.6364
Epoch 35/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5724 - recall: 0.7138 -
val_loss: 0.5645 - val_recall: 0.6413
Epoch 36/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5696 - recall: 0.7172 -
val_loss: 0.5630 - val_recall: 0.6486
Epoch 37/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5671 - recall: 0.7184 -
val_loss: 0.5624 - val_recall: 0.6413
Epoch 38/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5647 - recall: 0.7214 -
val_loss: 0.5624 - val_recall: 0.6437
Epoch 39/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5625 - recall: 0.7277 -
val_loss: 0.5608 - val_recall: 0.6462
Epoch 40/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5605 - recall: 0.7285 -
val_loss: 0.5608 - val_recall: 0.6486
Epoch 41/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5586 - recall: 0.7320 -
val_loss: 0.5595 - val_recall: 0.6486
Epoch 42/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5569 - recall: 0.7377 -
val_loss: 0.5575 - val_recall: 0.6511
Epoch 43/100
```

```
Epoch 43/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5553 - recall: 0.7348 -
val_loss: 0.5590 - val_recall: 0.6585
Epoch 44/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5538 - recall: 0.7404 -
val_loss: 0.5570 - val_recall: 0.6585
Epoch 45/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5524 - recall: 0.7415 -
val_loss: 0.5567 - val_recall: 0.6511
Epoch 46/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5511 - recall: 0.7379 -
val_loss: 0.5582 - val_recall: 0.6560
Epoch 47/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5500 - recall: 0.7429 -
val_loss: 0.5565 - val_recall: 0.6609
Epoch 48/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5489 - recall: 0.7421 -
val_loss: 0.5566 - val_recall: 0.6634
Epoch 49/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5479 - recall: 0.7415 -
val_loss: 0.5563 - val_recall: 0.6658
Epoch 50/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5469 - recall: 0.7425 -
val_loss: 0.5557 - val_recall: 0.6634
Epoch 51/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5460 - recall: 0.7450 -
val_loss: 0.5537 - val_recall: 0.6609
Epoch 52/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5452 - recall: 0.7425 -
val_loss: 0.5543 - val_recall: 0.6658
Epoch 53/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5444 - recall: 0.7444 -
val_loss: 0.5545 - val_recall: 0.6658
Epoch 54/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5437 - recall: 0.7473 -
val_loss: 0.5540 - val_recall: 0.6658
Epoch 55/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5430 - recall: 0.7492 -
val_loss: 0.5523 - val_recall: 0.6658
Epoch 56/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5423 - recall: 0.7477 -
val_loss: 0.5532 - val_recall: 0.6658
Epoch 57/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5417 - recall: 0.7469 -
val_loss: 0.5549 - val_recall: 0.6708
Epoch 58/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5411 - recall: 0.7509 -
val_loss: 0.5538 - val_recall: 0.6732
Epoch 59/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5405 - recall: 0.7511 -
val_loss: 0.5538 - val_recall: 0.6732
Epoch 60/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5399 - recall: 0.7517 -
val_loss: 0.5537 - val_recall: 0.6732
Epoch 61/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5393 - recall: 0.7498 -
val_loss: 0.5560 - val_recall: 0.6757
Epoch 62/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5388 - recall: 0.7555 -
val_loss: 0.5520 - val_recall: 0.6609
Epoch 63/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5383 - recall: 0.7507 -
val_loss: 0.5536 - val_recall: 0.6708
Epoch 64/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5378 - recall: 0.7547 -
val_loss: 0.5515 - val_recall: 0.6609
Epoch 65/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5373 - recall: 0.7524 -
val_loss: 0.5515 - val_recall: 0.6634
Epoch 66/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5368 - recall: 0.7526 -
val_loss: 0.5530 - val_recall: 0.6683
Epoch 67/100
```

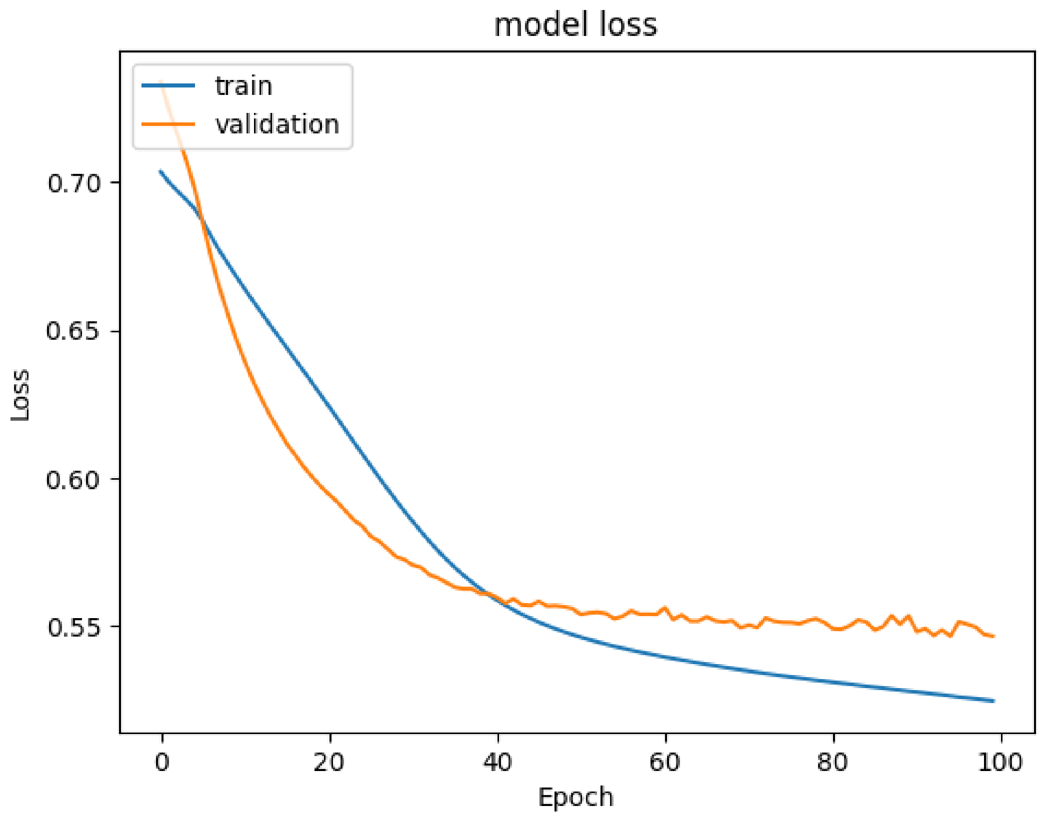
```
Epoch 67/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5364 - recall: 0.7563 -
val_loss: 0.5516 - val_recall: 0.6609
Epoch 68/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5359 - recall: 0.7563 -
val_loss: 0.5512 - val_recall: 0.6609
Epoch 69/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5355 - recall: 0.7549 -
val_loss: 0.5517 - val_recall: 0.6609
Epoch 70/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5350 - recall: 0.7576 -
val_loss: 0.5492 - val_recall: 0.6609
Epoch 71/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5346 - recall: 0.7561 -
val_loss: 0.5502 - val_recall: 0.6609
Epoch 72/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5342 - recall: 0.7586 -
val_loss: 0.5493 - val_recall: 0.6609
Epoch 73/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5338 - recall: 0.7536 -
val_loss: 0.5525 - val_recall: 0.6634
Epoch 74/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5334 - recall: 0.7578 -
val_loss: 0.5515 - val_recall: 0.6609
Epoch 75/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5330 - recall: 0.7580 -
val_loss: 0.5511 - val_recall: 0.6634
Epoch 76/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5326 - recall: 0.7578 -
val_loss: 0.5510 - val_recall: 0.6609
Epoch 77/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5322 - recall: 0.7586 -
val_loss: 0.5506 - val_recall: 0.6609
Epoch 78/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5319 - recall: 0.7578 -
val_loss: 0.5516 - val_recall: 0.6609
Epoch 79/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5315 - recall: 0.7586 -
val_loss: 0.5523 - val_recall: 0.6609
Epoch 80/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5312 - recall: 0.7597 -
val_loss: 0.5510 - val_recall: 0.6609
Epoch 81/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5308 - recall: 0.7607 -
val_loss: 0.5489 - val_recall: 0.6585
Epoch 82/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5305 - recall: 0.7588 -
val_loss: 0.5487 - val_recall: 0.6585
Epoch 83/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5302 - recall: 0.7576 -
val_loss: 0.5499 - val_recall: 0.6609
Epoch 84/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5298 - recall: 0.7580 -
val_loss: 0.5519 - val_recall: 0.6658
Epoch 85/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5295 - recall: 0.7597 -
val_loss: 0.5512 - val_recall: 0.6634
Epoch 86/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5291 - recall: 0.7605 -
val_loss: 0.5485 - val_recall: 0.6634
Epoch 87/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5288 - recall: 0.7582 -
val_loss: 0.5497 - val_recall: 0.6634
Epoch 88/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5284 - recall: 0.7549 -
val_loss: 0.5534 - val_recall: 0.6708
Epoch 89/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5281 - recall: 0.7614 -
val_loss: 0.5504 - val_recall: 0.6658
Epoch 90/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5278 - recall: 0.7561 -
val_loss: 0.5534 - val_recall: 0.6683
Epoch 91/100
```

```
Epoch 91/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5275 - recall: 0.7620 -
val_loss: 0.5479 - val_recall: 0.6634
Epoch 92/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5272 - recall: 0.7588 -
val_loss: 0.5490 - val_recall: 0.6658
Epoch 93/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5268 - recall: 0.7605 -
val_loss: 0.5466 - val_recall: 0.6585
Epoch 94/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5265 - recall: 0.7576 -
val_loss: 0.5485 - val_recall: 0.6634
Epoch 95/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5262 - recall: 0.7595 -
val_loss: 0.5464 - val_recall: 0.6585
Epoch 96/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5258 - recall: 0.7555 -
val_loss: 0.5513 - val_recall: 0.6658
Epoch 97/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5255 - recall: 0.7599 -
val_loss: 0.5504 - val_recall: 0.6634
Epoch 98/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5252 - recall: 0.7599 -
val_loss: 0.5495 - val_recall: 0.6634
Epoch 99/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5249 - recall: 0.7599 -
val_loss: 0.5470 - val_recall: 0.6609
Epoch 100/100
299/299 [==============================] - 1s 2ms/step - loss: 0.5245 - recall: 0.7599 -
val_loss: 0.5464 - val_recall: 0.6609
```

Loss function

In [93]:

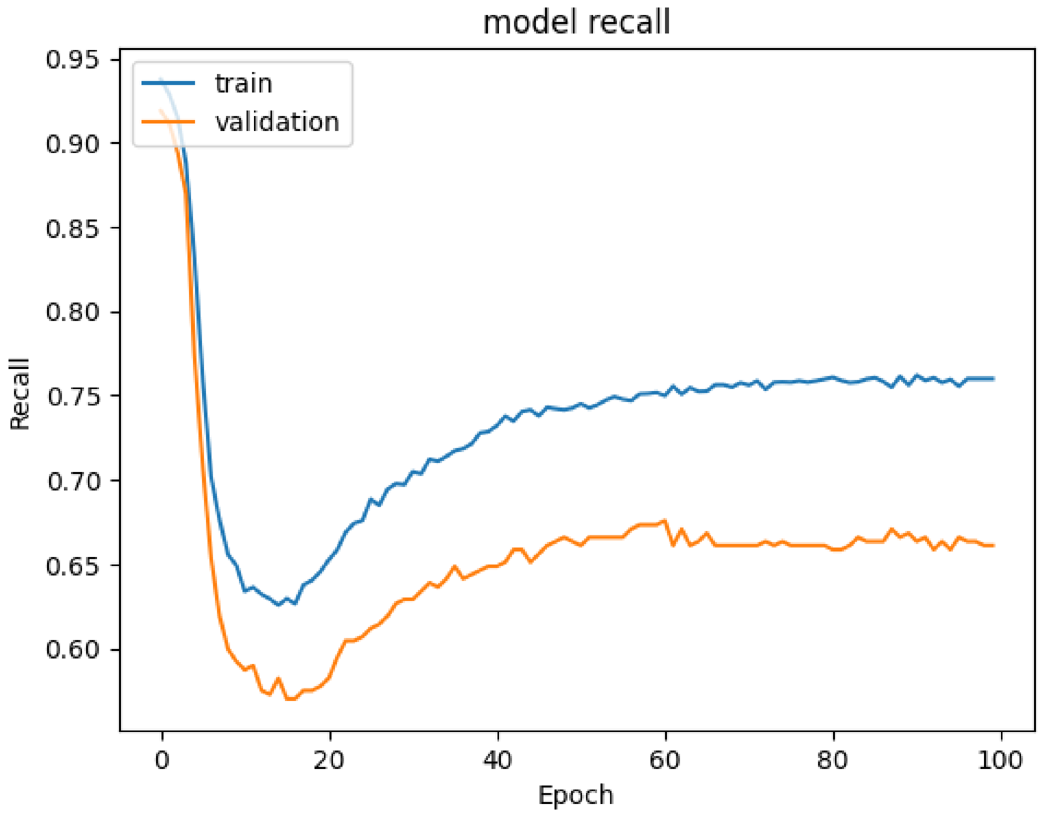
```
#Plotting Train Loss vs Validation Loss
plt.plot(history_3.history['loss'])
plt.plot(history_3.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



Recall

In [94]:

```
#Plotting Train recall vs Validation recall
plt.plot(history_3.history['recall'])
plt.plot(history_3.history['val_recall'])
plt.title('model recall')
plt.ylabel('Recall')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



In [95]:

```
#Predicting the results using 0.5 as the threshold
y_train_pred = model_3.predict(X_train_smote)
y_train_pred = (y_train_pred > 0.5)
y_train_pred
```

```
299/299 [==============================] - 0s 1ms/step
```

Out[95]:

```
array([[ True],
       [ True],
       [False],
       ...,
       [ True],
       [False],
       [ True]])
```

In [96]:

```
#Predicting the results using 0.5 as the threshold
y_val_pred = model_3.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
y_val_pred
```

```
63/63 [==============================] - 0s 1ms/step
```

Out[96]:

```
array([[False],
```

```
        [False],
        [ True],
        ...,
        [False],
        [False],
        [ True]])
```

In [97]:

```
model_name = "NN with SGD Smote "

train_metric_df.loc[model_name] = recall_score(y_train_smote,y_train_pred)
valid_metric_df.loc[model_name] = recall_score(y_val,y_val_pred)

print(train_metric_df)
```

```
                          recall
NN with SGD             0.219133
NN with Adam            0.678659
NN with Adam DropOut    0.538021
NN with SGD Smote       0.757379
```

Classification report

In [98]:

```
#lassification report
cr=classification_report(y_train_smote,y_train_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.75      0.73      0.74      4777
           1       0.74      0.76      0.75      4777

    accuracy                           0.75      9554
   macro avg       0.75      0.75      0.75      9554
weighted avg       0.75      0.75      0.75      9554
```

In [99]:

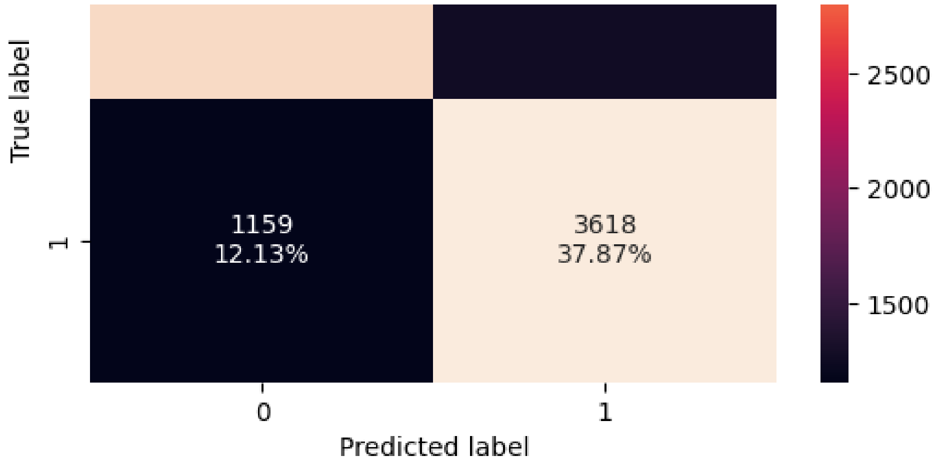
```
#classification report
cr=classification_report(y_val,y_val_pred)  ## Complete the code to check the model's per
formance on the validation set
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.89      0.73      0.81      1593
           1       0.39      0.66      0.49       407

    accuracy                           0.72      2000
   macro avg       0.64      0.70      0.65      2000
weighted avg       0.79      0.72      0.74      2000
```

Confusion matrix

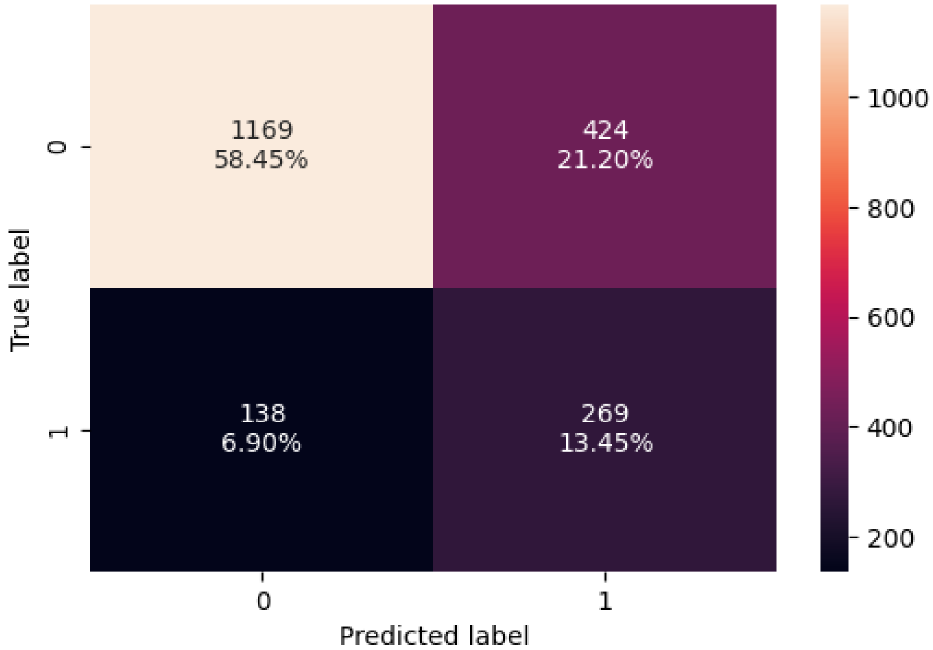
In [100]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_train_smote, y_train_pred)
```



In [101]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)  ## Complete the code to check the model's perfor
mance on the validation set
```



Neural Network with Balanced Data (by applying SMOTE) and Adam Optimizer

In [102]:

```
#sm  = SMOTE(random_state=42)
#Fit SMOTE on the training data.
#X_train_smote, y_train_smote= sm.fit_resample(X_train,y_train)
print('After UpSampling, the shape of train_X: {}'.format(X_train_smote.shape))
print('After UpSampling, the shape of train_y: {} \n'.format(y_train_smote.shape))
```

```
After UpSampling, the shape of train_X: (9554, 11)
After UpSampling, the shape of train_y: (9554,)
```

In [103]:

```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the same o
utput everytime
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [104]:

```
#Initializing the neural network
model_4 = Sequential()
# Adding the input layer with 32 neurons and relu as activation function
model_4.add(Dense(32,activation='relu',input_dim = X_train_smote.shape[1]))
# Add a hidden layer (specify the # of neurons and the activation function)
model_4.add(Dense(16,activation='relu'))
# Add a hidden layer (specify the # of neurons and the activation function)
model_4.add(Dense(8,activation='relu'))
# Add a hidden layer (specify the # of neurons and the activation function)
model_4.add(Dense(1, activation = 'sigmoid'))
```

In [105]:

```
#Complete the code to use Adam as the optimizer.
optimizer = tf.keras.optimizers.Adam()

# uncomment one of the following lines to define the metric to be used
# metric = 'accuracy'
metric = keras.metrics.Recall()
# metric = keras.metrics.Precision()
# metric = keras.metrics.F1Score()
```

In [106]:

```
# Complete the code to compile the model with binary cross entropy as loss function and r
ecall as the metric
model_4.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [107]:

```
model_4.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                384

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 8)                 136

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 1057 (4.13 KB)
Trainable params: 1057 (4.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [108]:

```
#Fitting the ANN
history_4 = model_4.fit(
    X_train_smote,y_train_smote,
    batch_size=32,
    validation_data=(X_val,y_val),
    epochs=100,
    verbose=1
)
```

```
Epoch 1/100
299/299 [==============================] - 2s 3ms/step - loss: 0.5888 - recall: 0.7115 -
val_loss: 0.6278 - val_recall: 0.7420
Epoch 2/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5260 - recall: 0.7582 -
val_loss: 0.6116 - val_recall: 0.7273
Epoch 3/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5055 - recall: 0.7691 -
val_loss: 0.5205 - val_recall: 0.6757
Epoch 4/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4802 - recall: 0.7819 -
```

```
val_loss: 0.5008 - val_recall: 0.6462
Epoch 5/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4628 - recall: 0.7900 -
val_loss: 0.4830 - val_recall: 0.6609
Epoch 6/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4472 - recall: 0.8053 -
val_loss: 0.4926 - val_recall: 0.6880
Epoch 7/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4352 - recall: 0.8066 -
val_loss: 0.4928 - val_recall: 0.7174
Epoch 8/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4267 - recall: 0.8133 -
val_loss: 0.4440 - val_recall: 0.6585
Epoch 9/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4173 - recall: 0.8137 -
val_loss: 0.5155 - val_recall: 0.7518
Epoch 10/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4103 - recall: 0.8177 -
val_loss: 0.5051 - val_recall: 0.7592
Epoch 11/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4040 - recall: 0.8231 -
val_loss: 0.4731 - val_recall: 0.7101
Epoch 12/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4011 - recall: 0.8191 -
val_loss: 0.4889 - val_recall: 0.7248
Epoch 13/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3978 - recall: 0.8214 -
val_loss: 0.4997 - val_recall: 0.7592
Epoch 14/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3930 - recall: 0.8304 -
val_loss: 0.4470 - val_recall: 0.6634
Epoch 15/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3905 - recall: 0.8246 -
val_loss: 0.4840 - val_recall: 0.7371
Epoch 16/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3872 - recall: 0.8304 -
val_loss: 0.4579 - val_recall: 0.7076
Epoch 17/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3840 - recall: 0.8286 -
val_loss: 0.4513 - val_recall: 0.6806
Epoch 18/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3818 - recall: 0.8260 -
val_loss: 0.4319 - val_recall: 0.6634
Epoch 19/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3805 - recall: 0.8304 -
val_loss: 0.4375 - val_recall: 0.6486
Epoch 20/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3770 - recall: 0.8371 -
val_loss: 0.4515 - val_recall: 0.7002
Epoch 21/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3763 - recall: 0.8346 -
val_loss: 0.4136 - val_recall: 0.6216
Epoch 22/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3755 - recall: 0.8359 -
val_loss: 0.4593 - val_recall: 0.6978
Epoch 23/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3718 - recall: 0.8367 -
val_loss: 0.4941 - val_recall: 0.7346
Epoch 24/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3717 - recall: 0.8382 -
val_loss: 0.4607 - val_recall: 0.6683
Epoch 25/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3695 - recall: 0.8388 -
val_loss: 0.4350 - val_recall: 0.6536
Epoch 26/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3668 - recall: 0.8396 -
val_loss: 0.4283 - val_recall: 0.6462
Epoch 27/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3668 - recall: 0.8363 -
val_loss: 0.4375 - val_recall: 0.6634
Epoch 28/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3657 - recall: 0.8380 -
```

```
val_loss: 0.4743 - val_recall: 0.7150
Epoch 29/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3643 - recall: 0.8415 -
val_loss: 0.4412 - val_recall: 0.6683
Epoch 30/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3617 - recall: 0.8420 -
val_loss: 0.4399 - val_recall: 0.6609
Epoch 31/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3614 - recall: 0.8415 -
val_loss: 0.4685 - val_recall: 0.7027
Epoch 32/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3597 - recall: 0.8440 -
val_loss: 0.4580 - val_recall: 0.6830
Epoch 33/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3580 - recall: 0.8484 -
val_loss: 0.4250 - val_recall: 0.6339
Epoch 34/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3578 - recall: 0.8424 -
val_loss: 0.4645 - val_recall: 0.7002
Epoch 35/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3577 - recall: 0.8440 -
val_loss: 0.4868 - val_recall: 0.7224
Epoch 36/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3546 - recall: 0.8461 -
val_loss: 0.4533 - val_recall: 0.6732
Epoch 37/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3531 - recall: 0.8476 -
val_loss: 0.4305 - val_recall: 0.6413
Epoch 38/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3523 - recall: 0.8503 -
val_loss: 0.4433 - val_recall: 0.6241
Epoch 39/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3521 - recall: 0.8482 -
val_loss: 0.4777 - val_recall: 0.6978
Epoch 40/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3510 - recall: 0.8537 -
val_loss: 0.4621 - val_recall: 0.6781
Epoch 41/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3498 - recall: 0.8526 -
val_loss: 0.4493 - val_recall: 0.6511
Epoch 42/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3485 - recall: 0.8528 -
val_loss: 0.4327 - val_recall: 0.6339
Epoch 43/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3462 - recall: 0.8489 -
val_loss: 0.4955 - val_recall: 0.7027
Epoch 44/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3449 - recall: 0.8547 -
val_loss: 0.5031 - val_recall: 0.7371
Epoch 45/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3466 - recall: 0.8535 -
val_loss: 0.4332 - val_recall: 0.6339
Epoch 46/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3459 - recall: 0.8537 -
val_loss: 0.4731 - val_recall: 0.6806
Epoch 47/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3427 - recall: 0.8526 -
val_loss: 0.5197 - val_recall: 0.7297
Epoch 48/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3431 - recall: 0.8530 -
val_loss: 0.4358 - val_recall: 0.6118
Epoch 49/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3400 - recall: 0.8543 -
val_loss: 0.5149 - val_recall: 0.7150
Epoch 50/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3394 - recall: 0.8585 -
val_loss: 0.4653 - val_recall: 0.6658
Epoch 51/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3378 - recall: 0.8589 -
val_loss: 0.5083 - val_recall: 0.7101
Epoch 52/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3410 - recall: 0.8535 -
```

```
299/299 [==============================] - 1s 3ms/step - loss: 0.3410 - recall: 0.8555 -
val_loss: 0.4667 - val_recall: 0.6511
Epoch 53/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3381 - recall: 0.8602 -
val_loss: 0.4568 - val_recall: 0.6339
Epoch 54/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3367 - recall: 0.8551 -
val_loss: 0.5154 - val_recall: 0.6953
Epoch 55/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3356 - recall: 0.8610 -
val_loss: 0.4307 - val_recall: 0.5774
Epoch 56/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3372 - recall: 0.8574 -
val_loss: 0.4414 - val_recall: 0.5946
Epoch 57/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3320 - recall: 0.8620 -
val_loss: 0.4625 - val_recall: 0.6192
Epoch 58/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3327 - recall: 0.8581 -
val_loss: 0.4792 - val_recall: 0.6560
Epoch 59/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3340 - recall: 0.8591 -
val_loss: 0.4724 - val_recall: 0.6437
Epoch 60/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3313 - recall: 0.8602 -
val_loss: 0.4590 - val_recall: 0.6241
Epoch 61/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3307 - recall: 0.8616 -
val_loss: 0.4664 - val_recall: 0.6413
Epoch 62/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3297 - recall: 0.8614 -
val_loss: 0.4585 - val_recall: 0.6339
Epoch 63/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3302 - recall: 0.8597 -
val_loss: 0.4489 - val_recall: 0.5799
Epoch 64/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3294 - recall: 0.8618 -
val_loss: 0.5048 - val_recall: 0.6929
Epoch 65/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3273 - recall: 0.8604 -
val_loss: 0.4829 - val_recall: 0.6609
Epoch 66/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3248 - recall: 0.8627 -
val_loss: 0.4657 - val_recall: 0.6241
Epoch 67/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3259 - recall: 0.8648 -
val_loss: 0.4880 - val_recall: 0.6437
Epoch 68/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3256 - recall: 0.8620 -
val_loss: 0.4646 - val_recall: 0.6241
Epoch 69/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3242 - recall: 0.8664 -
val_loss: 0.4458 - val_recall: 0.5921
Epoch 70/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3239 - recall: 0.8623 -
val_loss: 0.5144 - val_recall: 0.6953
Epoch 71/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3214 - recall: 0.8675 -
val_loss: 0.4542 - val_recall: 0.6044
Epoch 72/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3227 - recall: 0.8608 -
val_loss: 0.4507 - val_recall: 0.5749
Epoch 73/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3241 - recall: 0.8633 -
val_loss: 0.4931 - val_recall: 0.6511
Epoch 74/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3208 - recall: 0.8675 -
val_loss: 0.4424 - val_recall: 0.5749
Epoch 75/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3200 - recall: 0.8620 -
val_loss: 0.5188 - val_recall: 0.7027
Epoch 76/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3208 - recall: 0.8669 -
```

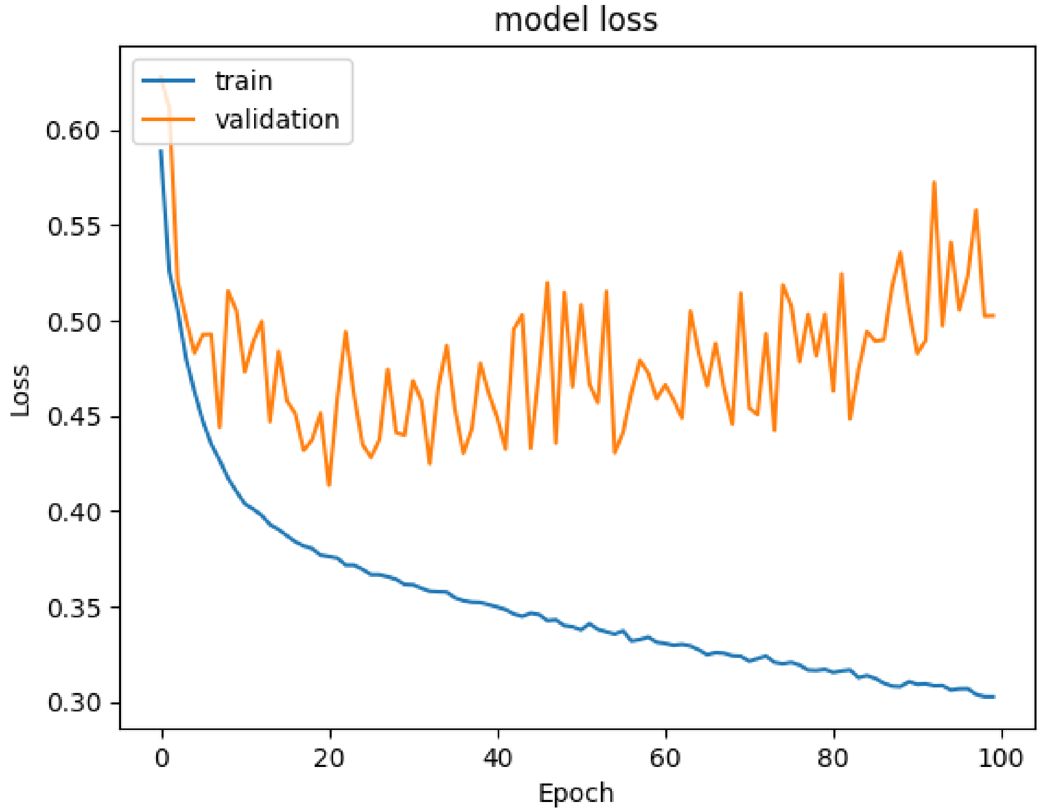
```
val_loss: 0.5077 - val_recall: 0.6781
Epoch 77/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3193 - recall: 0.8669 -
val_loss: 0.4784 - val_recall: 0.6167
Epoch 78/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3168 - recall: 0.8696 -
val_loss: 0.5033 - val_recall: 0.6609
Epoch 79/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3166 - recall: 0.8667 -
val_loss: 0.4814 - val_recall: 0.6314
Epoch 80/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3172 - recall: 0.8654 -
val_loss: 0.5034 - val_recall: 0.6486
Epoch 81/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3156 - recall: 0.8719 -
val_loss: 0.4630 - val_recall: 0.5872
Epoch 82/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3163 - recall: 0.8669 -
val_loss: 0.5246 - val_recall: 0.6880
Epoch 83/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3168 - recall: 0.8681 -
val_loss: 0.4485 - val_recall: 0.5577
Epoch 84/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3128 - recall: 0.8648 -
val_loss: 0.4742 - val_recall: 0.5946
Epoch 85/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3139 - recall: 0.8664 -
val_loss: 0.4943 - val_recall: 0.6511
Epoch 86/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3123 - recall: 0.8679 -
val_loss: 0.4892 - val_recall: 0.6314
Epoch 87/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3099 - recall: 0.8690 -
val_loss: 0.4899 - val_recall: 0.6241
Epoch 88/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3083 - recall: 0.8687 -
val_loss: 0.5176 - val_recall: 0.6486
Epoch 89/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3080 - recall: 0.8748 -
val_loss: 0.5357 - val_recall: 0.6855
Epoch 90/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3107 - recall: 0.8713 -
val_loss: 0.5055 - val_recall: 0.6339
Epoch 91/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3093 - recall: 0.8721 -
val_loss: 0.4828 - val_recall: 0.6118
Epoch 92/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3095 - recall: 0.8694 -
val_loss: 0.4896 - val_recall: 0.6044
Epoch 93/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3084 - recall: 0.8698 -
val_loss: 0.5724 - val_recall: 0.7002
Epoch 94/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3086 - recall: 0.8692 -
val_loss: 0.4973 - val_recall: 0.6093
Epoch 95/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3063 - recall: 0.8721 -
val_loss: 0.5410 - val_recall: 0.6708
Epoch 96/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3069 - recall: 0.8719 -
val_loss: 0.5056 - val_recall: 0.6364
Epoch 97/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3069 - recall: 0.8656 -
val_loss: 0.5237 - val_recall: 0.6757
Epoch 98/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3040 - recall: 0.8748 -
val_loss: 0.5580 - val_recall: 0.6830
Epoch 99/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3028 - recall: 0.8754 -
val_loss: 0.5024 - val_recall: 0.6290
Epoch 100/100
299/299 [==============================] - 1s 2ms/step - loss: 0.3028 - recall: 0.8746 -
```

```
val_loss: 0.5025 - val_recall: 0.6143
```

Loss function

In [109]:

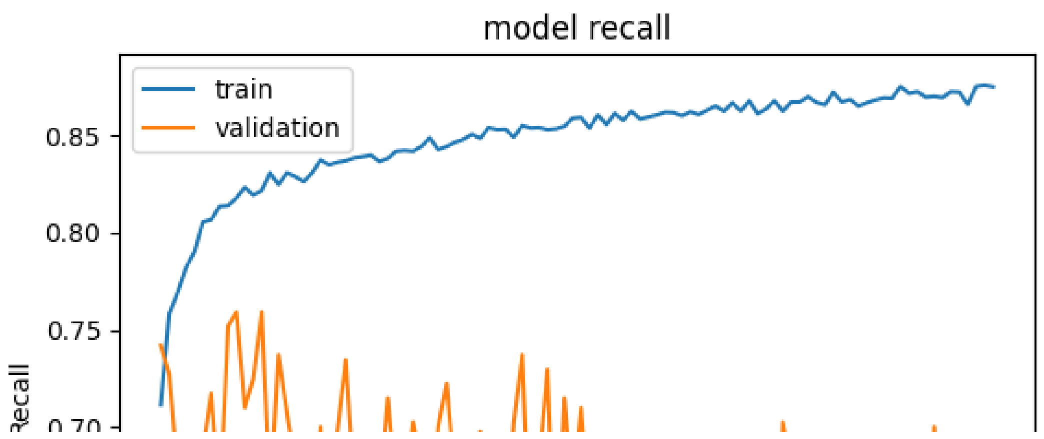
```
#Plotting Train Loss vs Validation Loss
plt.plot(history_4.history['loss'])
plt.plot(history_4.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

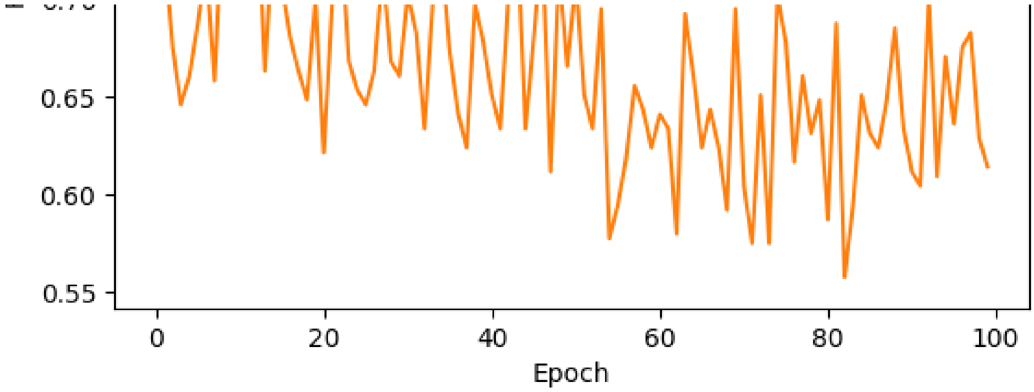


Recall

In [110]:

```
#Plotting Train recall vs Validation recall
plt.plot(history_4.history['recall'])
plt.plot(history_4.history['val_recall'])
plt.title('model recall')
plt.ylabel('Recall')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





In [111]:

```
#Predicting the results using 0.5 as the threshold
y_train_pred = model_4.predict(X_train_smote)
y_train_pred = (y_train_pred > 0.5)
y_train_pred
```

```
299/299 [==============================] - 1s 2ms/step
```

Out[111]:

```
array([[False],
       [ True],
       [ True],
       ...,
       [ True],
       [False],
       [ True]])
```

In [112]:

```
#Predicting the results using 0.5 as the threshold
y_val_pred = model_4.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
y_val_pred
```

```
63/63 [==============================] - 0s 2ms/step
```

Out[112]:

```
array([[False],
       [False],
       [ True],
       ...,
       [False],
       [False],
       [False]])
```

In [113]:

```
model_name = "NN with Adam Smote "

train_metric_df.loc[model_name] = recall_score(y_train_smote,y_train_pred)
valid_metric_df.loc[model_name] = recall_score(y_val,y_val_pred)

print(train_metric_df)
```

```
                        recall
NN with SGD           0.219133
NN with Adam          0.678659
NN with Adam DropOut  0.538021
NN with SGD Smote     0.757379
NN with Adam Smote    0.877329
```

Classification report

In [114]:

```
#lassification report
cr=classification_report(y_train_smote,y_train_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.88      0.87      0.87      4777
           1       0.87      0.88      0.87      4777

    accuracy                           0.87      9554
   macro avg       0.87      0.87      0.87      9554
weighted avg       0.87      0.87      0.87      9554
```

In [115]:

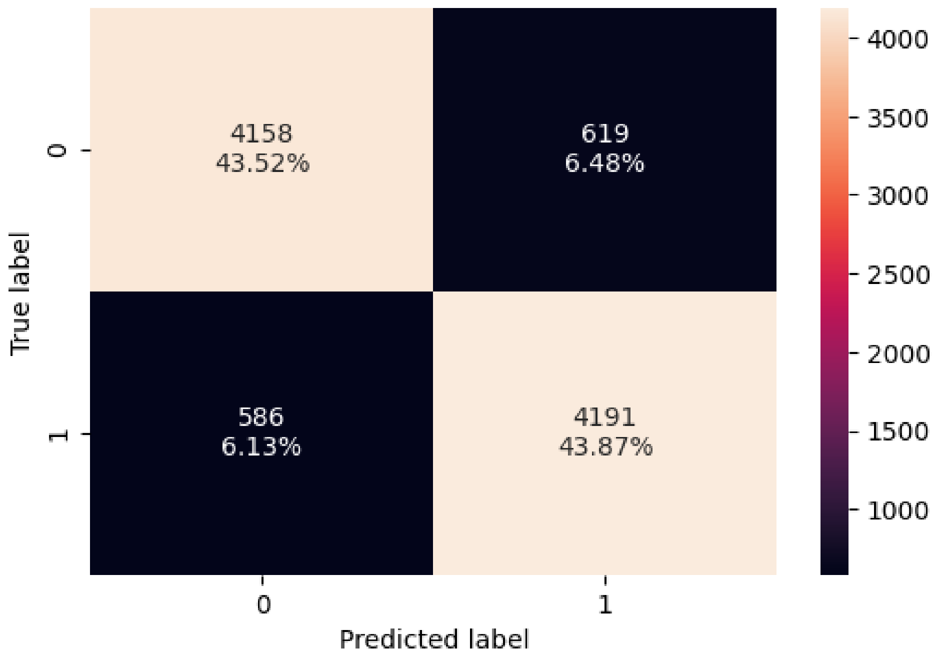
```
#classification report
cr=classification_report(y_val,y_val_pred)  ## Complete the code to check the model's per
formance on the validation set
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.89      0.84      0.86      1593
           1       0.49      0.61      0.54       407

    accuracy                           0.79      2000
   macro avg       0.69      0.73      0.70      2000
weighted avg       0.81      0.79      0.80      2000
```

Confusion matrix

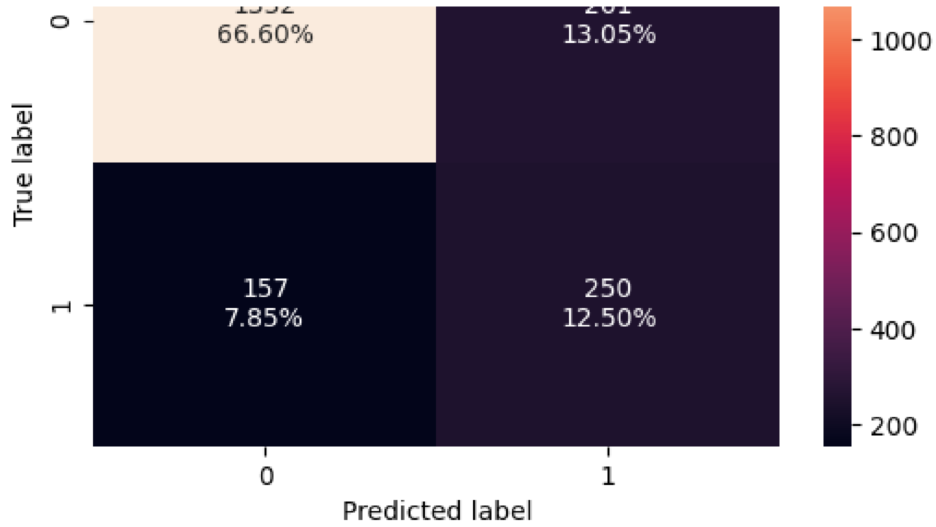
In [116]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_train_smote, y_train_pred)
```



In [117]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)  ## Complete the code to check the model's perfor
mance on the validation set
```



Neural Network with Balanced Data (by applying SMOTE), Adam Optimizer, and Dropout

In [118]:

```
#sm  = SMOTE(random_state=42)
#Fit SMOTE on the training data.
#X_train_smote, y_train_smote= sm.fit_resample(X_train,y_train)
print('After UpSampling, the shape of train_X: {}'.format(X_train_smote.shape))
print('After UpSampling, the shape of train_y: {} \n'.format(y_train_smote.shape))
```

```
After UpSampling, the shape of train_X: (9554, 11)
After UpSampling, the shape of train_y: (9554,)
```

In [119]:

```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the same o
utput everytime
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [120]:

```
#Initializing the neural network
model_5 = Sequential()
# Adding the input layer with 32 neurons and relu as activation function
model_5.add(Dense(32,activation='relu',input_dim = X_train_smote.shape[1]))
#Complete the code to add dropout rate
model_5.add(Dropout(0.2))
# Add a hidden layer (specify the # of neurons and the activation function)
model_5.add(Dense(16,activation='relu'))
#Complete the code to add dropout rate
model_5.add(Dropout(0.1))
# Add a hidden layer (specify the # of neurons and the activation function)
model_5.add(Dense(8,activation='relu'))
# Add a hidden layer (specify the # of neurons and the activation function)
model_5.add(Dense(1, activation = 'sigmoid'))
```

In [121]:

```
#Complete the code to use Adam as the optimizer.
optimizer = tf.keras.optimizers.Adam()

# uncomment one of the following lines to define the metric to be used
# metric = 'accuracy'
metric = keras.metrics.Recall()
# metric = keras.metrics.Precision()
# metric = keras.metrics.F1Score()
```

In [122]:

```python
# Complete the code to compile the model with binary cross entropy as loss function and r
ecall as the metric
model_5.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [123]:

```python
model_5.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                384

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 16)                528

 dropout_1 (Dropout)         (None, 16)                0

 dense_2 (Dense)             (None, 8)                 136

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 1057 (4.13 KB)
Trainable params: 1057 (4.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [124]:

```python
#Fitting the ANN
history_5 = model_5.fit(
    X_train_smote,y_train_smote,
    batch_size=32,
    validation_data=(X_val,y_val),
    epochs=100,
    verbose=1
)
```

```
Epoch 1/100
299/299 [==============================] - 5s 11ms/step - loss: 0.6157 - recall: 0.6703 -
val_loss: 0.5860 - val_recall: 0.7002
Epoch 2/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5574 - recall: 0.7331 -
val_loss: 0.5786 - val_recall: 0.6855
Epoch 3/100
299/299 [==============================] - 1s 4ms/step - loss: 0.5430 - recall: 0.7463 -
val_loss: 0.5455 - val_recall: 0.6683
Epoch 4/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5299 - recall: 0.7524 -
val_loss: 0.5280 - val_recall: 0.6658
Epoch 5/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5176 - recall: 0.7616 -
val_loss: 0.5335 - val_recall: 0.6855
Epoch 6/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5074 - recall: 0.7722 -
val_loss: 0.5295 - val_recall: 0.6806
Epoch 7/100
299/299 [==============================] - 1s 3ms/step - loss: 0.5004 - recall: 0.7674 -
val_loss: 0.5080 - val_recall: 0.6462
Epoch 8/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4900 - recall: 0.7752 -
val_loss: 0.4914 - val_recall: 0.6339
Epoch 9/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4805 - recall: 0.7756 -
val_loss: 0.4923 - val_recall: 0.6757
Epoch 10/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4799 - recall: 0.7764
```

```
299/299 [==============================] - 1s 3ms/step - loss: 0.4799 - recall: 0.7764 -
val_loss: 0.5056 - val_recall: 0.7150
Epoch 11/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4674 - recall: 0.7842 -
val_loss: 0.4862 - val_recall: 0.7002
Epoch 12/100
299/299 [==============================] - 1s 2ms/step - loss: 0.4620 - recall: 0.7865 -
val_loss: 0.5004 - val_recall: 0.7101
Epoch 13/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4598 - recall: 0.7835 -
val_loss: 0.4857 - val_recall: 0.7076
Epoch 14/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4570 - recall: 0.7963 -
val_loss: 0.4669 - val_recall: 0.6806
Epoch 15/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4443 - recall: 0.7988 -
val_loss: 0.4834 - val_recall: 0.7297
Epoch 16/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4418 - recall: 0.8020 -
val_loss: 0.4568 - val_recall: 0.6855
Epoch 17/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4339 - recall: 0.8036 -
val_loss: 0.4783 - val_recall: 0.7101
Epoch 18/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4332 - recall: 0.7972 -
val_loss: 0.4476 - val_recall: 0.6830
Epoch 19/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4308 - recall: 0.8024 -
val_loss: 0.4564 - val_recall: 0.6978
Epoch 20/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4276 - recall: 0.8043 -
val_loss: 0.4555 - val_recall: 0.7002
Epoch 21/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4291 - recall: 0.8036 -
val_loss: 0.4544 - val_recall: 0.7101
Epoch 22/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4247 - recall: 0.8137 -
val_loss: 0.4599 - val_recall: 0.7199
Epoch 23/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4203 - recall: 0.8147 -
val_loss: 0.4779 - val_recall: 0.7396
Epoch 24/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4229 - recall: 0.8126 -
val_loss: 0.4507 - val_recall: 0.7052
Epoch 25/100
299/299 [==============================] - 1s 2ms/step - loss: 0.4180 - recall: 0.8141 -
val_loss: 0.4423 - val_recall: 0.6830
Epoch 26/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4218 - recall: 0.8091 -
val_loss: 0.4338 - val_recall: 0.6830
Epoch 27/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4195 - recall: 0.8049 -
val_loss: 0.4521 - val_recall: 0.7052
Epoch 28/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4194 - recall: 0.8166 -
val_loss: 0.4464 - val_recall: 0.7027
Epoch 29/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4142 - recall: 0.8097 -
val_loss: 0.4503 - val_recall: 0.7002
Epoch 30/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4175 - recall: 0.8068 -
val_loss: 0.4639 - val_recall: 0.7174
Epoch 31/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4140 - recall: 0.8103 -
val_loss: 0.4556 - val_recall: 0.7150
Epoch 32/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4117 - recall: 0.8137 -
val_loss: 0.4447 - val_recall: 0.6978
Epoch 33/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4114 - recall: 0.8198 -
val_loss: 0.4590 - val_recall: 0.7248
Epoch 34/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4142 - recall: 0.8193
```

```
299/299 [==============================] - 1s 4ms/step - loss: 0.4142 - recall: 0.8193 -
val_loss: 0.4647 - val_recall: 0.7297
Epoch 35/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4122 - recall: 0.8177 -
val_loss: 0.4499 - val_recall: 0.7125
Epoch 36/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4096 - recall: 0.8168 -
val_loss: 0.4464 - val_recall: 0.7101
Epoch 37/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4115 - recall: 0.8177 -
val_loss: 0.4359 - val_recall: 0.6855
Epoch 38/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4114 - recall: 0.8085 -
val_loss: 0.4351 - val_recall: 0.6880
Epoch 39/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4076 - recall: 0.8126 -
val_loss: 0.4527 - val_recall: 0.7101
Epoch 40/100
299/299 [==============================] - 1s 2ms/step - loss: 0.4092 - recall: 0.8152 -
val_loss: 0.4400 - val_recall: 0.7076
Epoch 41/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4017 - recall: 0.8149 -
val_loss: 0.4698 - val_recall: 0.7199
Epoch 42/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4056 - recall: 0.8164 -
val_loss: 0.4568 - val_recall: 0.7174
Epoch 43/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4049 - recall: 0.8057 -
val_loss: 0.4446 - val_recall: 0.6929
Epoch 44/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4030 - recall: 0.8139 -
val_loss: 0.4364 - val_recall: 0.6929
Epoch 45/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4004 - recall: 0.8187 -
val_loss: 0.4521 - val_recall: 0.7052
Epoch 46/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4024 - recall: 0.8200 -
val_loss: 0.4601 - val_recall: 0.7224
Epoch 47/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4066 - recall: 0.8126 -
val_loss: 0.4528 - val_recall: 0.7150
Epoch 48/100
299/299 [==============================] - 1s 4ms/step - loss: 0.4046 - recall: 0.8141 -
val_loss: 0.4483 - val_recall: 0.7101
Epoch 49/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3997 - recall: 0.8193 -
val_loss: 0.4493 - val_recall: 0.7174
Epoch 50/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4034 - recall: 0.8177 -
val_loss: 0.4527 - val_recall: 0.7174
Epoch 51/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4032 - recall: 0.8162 -
val_loss: 0.4648 - val_recall: 0.7273
Epoch 52/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4001 - recall: 0.8164 -
val_loss: 0.4452 - val_recall: 0.6904
Epoch 53/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4028 - recall: 0.8143 -
val_loss: 0.4398 - val_recall: 0.6953
Epoch 54/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3987 - recall: 0.8152 -
val_loss: 0.4468 - val_recall: 0.7150
Epoch 55/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4013 - recall: 0.8196 -
val_loss: 0.4491 - val_recall: 0.7052
Epoch 56/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4053 - recall: 0.8145 -
val_loss: 0.4492 - val_recall: 0.7076
Epoch 57/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3922 - recall: 0.8189 -
val_loss: 0.4415 - val_recall: 0.6904
Epoch 58/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4021 - recall: 0.8145
```

```
299/299 [==============================] - 1s 3ms/step - loss: 0.4021 - recall: 0.8145 -
val_loss: 0.4473 - val_recall: 0.7027
Epoch 59/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3956 - recall: 0.8198 -
val_loss: 0.4342 - val_recall: 0.6781
Epoch 60/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4005 - recall: 0.8212 -
val_loss: 0.4389 - val_recall: 0.7101
Epoch 61/100
299/299 [==============================] - 1s 3ms/step - loss: 0.4015 - recall: 0.8177 -
val_loss: 0.4566 - val_recall: 0.7420
Epoch 62/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3974 - recall: 0.8206 -
val_loss: 0.4513 - val_recall: 0.7199
Epoch 63/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3979 - recall: 0.8181 -
val_loss: 0.4548 - val_recall: 0.7248
Epoch 64/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3970 - recall: 0.8196 -
val_loss: 0.4448 - val_recall: 0.7199
Epoch 65/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3974 - recall: 0.8204 -
val_loss: 0.4296 - val_recall: 0.6855
Epoch 66/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3905 - recall: 0.8175 -
val_loss: 0.4688 - val_recall: 0.7420
Epoch 67/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3925 - recall: 0.8233 -
val_loss: 0.4416 - val_recall: 0.7027
Epoch 68/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3975 - recall: 0.8210 -
val_loss: 0.4377 - val_recall: 0.7150
Epoch 69/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3983 - recall: 0.8204 -
val_loss: 0.4454 - val_recall: 0.7125
Epoch 70/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3961 - recall: 0.8198 -
val_loss: 0.4587 - val_recall: 0.7273
Epoch 71/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3933 - recall: 0.8288 -
val_loss: 0.4391 - val_recall: 0.7076
Epoch 72/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3985 - recall: 0.8242 -
val_loss: 0.4252 - val_recall: 0.6904
Epoch 73/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3967 - recall: 0.8181 -
val_loss: 0.4578 - val_recall: 0.7248
Epoch 74/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3904 - recall: 0.8237 -
val_loss: 0.4454 - val_recall: 0.6978
Epoch 75/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3957 - recall: 0.8206 -
val_loss: 0.4486 - val_recall: 0.7101
Epoch 76/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3922 - recall: 0.8202 -
val_loss: 0.4479 - val_recall: 0.7248
Epoch 77/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3887 - recall: 0.8357 -
val_loss: 0.4296 - val_recall: 0.7052
Epoch 78/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3910 - recall: 0.8288 -
val_loss: 0.4434 - val_recall: 0.7076
Epoch 79/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3936 - recall: 0.8258 -
val_loss: 0.4377 - val_recall: 0.7125
Epoch 80/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3887 - recall: 0.8229 -
val_loss: 0.4406 - val_recall: 0.6929
Epoch 81/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3933 - recall: 0.8256 -
val_loss: 0.4495 - val_recall: 0.7076
Epoch 82/100
```

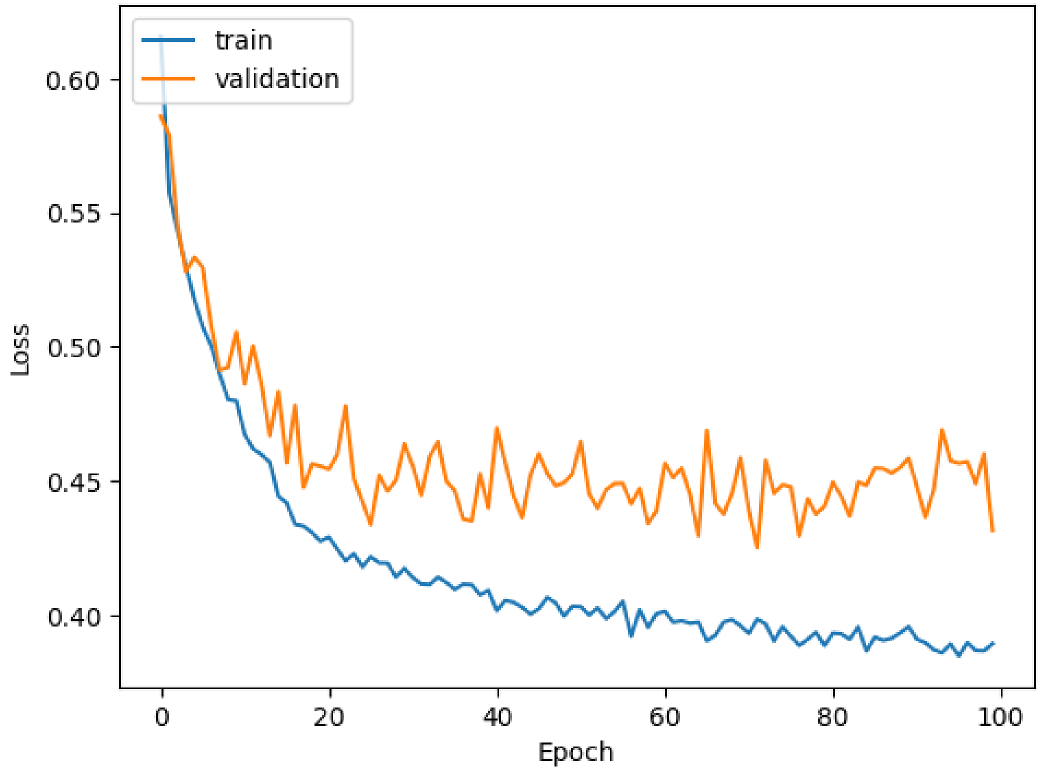
```
299/299 [==============================] - 1s 3ms/step - loss: 0.3931 - recall: 0.8137 -
val_loss: 0.4444 - val_recall: 0.7224
Epoch 83/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3911 - recall: 0.8273 -
val_loss: 0.4370 - val_recall: 0.7052
Epoch 84/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3956 - recall: 0.8216 -
val_loss: 0.4496 - val_recall: 0.7248
Epoch 85/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3867 - recall: 0.8246 -
val_loss: 0.4484 - val_recall: 0.7322
Epoch 86/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3918 - recall: 0.8254 -
val_loss: 0.4548 - val_recall: 0.7297
Epoch 87/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3906 - recall: 0.8252 -
val_loss: 0.4546 - val_recall: 0.7322
Epoch 88/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3914 - recall: 0.8237 -
val_loss: 0.4529 - val_recall: 0.7297
Epoch 89/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3934 - recall: 0.8265 -
val_loss: 0.4550 - val_recall: 0.7224
Epoch 90/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3958 - recall: 0.8202 -
val_loss: 0.4586 - val_recall: 0.7346
Epoch 91/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3911 - recall: 0.8317 -
val_loss: 0.4480 - val_recall: 0.7101
Epoch 92/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3898 - recall: 0.8265 -
val_loss: 0.4365 - val_recall: 0.6953
Epoch 93/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3871 - recall: 0.8227 -
val_loss: 0.4469 - val_recall: 0.7174
Epoch 94/100
299/299 [==============================] - 1s 4ms/step - loss: 0.3860 - recall: 0.8292 -
val_loss: 0.4691 - val_recall: 0.7543
Epoch 95/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3892 - recall: 0.8208 -
val_loss: 0.4576 - val_recall: 0.7346
Epoch 96/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3849 - recall: 0.8279 -
val_loss: 0.4566 - val_recall: 0.7248
Epoch 97/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3898 - recall: 0.8292 -
val_loss: 0.4572 - val_recall: 0.7273
Epoch 98/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3869 - recall: 0.8273 -
val_loss: 0.4490 - val_recall: 0.7248
Epoch 99/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3867 - recall: 0.8315 -
val_loss: 0.4601 - val_recall: 0.7371
Epoch 100/100
299/299 [==============================] - 1s 3ms/step - loss: 0.3894 - recall: 0.8246 -
val_loss: 0.4316 - val_recall: 0.6978
```

Loss function

In [125]:

```
#Plotting Train Loss vs Validation Loss
plt.plot(history_5.history['loss'])
plt.plot(history_5.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

model loss



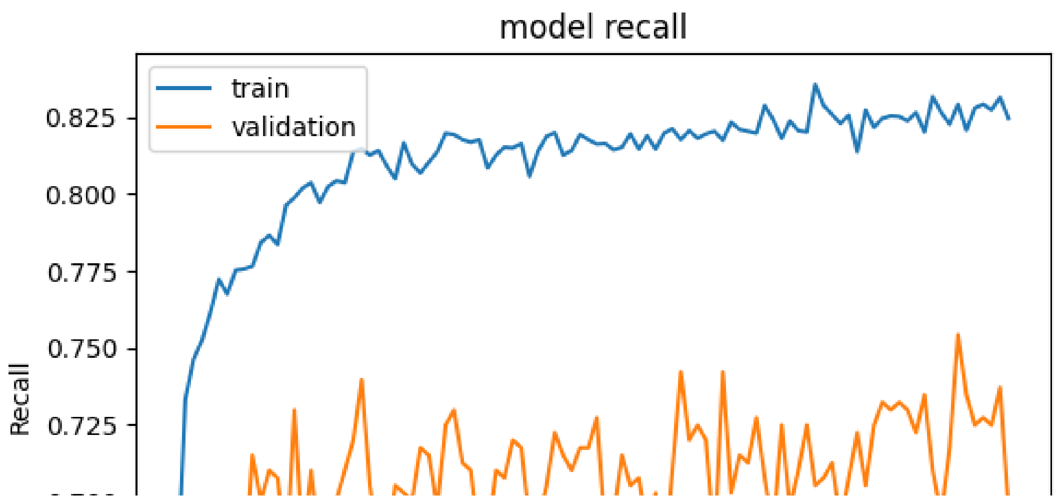
Model Loss Over Epochs

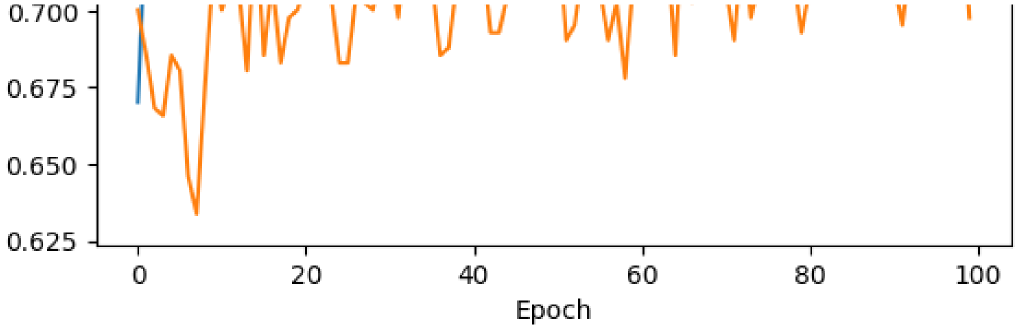
- **Initially, both training and validation loss decrease sharply, indicating that the model is learning and improving its predictions.**
- **The training loss continues to decrease and begins to plateau, showing signs of model convergence.**
- **In contrast, the validation loss decreases initially but then starts to exhibit fluctuations, suggesting variability in the model's performance on the validation data.**
- **Around epoch 20, the validation loss begins to diverge from the training loss, possibly indicating the beginning of overfitting.**
- **The model seems to generalize well initially but may benefit from regularization or early stopping to address the divergence in later epochs.**

Recall

In [126]:

```
#Plotting Train recall vs Validation recall
plt.plot(history_5.history['recall'])
plt.plot(history_5.history['val_recall'])
plt.title('model recall')
plt.ylabel('Recall')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





Model Recall Over Epochs

- **The recall on the training set shows a steady increase early on and levels off, maintaining a high recall above 80% after approximately 20 epochs.**
- **In contrast, the validation recall is more volatile, with significant fluctuations throughout the training process.**
- **Although the validation recall improves initially, it does not achieve the same level of stability or performance as the training recall, suggesting potential overfitting or that the model may not generalize as well to unseen data.**
- **The gap between training and validation recall suggests the model could be improved, potentially by fine-tuning, adding regularization, or implementing other techniques to enhance generalization to the validation set.**

In [127]:

```
#Predicting the results using 0.5 as the threshold
y_train_pred = model_5.predict(X_train_smote)
y_train_pred = (y_train_pred > 0.5)
y_train_pred
```

```
299/299 [==============================] - 0s 1ms/step
```

Out[127]:

```
array([[False],
       [ True],
       [ True],
       ...,
       [ True],
       [False],
       [ True]])
```

In [128]:

```
#Predicting the results using 0.5 as the threshold
y_val_pred = model_5.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
y_val_pred
```

```
63/63 [==============================] - 0s 1ms/step
```

Out[128]:

```
array([[False],
       [False],
       [ True],
       ...,
       [False],
       [False],
       [ True]])
```

In [129]:

```
model_name = "NN with Adam Smote Dropout"

train_metric_df.loc[model_name] = recall_score(y_train_smote,y_train_pred)
valid_metric_df.loc[model_name] = recall_score(y_val,y_val_pred)
```

```
print(train_metric_df)
```

```
                              recall
NN with SGD                 0.219133
NN with Adam                0.678659
NN with Adam DropOut        0.538021
NN with SGD Smote           0.757379
NN with Adam Smote          0.877329
NN with Adam Smote Dropout  0.839230
```

Classification report

In [130]:

```
#lassification report
cr=classification_report(y_train_smote,y_train_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.84      0.85      0.85      4777
           1       0.85      0.84      0.85      4777

    accuracy                           0.85      9554
   macro avg       0.85      0.85      0.85      9554
weighted avg       0.85      0.85      0.85      9554
```

In [131]:

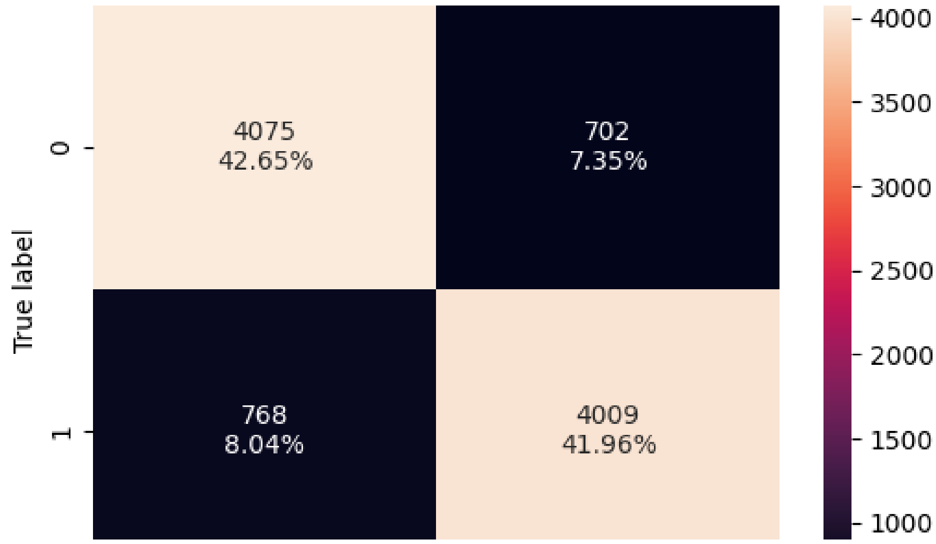
```
#classification report
cr=classification_report(y_val,y_val_pred)  ## Complete the code to check the model's per
formance on the validation set
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.92      0.83      0.87      1593
           1       0.52      0.70      0.59       407

    accuracy                           0.81      2000
   macro avg       0.72      0.77      0.73      2000
weighted avg       0.83      0.81      0.82      2000
```

Confusion matrix

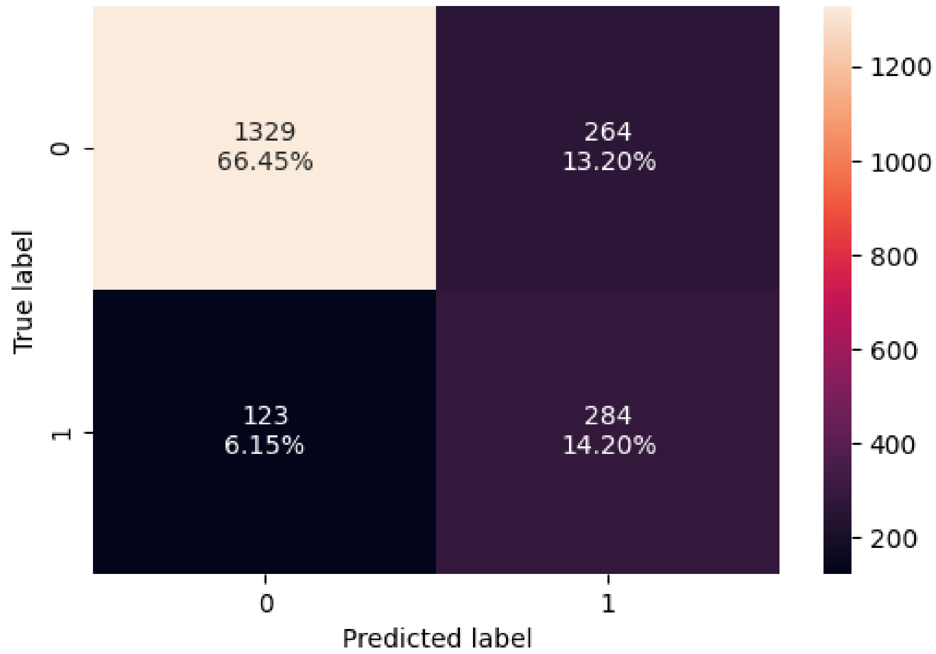
In [132]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_train_smote, y_train_pred)
```



In [133]:

```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)  ## Complete the code to check the model's perfor
mance on the validation set
```



Model Performance Comparison and Final Model Selection

In [134]:

```
print("Training performance comparison")
train_metric_df
```

```
Training performance comparison
```

Out[134]:

	recall
NN with SGD	0.219133
NN with Adam	0.678659
NN with Adam DropOut	0.538021
NN with SGD Smote	0.757379
NN with Adam Smote	0.877329
NN with Adam Smote Dropout	0.839230

NN with Adam Smote is our best model on Training set

In [135]:

```
print("Validation set performance comparison")
valid_metric_df
```

```
Validation set performance comparison
```

Out[135]:

	recall

	recall
NN with SGD	0.194103
NN with Adam	0.493857
NN with Adam DropOut	0.437346
NN with SGD Smote	0.660934
NN with Adam Smote	0.614251
NN with Adam Smote Dropout	0.697789

NN with Adam Smote Dropout is our best model on validation set

In [136]:

```
train_metric_df - valid_metric_df
```

Out[136]:

	recall
NN with SGD	0.025030
NN with Adam	0.184802
NN with Adam DropOut	0.100675
NN with SGD Smote	0.096445
NN with Adam Smote	0.263078
NN with Adam Smote Dropout	0.141441

Model 5 which is NN with Adam Smote Dropout has best performance in validation set and closer to train set result.

In [137]:

```
# Test set using best model - Model 5
y_test_pred = model_5.predict(X_test)
y_test_pred = (y_test_pred > 0.5)
print(y_test_pred)
```

```
63/63 [==============================] - 0s 1ms/step
[[False]
 [False]
 [False]
 ...
 [ True]
 [False]
 [False]]
```

In [138]:

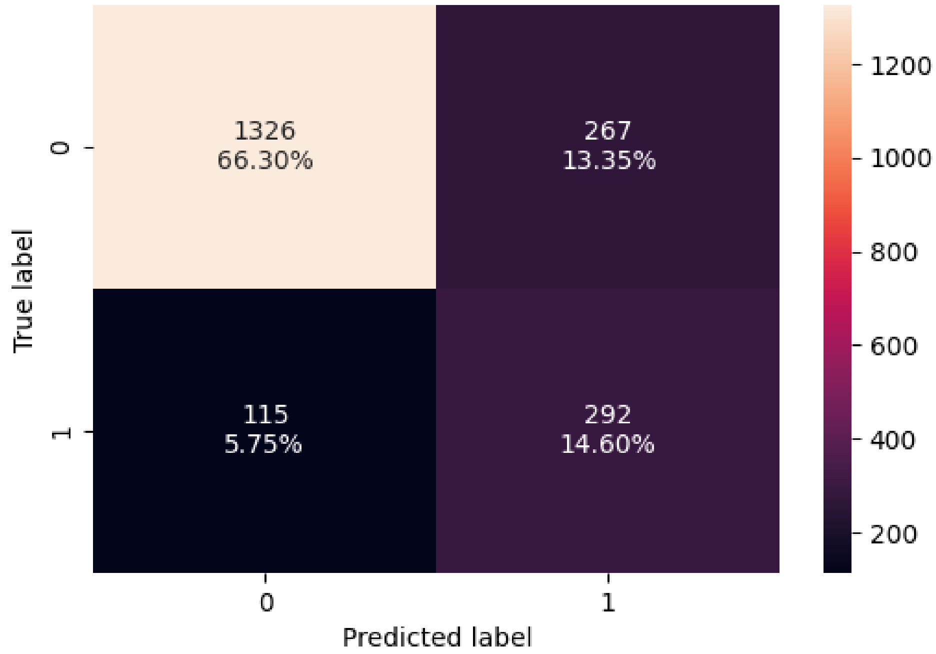
```
#lets print classification report
cr=classification_report(y_test,y_test_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.92      0.83      0.87      1593
           1       0.52      0.72      0.60       407

    accuracy                           0.81      2000
   macro avg       0.72      0.77      0.74      2000
weighted avg       0.84      0.81      0.82      2000
```

In [139]:

```
#Calculating the confusion matrix
```

```
make_confusion_matrix(y_test,y_test_pred)
```



Actionable Insights and Business Recommendations

Final Model Selection and Business Recommendations

Based on a comprehensive evaluation of six neural network configurations, the chosen model incorporates Adam optimization, SMOTE for handling class imbalance, and Dropout for regularization . The model demonstrates a compelling balance between recall and precision, achieving an overall accuracy of 81% on the validation set.

Model Performance Highlights

- **The final model significantly outperforms others, particularly in recall, capturing 72% of actual churn cases.**
- **While the precision is lower for predicting churn (52%), this is offset by the model's ability to correctly identify a higher number of customers who might churn.**

Confusion Matrix Analysis

- **True Negatives (TN): 1326 customers were correctly predicted to not churn, accounting for 66.30%.**
- **False Positives (FP): 267 customers were incorrectly predicted to churn, 13.35% of predictions.**
- **True Positives (TP): 292 customers who churned were correctly identified, representing 14.60% of predictions.**
- **False Negatives (FN): 115 customers were incorrectly predicted to not churn, 5.75%.**

Actionable Insights

- **Customer Retention Focus: With a high recall, the bank can deploy targeted retention strategies to the identified at-risk customers, potentially preventing actual churn.**
- **Precision Trade-Off: The lower precision is an acceptable trade-off for higher recall in the context of churn prediction. It is more cost-effective to engage false positives than miss true positives in retention efforts.**

Business Recommendations

- **Retention Campaigns: Prioritize retention campaigns for customers identified as high risk by the model, especially older customers and those with higher balances.**
- **Product Engagement: Investigate why customers with more products are less likely to churn and consider cross-selling or upselling strategies.**
- **Loyalty Programs: Enhance loyalty programs aimed at increasing active membership, as it is strongly correlated with reduced churn.**

correlated with reduced churn.

- **Customer Service Enhancement**: Improve customer service touchpoints for customers in the higher age brackets to address their specific needs and concerns, as age has shown to be a strong churn predictor.
- **Further Analysis**: Conduct further analysis on customers with low balances and those holding multiple products, as the model indicates these factors contribute to lower churn.
- **Feedback Loop**: Establish a feedback loop where model predictions are constantly evaluated against actual outcomes to continuously refine the prediction model and retention strategies.

This model, with its focus on recall, empowers the bank to proactively manage and mitigate churn, thereby enhancing customer value and loyalty.

Power Ahead *AY*
