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)**

In [1]:

```
!pip install tensorflow==2.15.0 scikit-learn==1.2.2 matplotlib===3.7.1 seaborn==0.13.1 n
umpy==1.25.2 pandas==1.5.3 -q --user
```

Importing necessary libraries

In [2]:

```
# Library for data manipulation and analysis.
import pandas as pd
# Fundamental package for scientific computing.
import numpy as np
#splitting datasets into training and testing sets.
from sklearn.model_selection import train_test_split
#Imports tools for data preprocessing including label encoding, one-hot encoding, and sta
ndard scaling
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,StandardScaler
#Imports a class for imputing missing values in datasets.
from sklearn.impute import SimpleImputer
```

```
#Imports the Matplotlib library for creating visualizations.
import matplotlib.pyplot as plt
# Imports the Seaborn library for statistical data visualization.
import seaborn as sns
# Time related functions.
import time
#Imports functions for evaluating the performance of machine learning models
from sklearn.metrics import confusion_matrix, f1_score,accuracy_score, recall_score, prec
ision_score, classification_report

# to suppress unnecessary warnings
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
import random

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

# importing SMOTE
from imblearn.over_sampling import SMOTE
```

In [4]:

```
from tensorflow import keras
from keras import backend
from keras.models import Sequential
from keras.layers import Dense, Dropout
#Imports the tensorflow,keras and layers.
import tensorflow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dense, Input, Dropout,BatchNormalization
from tensorflow.keras import backend
```

Loading the dataset

Working with a copy of the data helps maintain data integrity, enables error-free analysis, supports reproducibility, and allows for safe experimentation and version control.

In [5]:

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
("/content/drive", force_remount=True).
```

In [6]:

```
data = pd.read_csv("/content/drive/MyDrive/AIML_Neural Network/Churn.csv")
```

In [7]:

```
ds = data.copy()
```

Data Overview

In [8]:

```
ds.head(2)
```

Out[8]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0

In [9]:

```
ds.tail(2)
```

Out[9]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCr
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	

In [10]:

```
ds.describe()
```

Out[10]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	

In [11]:

```
ds.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 [12]:

```
ds.shape
```

Out[12]:

```
(10000, 14)
```

In [13]:

```
ds.isnull().sum()
```

Out[13]:

	0
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 [14]:

```
ds.nunique()
```

Out[14]:

	0
RowNumber	10000
CustomerId	10000
Surname	2932
CreditScore	460
Geography	3
Gender	2
Age	70
Tenure	11
Balance	6382
NumOfProducts	4
HasCrCard	2
IsActiveMember	2
EstimatedSalary	9999
Exited	2

dtype: int64

In [15]:

```
#We don't need Row number, CustomerID and Surname for model tunning. So, let's drop it.

ds = ds.drop(["CustomerId","Surname","RowNumber"], axis=1)
```

Exploratory Data Analysis

Why EDA?

Unilateral and bilateral analyses are crucial in Exploratory Data Analysis (EDA) for the following reasons:

Unilateral Analysis (Univariate):

Helps understand individual variable distributions (mean, median, skewness). Identifies outliers or anomalies that may affect modeling. Assists in data transformations (e.g., log or normalization). Provides insights into the range, spread, and central tendency of data.

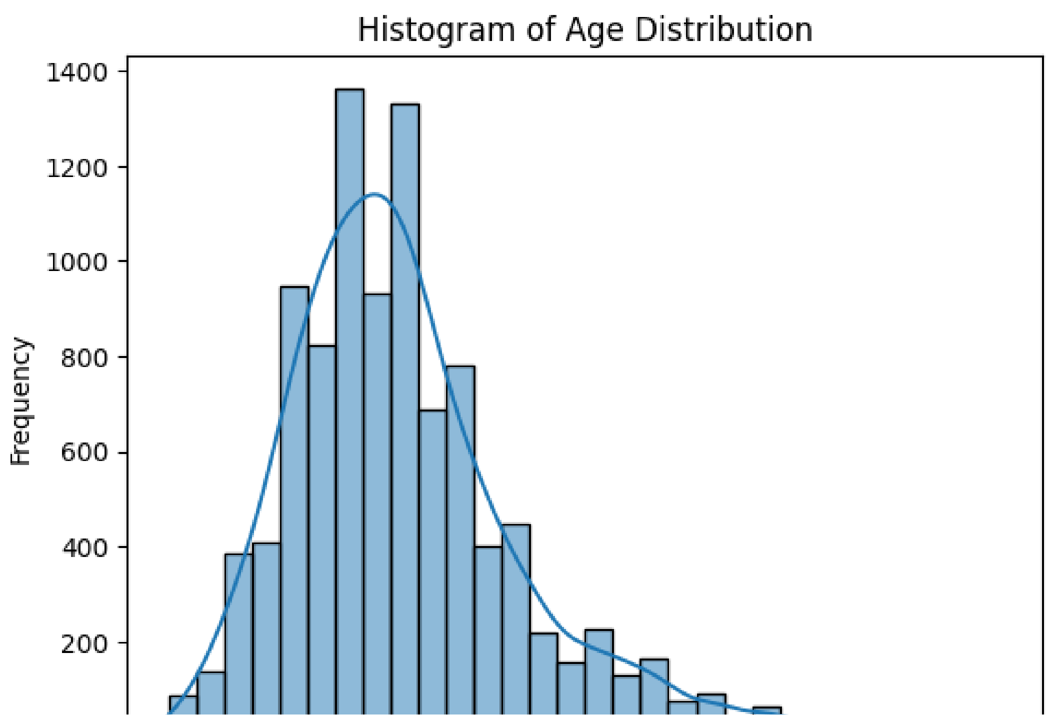
Bilateral Analysis (Bivariate):

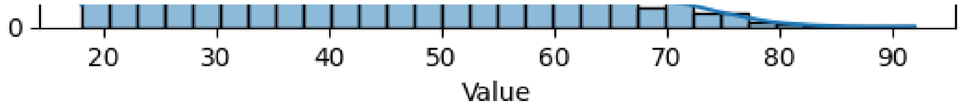
Reveals relationships between two variables (correlations, dependencies). Aids in understanding how variables interact or influence each other. Essential for feature selection and predictive modeling decisions. Supports hypothesis testing and data-driven insights.

Univariate Analysis

In [16]:

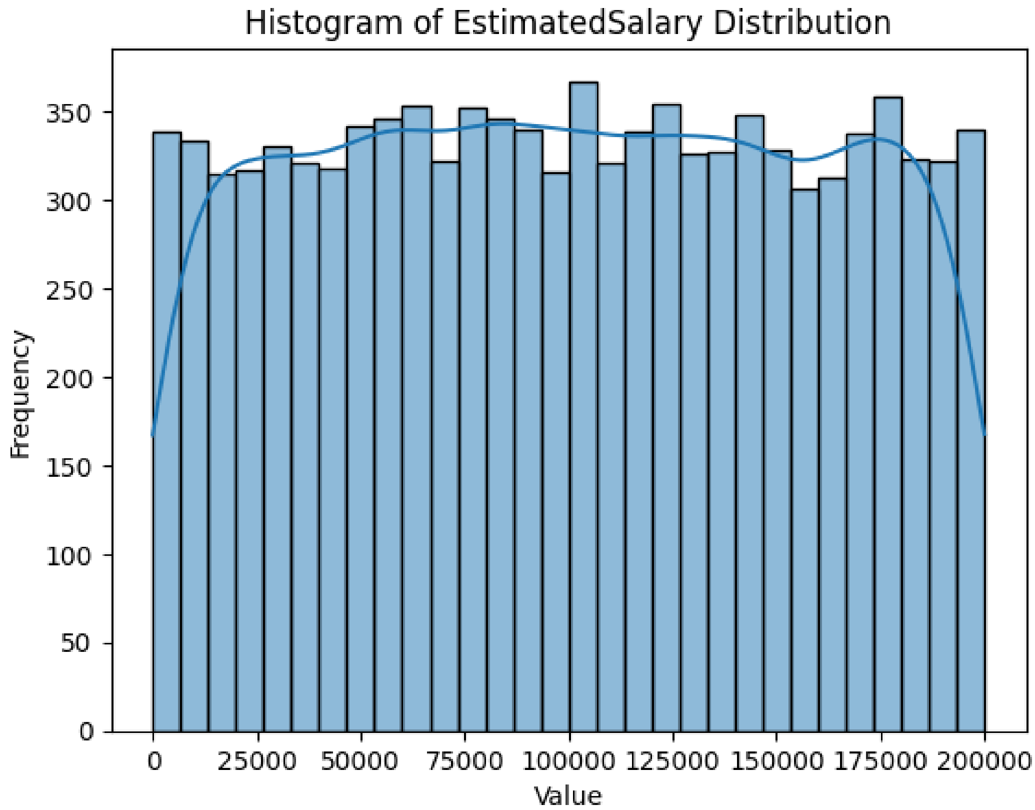
```
# Create a histogram
sns.histplot(ds, x= "Age",bins=30, kde=True)   # kde=True adds a Kernel Density Estimate
curve

# Add titles and labels
plt.title('Histogram of Age Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Display the plot
plt.show()
```





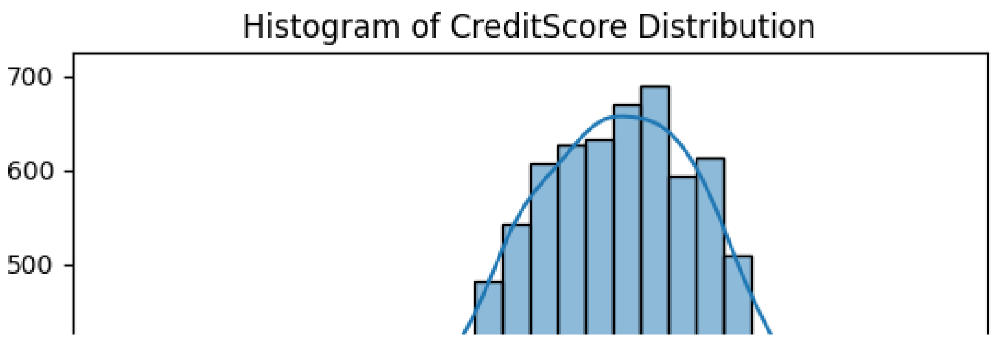
In [17]:

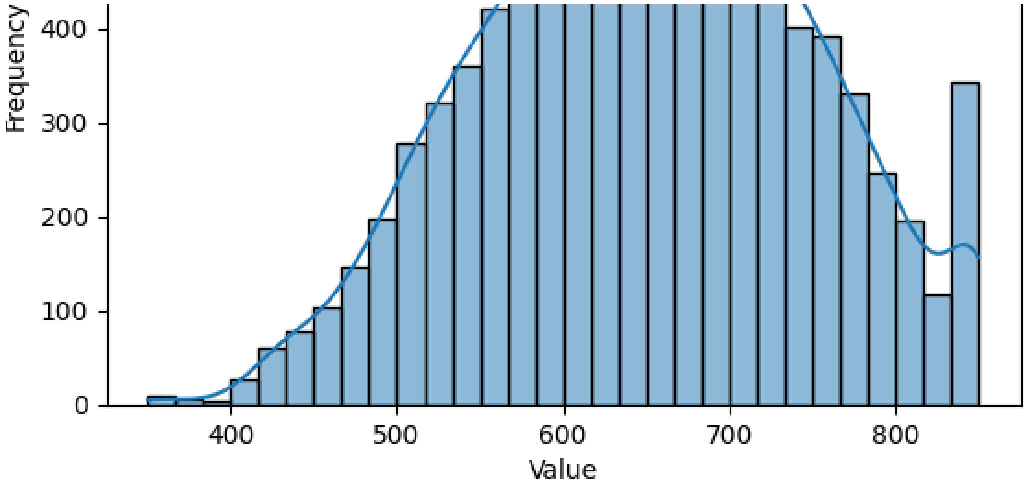
```
# Create a histogram
sns.histplot(ds, x= "EstimatedSalary",bins=30, kde=True)   # kde=True adds a Kernel Densi
ty Estimate curve

# Add titles and labels
plt.title('Histogram of EstimatedSalary Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Display the plot
plt.show()
```



In [18]:

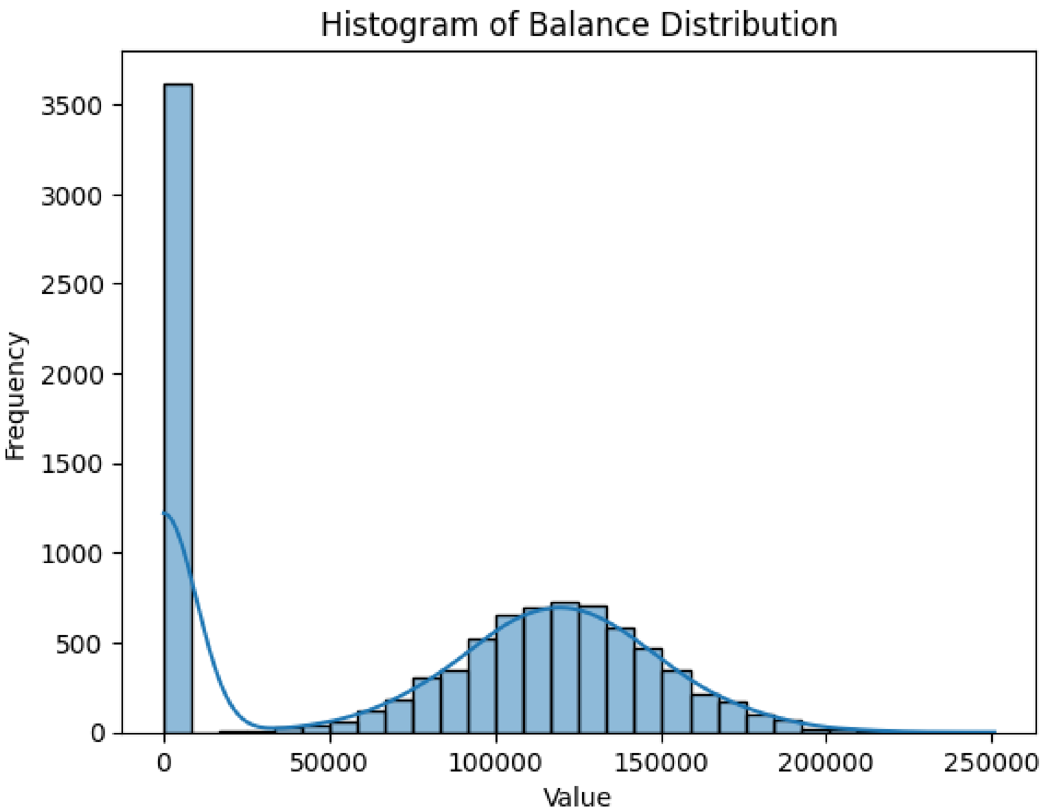
```
# Create a histogram
sns.histplot(ds, x= "CreditScore",bins=30, kde=True)   # kde=True adds a Kernel Density Es
timate curve

# Add titles and labels
plt.title('Histogram of CreditScore Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Display the plot
plt.show()
```





In [19]:

```
# Create a histogram
sns.histplot(ds, x= "Balance",bins=30, kde=True)   # kde=True adds a Kernel Density Estim
ate curve

# Add titles and labels
plt.title('Histogram of Balance Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Display the plot
plt.show()
```



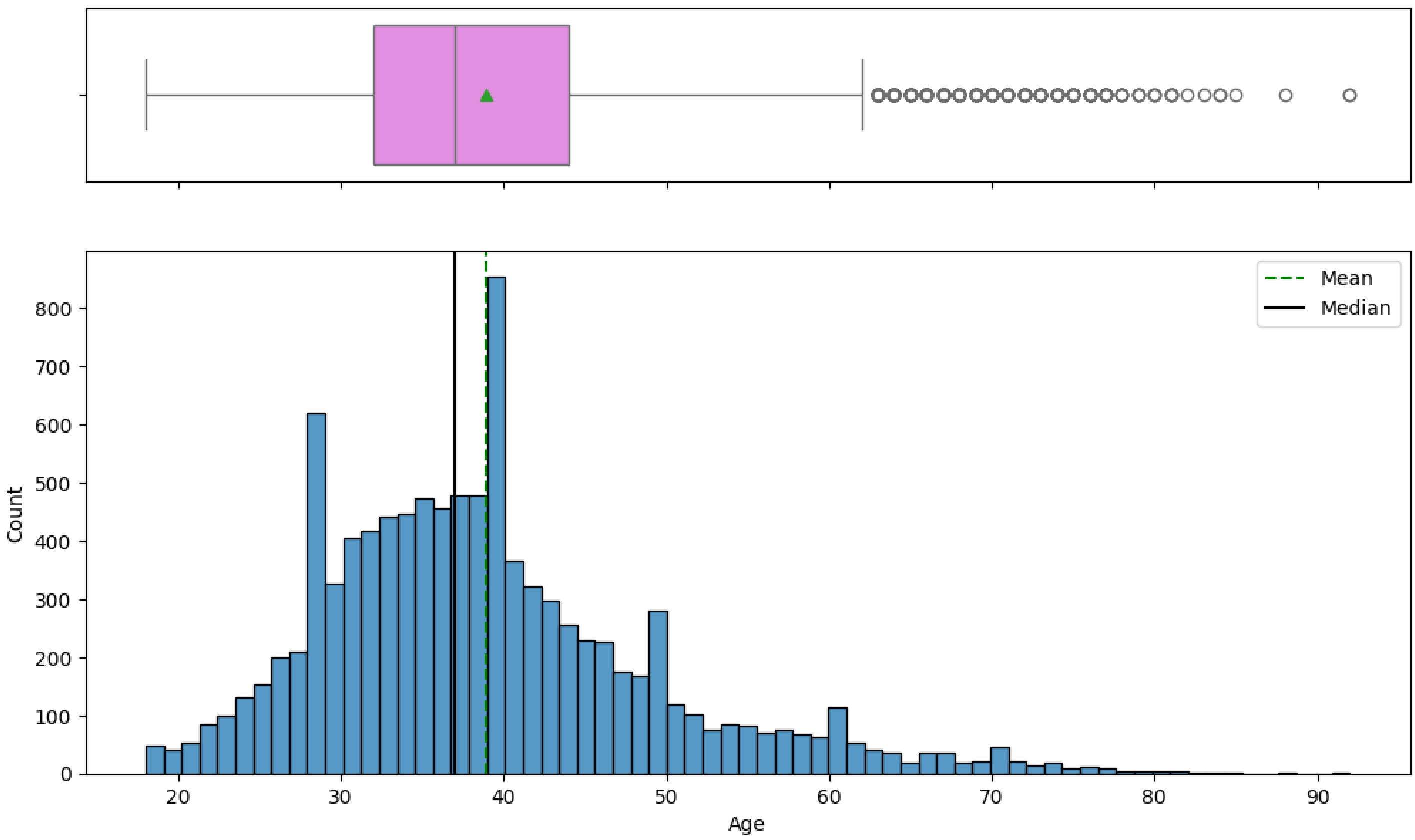
In [20]:

```
def plot_histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Creates a combined boxplot and histogram on the same scale.

    Parameters:
    data (DataFrame): The input data.
    feature (str): The column in the DataFrame to plot.
    figsize (tuple, optional): The size of the figure (default is (12, 7)).
    kde (bool, optional): Whether to display the density curve on the histogram (default
is False).
```

```
    bins (int or sequence, optional): Number of bins for the histogram (default is None).
    """
    # Create a figure with two subplots (boxplot and histogram)
    fig, (ax_box, ax_hist) = plt.subplots(
        nrows=2,         # Number of rows (boxplot on top, histogram below)
        sharex=True,     # Share x-axis between the plots
        gridspec_kw={"height_ratios": (0.25, 0.75)},  # Boxplot takes up 25% of the space
        figsize=figsize
    )

    # Plot the boxplot
    sns.boxplot(
        data=data, x=feature, ax=ax_box, showmeans=True, color="violet"
    )

    # Plot the histogram with optional KDE
    if bins is not None:
        sns.histplot(data=data, x=feature, kde=kde, ax=ax_hist, bins=bins, palette="wint
er")
    else:
        sns.histplot(data=data, x=feature, kde=kde, ax=ax_hist, palette="winter")

    # Add vertical lines for the mean and median
    ax_hist.axvline(data[feature].mean(), color="green", linestyle="--", label="Mean")
    ax_hist.axvline(data[feature].median(), color="black", linestyle="-", label="Median"
)

    # Display the legend
    ax_hist.legend(loc="upper right")

    # Display the plot
    plt.show()
```

In [21]:

```
plot_histogram_boxplot(ds,'Age')
```



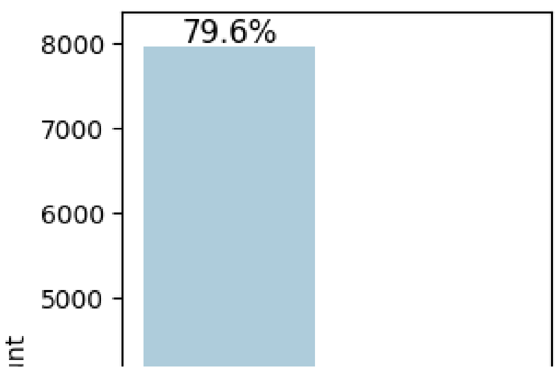
In [22]:

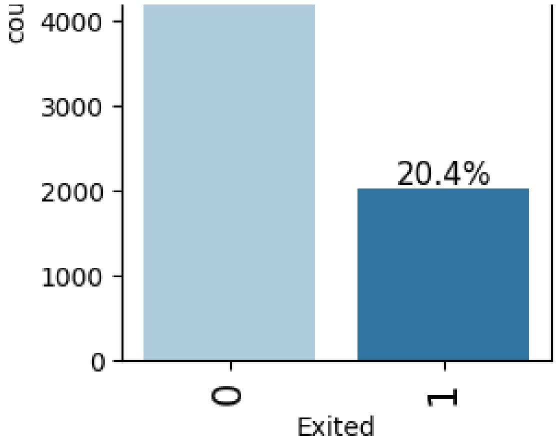
```
def plot_labeled_barplot(data, feature, perc=False, n=None):
    """
    Create a barplot with labels displaying either counts or percentages at the top.
```

```
    Parameters:
    data (DataFrame): The input data.
    feature (str): The column in the DataFrame to plot.
    perc (bool, optional): Whether to display percentages instead of counts. Default is F
alse.
    n (int, optional): Number of top categories to display. If None, all categories are s
hown. Default is None.
    """
    # Total number of rows in the feature column
    total_count = len(data[feature])

    # Number of unique categories in the feature
    unique_categories = data[feature].nunique()

    # Set figure size based on number of categories or top n categories
    if n is None:
        plt.figure(figsize=(unique_categories + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    # Rotate x-axis labels for better visibility
    plt.xticks(rotation=90, fontsize=15)

    # Create the countplot with optional ordering by the top n categories
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    # Annotate bars with either count or percentage
    for p in ax.patches:
        if perc:
            label = "{:.1f}%".format(100 * p.get_height() / total_count)
        else:
            label = p.get_height()

        # Get the position of the label
        x = p.get_x() + p.get_width() / 2
        y = p.get_height()

        # Annotate the label at the top of the bar
        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )

    # Display the plot
    plt.show()
```

In [23]:

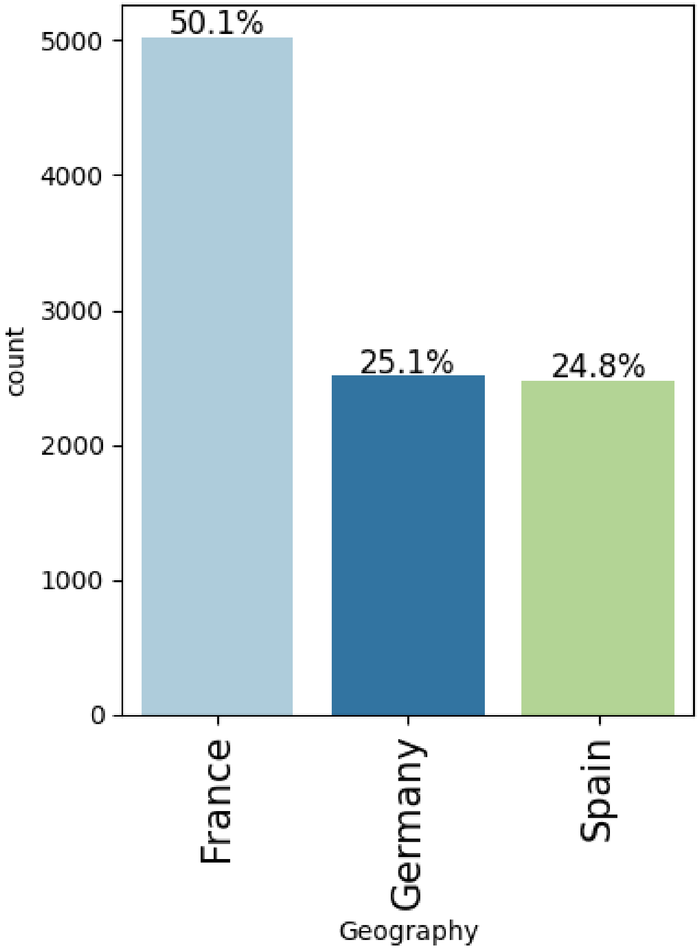
```
plot_labeled_barplot(ds, "Exited", perc=True)
```





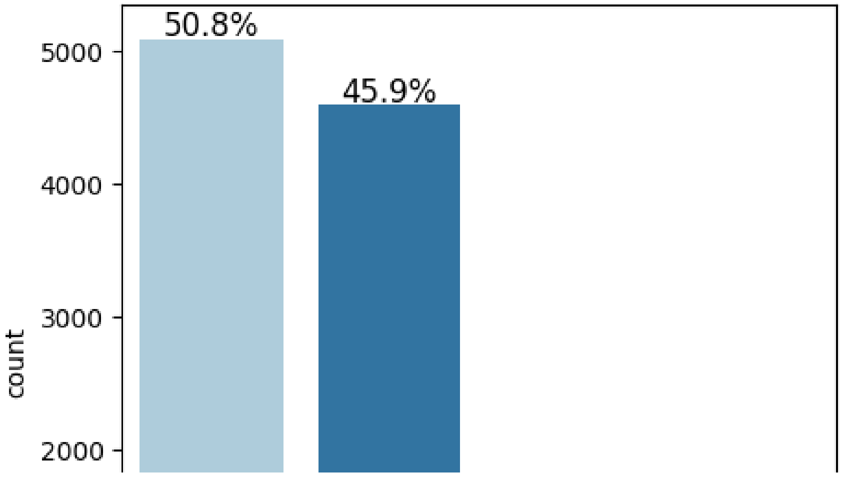
In [24]:

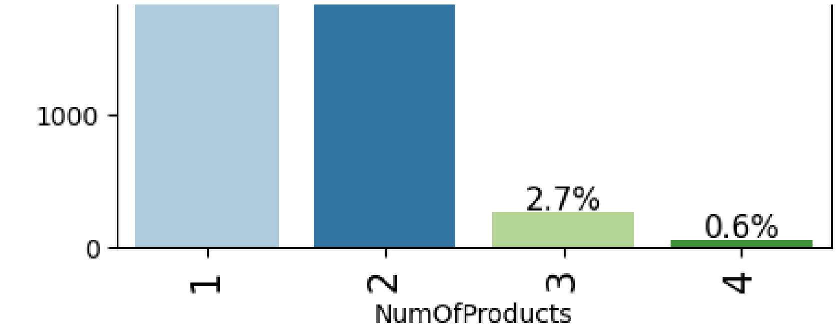
```
plot_labeled_barplot(ds, "Geography", perc=True)
```



In [25]:

```
plot_labeled_barplot(ds, "NumOfProducts", perc=True)
```





Bivariate Analysis

Bilateral Analysis (Bivariate):

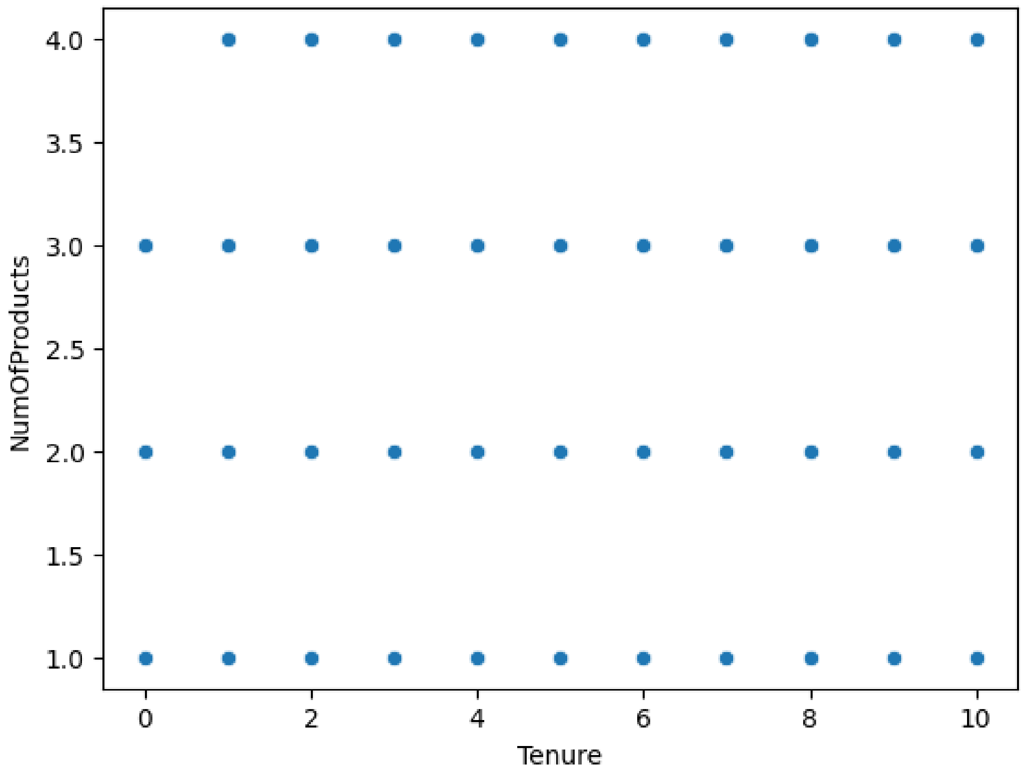
Reveals relationships between two variables (correlations, dependencies). Aids in understanding how variables interact or influence each other. Essential for feature selection and predictive modeling decisions. Supports hypothesis testing and data-driven insights.

In [26]:

```
sns.scatterplot(x='Tenure', y='NumOfProducts', data=ds)
```

Out[26]:

```
<Axes: xlabel='Tenure', ylabel='NumOfProducts'>
```

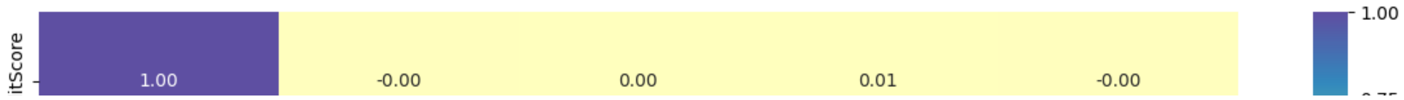


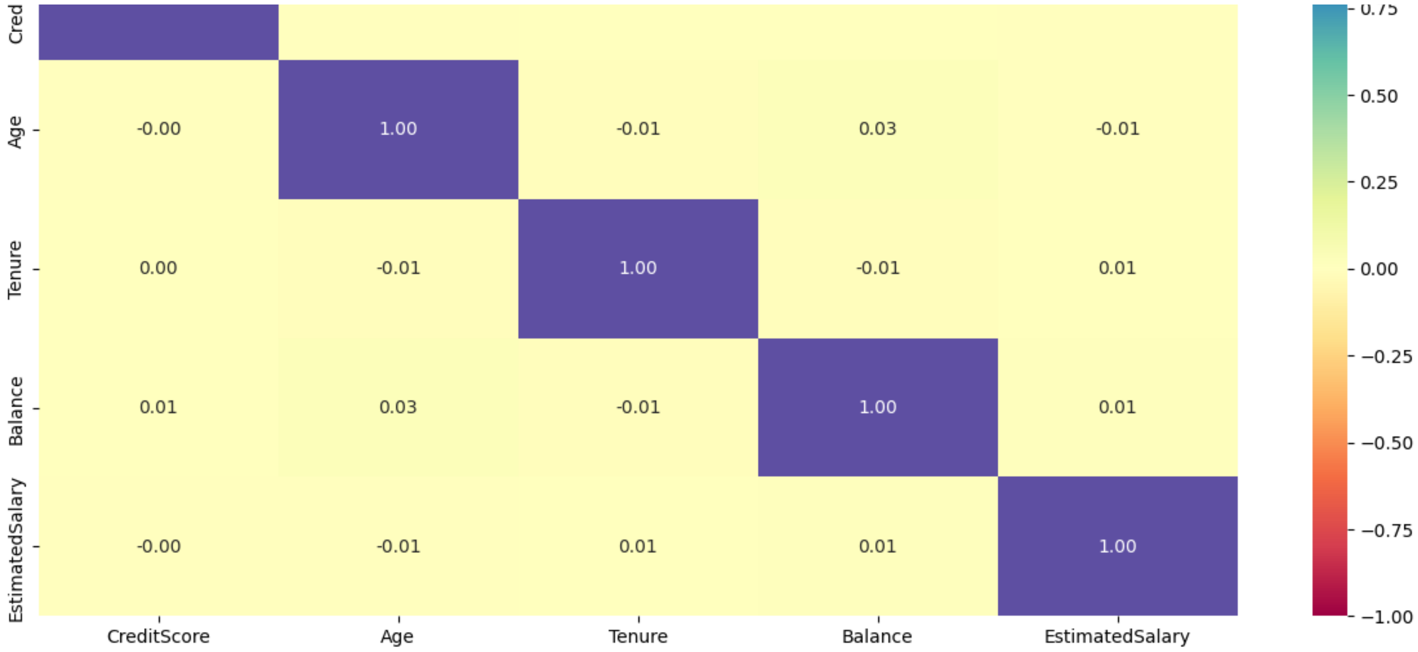
In [27]:

```
cols_list = ["CreditScore","Age","Tenure","Balance","EstimatedSalary"]
```

In [28]:

```
plt.figure(figsize=(15, 7))
sns.heatmap(ds[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral
")
plt.show()
```



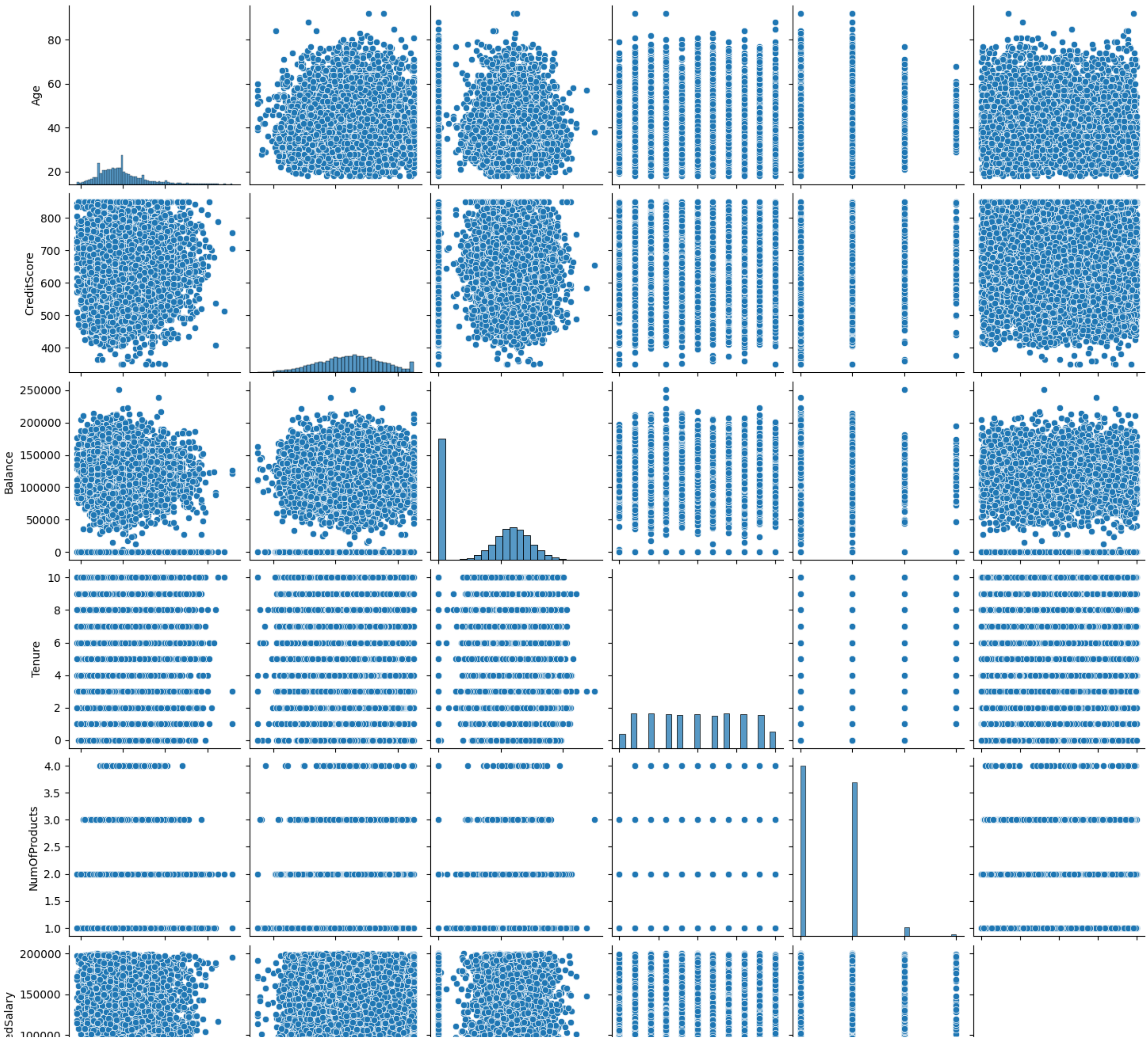


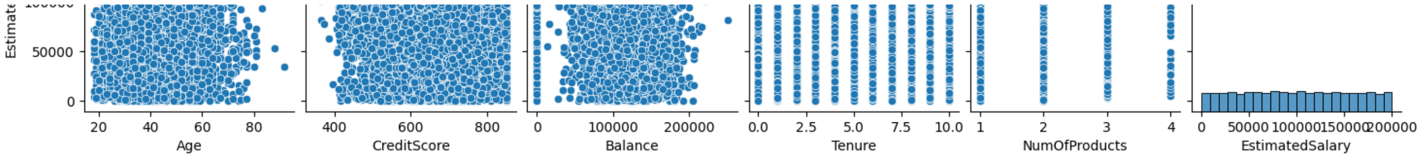
In [29]:

```
sns.pairplot(ds, vars=['Age', 'CreditScore', 'Balance','Tenure','NumOfProducts','EstimatedSalary'])
```

Out[29]:

<seaborn.axisgrid.PairGrid at 0x7a55ac18af80>





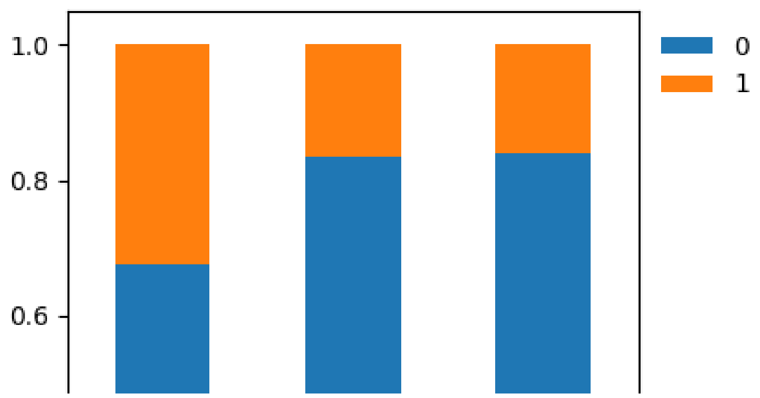
In [30]:

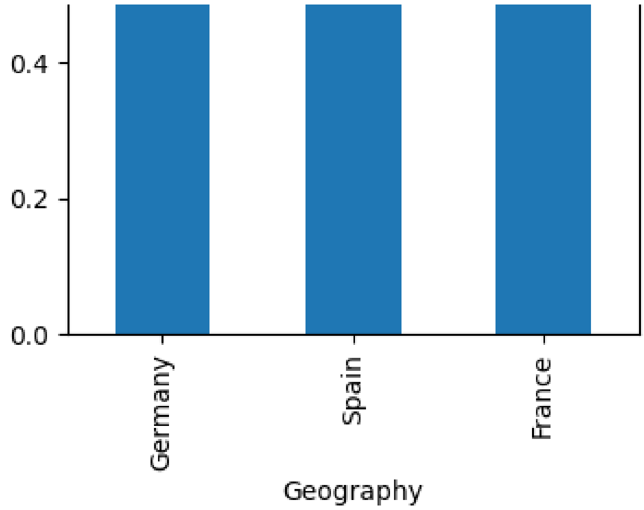
```
def plot_stacked_bar_chart(data, predictor, target):
    """
    Prints the category counts and plots a stacked bar chart.

    Parameters:
    data (DataFrame): The input data.
    predictor (str): The independent variable (column name).
    target (str): The dependent variable (column name).
    """
    # Count unique categories in the predictor
    num_categories = data[predictor].nunique()

    # Determine the sorting order based on the target variable's value counts
    sort_order = data[target].value_counts().index[-1]

    # Create a crosstab of the predictor and target variables
    crosstab = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sort_order, ascending=False
    )
    print(crosstab)
    print("-" * 120)

    # Normalize the crosstab by index (rows) and sort by the target variable
    normalized_crosstab = pd.crosstab(data[predictor], data[target], normalize="index").
sort_values(
        by=sort_order, ascending=False
    )

    # Plot the stacked bar chart
    normalized_crosstab.plot(kind="bar", stacked=True, figsize=(num_categories + 1, 5))

    # Customize legend position
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1), frameon=False)

    # Display the plot
    plt.show()
```

In [31]:

```
plot_stacked_bar_chart(ds, "Geography", "Exited" )
(ds, "Age", "Exited" )
```

```
Exited        0     1    All
Geography
All        7963  2037  10000
Germany    1695   814   2509
France     4204   810   5014
Spain      2064   413   2477
----------------------------------------------------------------------------------------
--------------------------------
```





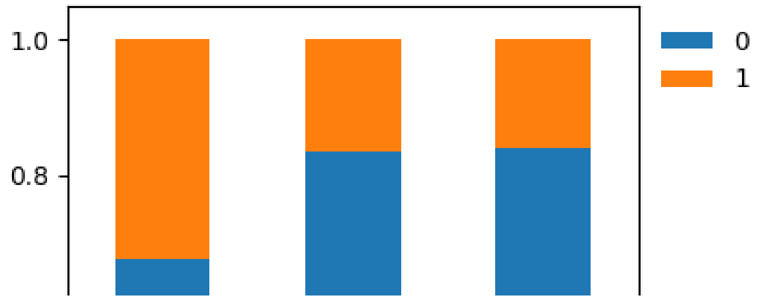
Out[31]:

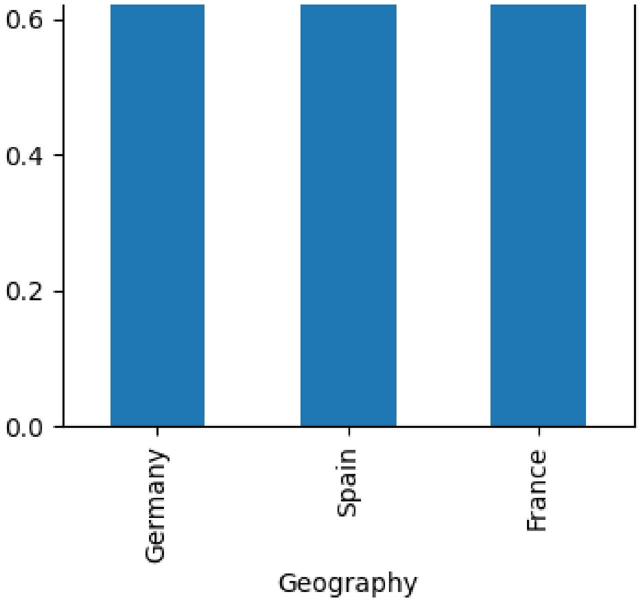
```
(      CreditScore Geography  Gender  Age  Tenure    Balance  NumOfProducts  \
 0             619    France  Female   42       2       0.00              1
 1             608     Spain  Female   41       1   83807.86              1
 2             502    France  Female   42       8  159660.80              3
 3             699    France  Female   39       1       0.00              2
 4             850     Spain  Female   43       2  125510.82              1
 ...           ...       ...     ...  ...     ...        ...            ...
 9995          771    France    Male   39       5       0.00              2
 9996          516    France    Male   35      10   57369.61              1
 9997          709    France  Female   36       7       0.00              1
 9998          772   Germany    Male   42       3   75075.31              2
 9999          792    France  Female   28       4  130142.79              1

       HasCrCard  IsActiveMember  EstimatedSalary  Exited
 0             1               1        101348.88       1
 1             0               1        112542.58       0
 2             1               0        113931.57       1
 3             0               0         93826.63       0
 4             1               1         79084.10       0
 ...         ...             ...              ...     ...
 9995          1               0         96270.64       0
 9996          1               1        101699.77       0
 9997          0               1         42085.58       1
 9998          1               0         92888.52       1
 9999          1               0         38190.78       0

 [10000 rows x 11 columns],
 'Age',
 'Exited')
```

In [32]:

```
plot_stacked_bar_chart(ds, "Geography", "Exited" )
```

```
Exited        0     1    All
Geography
All        7963  2037  10000
Germany    1695   814   2509
France     4204   810   5014
Spain      2064   413   2477
----------------------------------------------------------------------------------------
---------------------------------
```

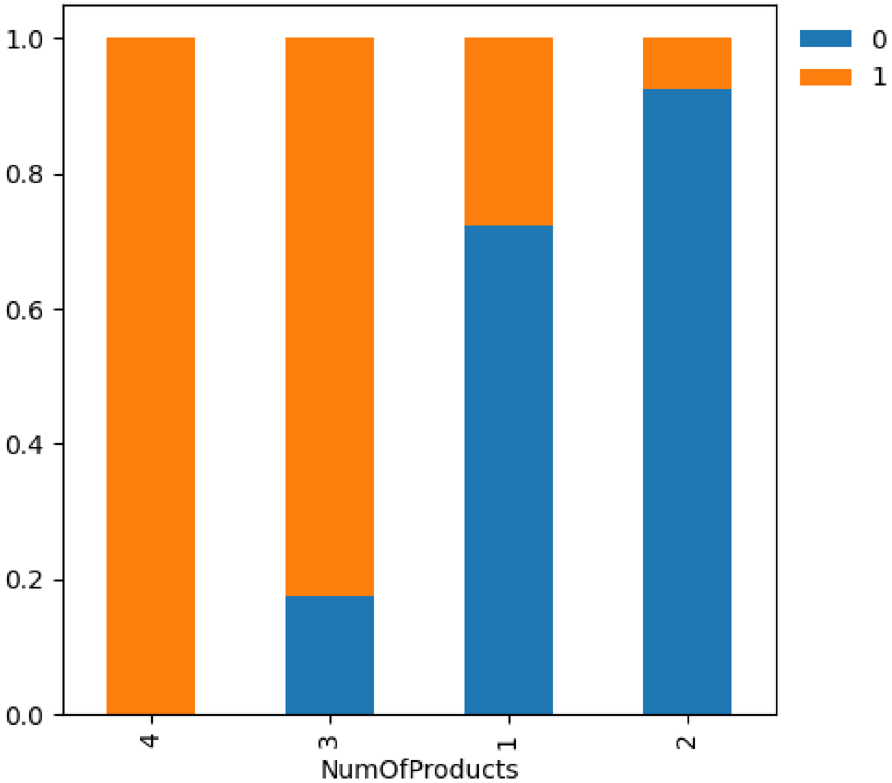




In [33]:

```
plot_stacked_bar_chart(ds, "NumOfProducts", "Exited" )
```

```
Exited             0     1    All
NumOfProducts
All             7963  2037  10000
1               3675  1409   5084
2               4242   348   4590
3                 46   220    266
4                  0    60     60
-------------------------------------------------------------------------------------
--------------------------------
```

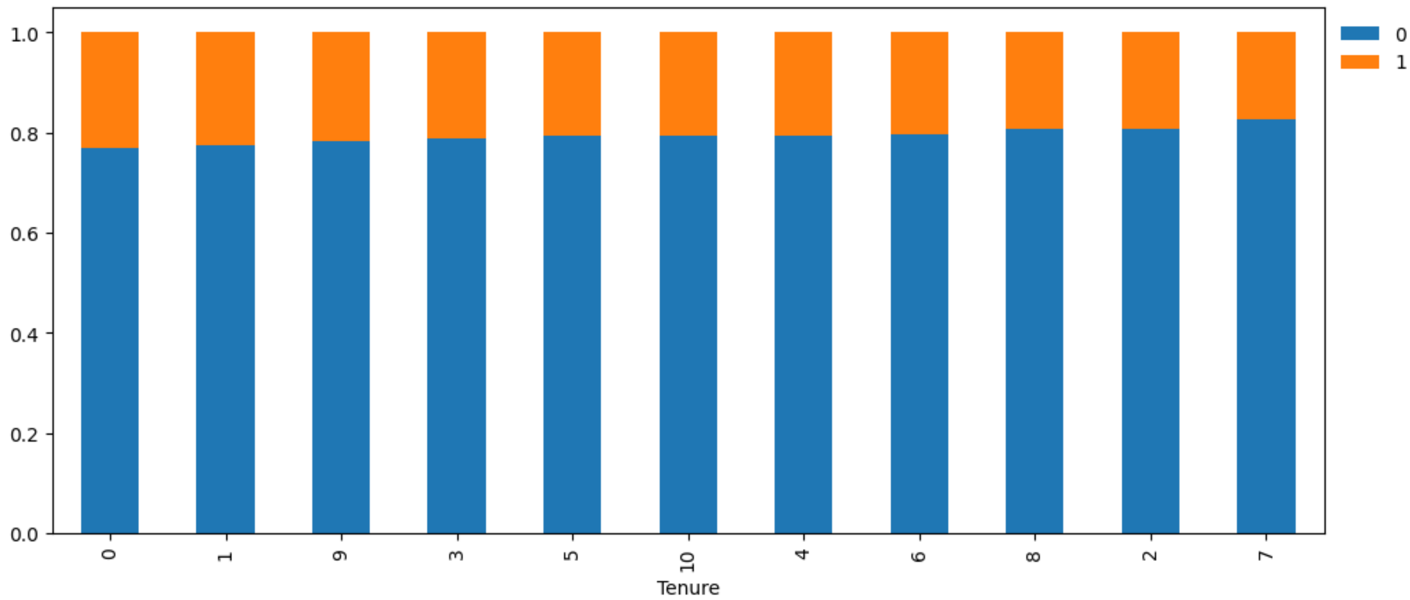


In [34]:

```
plot_stacked_bar_chart(ds, "Tenure", "Exited" )
```

```
Exited     0     1    All
Tenure
All     7963  2037  10000
1        803   232   1035
3        796   213   1009
9        771   213    984
```

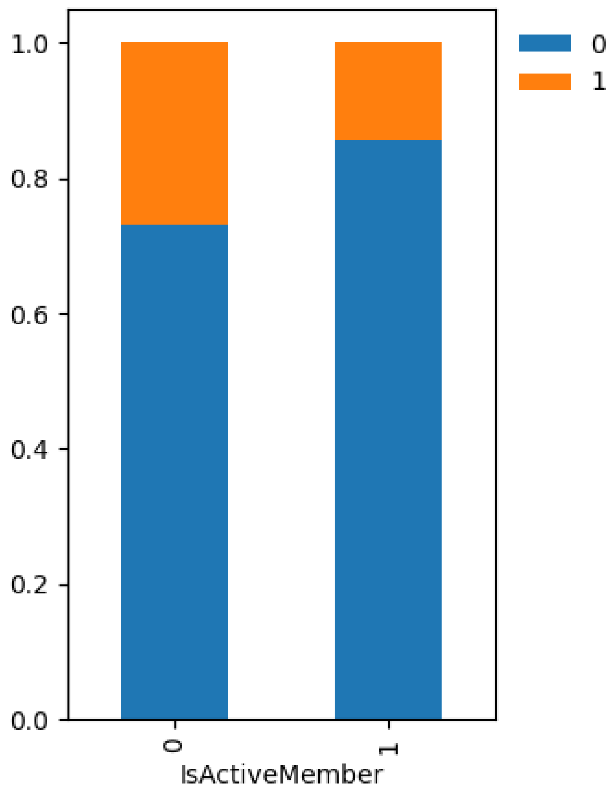
```
5          803   209   1012
4          786   203    989
2          847   201   1048
8          828   197   1025
6          771   196    967
7          851   177   1028
10         389   101    490
0          318    95    413
-----------------------------------------------------------------------------------------
---------------------------------
```



In [35]:

```
plot_stacked_bar_chart(ds, "IsActiveMember", "Exited" )
```

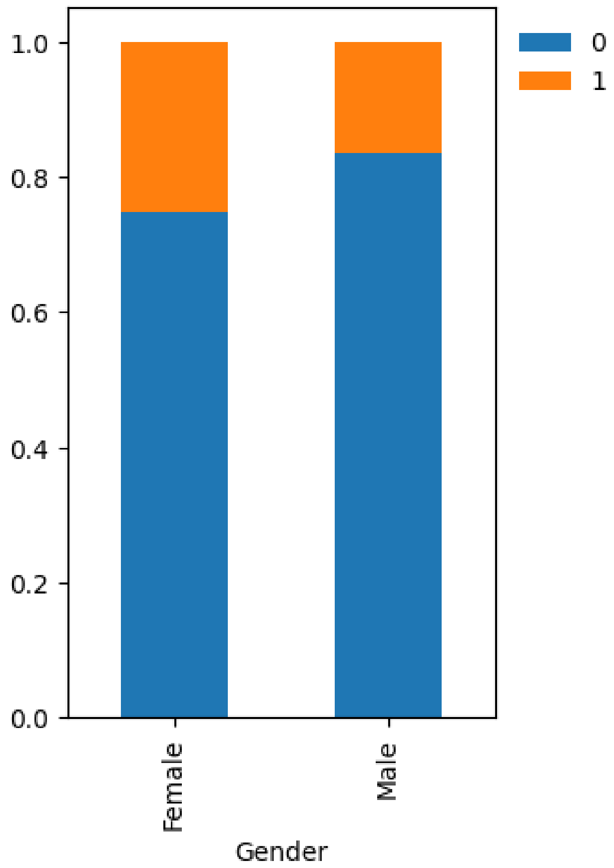
```
Exited             0     1    All
IsActiveMember
All             7963  2037  10000
0               3547  1302   4849
1               4416   735   5151
-----------------------------------------------------------------------------------------
---------------------------------
```



In [36]:

```
plot_stacked_bar_chart(ds, "Gender", "Exited" )
```

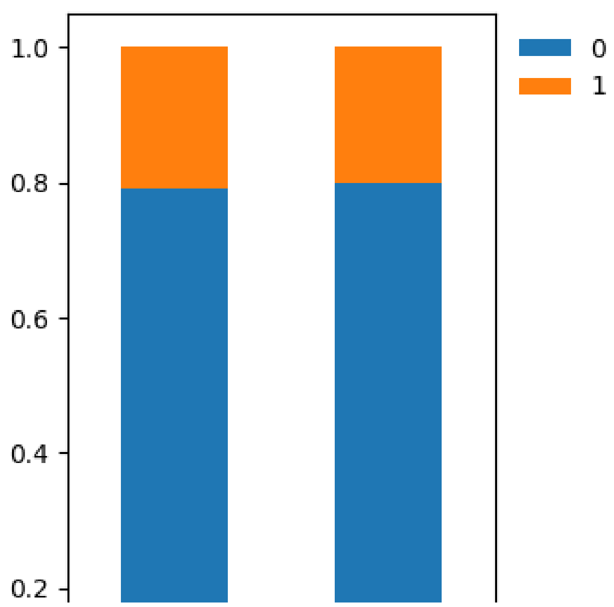
```
Exited     0     1    All
Gender
All     7963  2037  10000
Female  3404  1139   4543
Male    4559   898   5457
----------------------------------------------------------------------------------------
---------------------------------
```

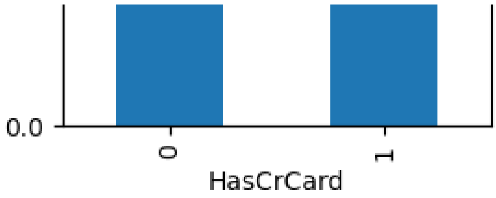


In [37]:

```
plot_stacked_bar_chart(ds, "HasCrCard", "Exited" )
```

```
Exited        0     1    All
HasCrCard
All        7963  2037  10000
1          5631  1424   7055
0          2332   613   2945
----------------------------------------------------------------------------------------
---------------------------------
```





Data Preprocessing

Dummy Variable Creation

The code is needed to preprocess categorical data (Geography and Gender) by converting them into numerical values using one-hot encoding, which creates binary columns for each category. Additionally, it converts integer data types to float for consistency in numerical operations, ensuring the dataset is ready for machine learning models.

In [38]:

```
# We have Geography and Gender as Object data types. We need to convert those object data
types to numerical values.
# We also need to convert all int data types to float
ds = pd.get_dummies(ds,columns=ds.select_dtypes(include=["object"]).columns.tolist(),dro
p_first=True)
ds = ds.astype(float)
ds.head()
```

Out[38]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_Ge
0	619.0	42.0	2.0	0.00	1.0	1.0	1.0	101348.88	1.0	
1	608.0	41.0	1.0	83807.86	1.0	0.0	1.0	112542.58	0.0	
2	502.0	42.0	8.0	159660.80	3.0	1.0	0.0	113931.57	1.0	
3	699.0	39.0	1.0	0.00	2.0	0.0	0.0	93826.63	0.0	
4	850.0	43.0	2.0	125510.82	1.0	1.0	1.0	79084.10	0.0	

In [39]:

```
ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   CreditScore        10000 non-null  float64
 1   Age                10000 non-null  float64
 2   Tenure             10000 non-null  float64
 3   Balance            10000 non-null  float64
 4   NumOfProducts      10000 non-null  float64
 5   HasCrCard          10000 non-null  float64
 6   IsActiveMember     10000 non-null  float64
 7   EstimatedSalary    10000 non-null  float64
 8   Exited             10000 non-null  float64
 9   Geography_Germany  10000 non-null  float64
 10  Geography_Spain    10000 non-null  float64
 11  Gender_Male        10000 non-null  float64
dtypes: float64(12)
memory usage: 937.6 KB
```

Train-validation-test Split

Lets drop "Exited" and split the data into train, validation and test set.

In [40]:

```
X = ds.drop(['Exited'],axis=1)
y = ds['Exited']
```

Lets split the data into 80-20%

In [41]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
= 42,stratify=y,shuffle = True)
```

Lets split the training set into validation and training set

In [42]:

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.2, ran
dom_state = 42,stratify=y_train, shuffle = True)
```

In [43]:

```
print("training set shape : ", X_train.shape)
print("validation set shape : ", X_val.shape)
print("test set shape : ", X_test.shape)
```

```
training set shape :  (6400, 11)
validation set shape :  (1600, 11)
test set shape :  (2000, 11)
```

From 10000 sets, we divide 20% into test set and 20% of 80% we keep as validation set

In [44]:

```
print("training set shape : ", y_train.shape)
print("validation set shape : ", y_val.shape)
print("test set shape : ", y_test.shape)
```

```
training set shape :  (6400,)
validation set shape :  (1600,)
test set shape :  (2000,)
```

Data Normalization

Why standardization?:

Many machine learning algorithms (such as those based on distance metrics like k-NN or algorithms involving gradient descent, like linear regression, logistic regression, and neural networks) work better when the features are on similar scales. Features with larger ranges can disproportionately influence the model, potentially leading to inaccurate predictions. Standardizing ensures all features contribute equally to the model's training process. python

StandardScaler():

This line creates an instance of the StandardScaler class from the sklearn.preprocessing library. The StandardScaler is used to standardize the features of the dataset. Standardization is a process that removes the mean and scales the data to have unit variance, resulting in a transformed feature with a mean of 0 and a standard deviation of 1.

we need to normalize the data since all the data values on a different scale

In [45]:

```
column_list = ["Age","Tenure","Balance","EstimatedSalary","CreditScore"]
```

In [46]:

```
sc = StandardScaler()
X_train[column_list] = sc.fit_transform(X_train[column_list])
X_val[column_list] = sc.transform(X_val[column_list])
X_test[column_list] = sc.transform(X_test[column_list])
```

Model Building

Why Recall is an Important Metric:

In the context of the problem statement related to Customer Churn prediction, the most important metric among F1, accuracy, and recall would likely be Recall. Here’s why:

Recall measures the ability of the model to correctly identify customers who will leave (true positives). In a churn prediction scenario, it's crucial to minimize the number of customers who are predicted to stay but will actually leave (false negatives). Missing a customer who is about to churn can be costly for the bank.

Accuracy might not be the best metric here because if most customers stay, a model predicting "stay" for everyone would still have high accuracy, but it wouldn't help in identifying those who are likely to churn. This would not be useful for the bank's efforts to retain customers.

F1 Score is a balance between precision and recall. It becomes useful if both false positives (predicting churn when a customer stays) and false negatives (predicting stay when a customer churns) are equally important. However, if the focus is on minimizing churn loss, recall would be prioritized.

Thus, recall is key to ensuring that the bank can focus its efforts on retaining customers who are most likely to leave.

Lets write a function to create a confusion matrix. I'm using the same code from the previous projects.

In [47]:

```
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")
```

Model Evaluation Criterion

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

-

In [48]:

```
trainning_perf = pd.DataFrame(columns=["recall"])
validation_perf = pd.DataFrame(columns=["recall"])
```

Neural Network with SGD Optimizer

SGD (Stochastic Gradient Descent) is an optimization algorithm commonly used in training machine learning models, including neural networks. It is a variant of gradient descent, but instead of using the entire dataset to calculate the gradient of the loss function at each step, it uses only a single data point (or a small batch) at a time.

Lets use the same seed so that we can re-generate the output

In [49]:

```
backend.clear_session()
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

backend.clear_session() is used to clear the current Keras session, essentially resetting the state of the Keras backend. This is helpful in the following situations:

When training models in a loop (e.g., cross-validation), clearing the session helps to avoid memory buildup from previous models.

It releases memory and resources associated with previous models, ensuring that subsequent models are not impacted by the previous model's memory usage.

It also helps avoid potential conflicts when you are defining and training multiple models within the same program.

Important: This does not affect the TensorFlow session directly, but ensures Keras releases resources properly.

model_0 = Sequential() initializes the neural network. Adds input layer with 64 neurons and ReLU activation. Adds hidden layer with 32 neurons and ReLU activation. Adds output layer with 1 neuron and sigmoid activation (for binary classification).

In [50]:

```
model_0 = Sequential()
model_0.add(Dense(64, activation='relu', input_dim = X_train.shape[1]))
model_0.add(Dense(32, activation='relu'))
model_0.add(Dense(1, activation = 'sigmoid'))
```

In the context of the code optimizer = tf.keras.optimizers.SGD(0.001), the value 0.001 represents the learning rate of the Stochastic Gradient Descent (SGD) optimizer.

Explanation of Learning Rate: The learning rate is a hyperparameter that controls how much the model's weights are adjusted with respect to the loss gradient during training. Specifically, it determines the size of the steps the optimizer takes when updating the model’s weights to minimize the loss function.

A smaller learning rate (e.g., 0.001) means that the model takes smaller steps, which might result in more precise convergence but can also lead to slower training.

In [51]:

```
optimizer = tf.keras.optimizers.SGD(0.001)
```

```
metric = keras.metrics.Recall()
```

In [52]:

```
model_0.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [53]:

```
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 [54]:

```
model0_performace = model_0.fit(
    X_train, y_train,
    batch_size=64,
    validation_data=(X_val,y_val),
    epochs=100,
    verbose=1
)
```

```
Epoch 1/100
100/100 [==============================] - 2s 10ms/step - loss: 0.6107 - recall: 0.1449 -
val_loss: 0.5945 - val_recall: 0.1472
Epoch 2/100
100/100 [==============================] - 1s 6ms/step - loss: 0.5865 - recall: 0.0844 -
val_loss: 0.5732 - val_recall: 0.0859
Epoch 3/100
100/100 [==============================] - 1s 5ms/step - loss: 0.5678 - recall: 0.0498 -
val_loss: 0.5568 - val_recall: 0.0460
Epoch 4/100
100/100 [==============================] - 1s 5ms/step - loss: 0.5533 - recall: 0.0230 -
val_loss: 0.5440 - val_recall: 0.0337
Epoch 5/100
100/100 [==============================] - 1s 5ms/step - loss: 0.5420 - recall: 0.0130 -
val_loss: 0.5339 - val_recall: 0.0153
Epoch 6/100
100/100 [==============================] - 1s 6ms/step - loss: 0.5330 - recall: 0.0046 -
val_loss: 0.5258 - val_recall: 0.0061
Epoch 7/100
100/100 [==============================] - 1s 5ms/step - loss: 0.5258 - recall: 0.0031 -
val_loss: 0.5192 - val_recall: 0.0000e+00
Epoch 8/100
100/100 [==============================] - 0s 5ms/step - loss: 0.5199 - recall: 7.6687e-0
4 - val_loss: 0.5138 - val_recall: 0.0000e+00
Epoch 9/100
100/100 [==============================] - 1s 5ms/step - loss: 0.5150 - recall: 0.0000e+0
0 - val_loss: 0.5093 - val_recall: 0.0000e+00
Epoch 10/100
100/100 [==============================] - 0s 4ms/step - loss: 0.5109 - recall: 0.0000e+0
0 - val_loss: 0.5055 - val_recall: 0.0000e+00
Epoch 11/100
100/100 [==============================] - 0s 3ms/step - loss: 0.5073 - recall: 0.0000e+0
0 - val_loss: 0.5022 - val_recall: 0.0000e+00
Epoch 12/100
100/100 [==============================] - 0s 3ms/step - loss: 0.5043 - recall: 0.0000e+0
0 - val_loss: 0.4992 - val_recall: 0.0000e+00
```

```
Epoch 13/100
100/100 [==============================] - 0s 4ms/step - loss: 0.5015 - recall: 0.0000e+0
0 - val_loss: 0.4966 - val_recall: 0.0000e+00
Epoch 14/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4990 - recall: 0.0000e+0
0 - val_loss: 0.4942 - val_recall: 0.0000e+00
Epoch 15/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4967 - recall: 0.0000e+0
0 - val_loss: 0.4920 - val_recall: 0.0000e+00
Epoch 16/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4946 - recall: 0.0000e+0
0 - val_loss: 0.4899 - val_recall: 0.0000e+00
Epoch 17/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4926 - recall: 0.0000e+0
0 - val_loss: 0.4880 - val_recall: 0.0000e+00
Epoch 18/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4908 - recall: 0.0000e+0
0 - val_loss: 0.4861 - val_recall: 0.0000e+00
Epoch 19/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4890 - recall: 0.0000e+0
0 - val_loss: 0.4844 - val_recall: 0.0000e+00
Epoch 20/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4873 - recall: 0.0000e+0
0 - val_loss: 0.4827 - val_recall: 0.0000e+00
Epoch 21/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4856 - recall: 0.0000e+0
0 - val_loss: 0.4811 - val_recall: 0.0000e+00
Epoch 22/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4841 - recall: 0.0000e+0
0 - val_loss: 0.4795 - val_recall: 0.0000e+00
Epoch 23/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4826 - recall: 0.0000e+0
0 - val_loss: 0.4780 - val_recall: 0.0000e+00
Epoch 24/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4811 - recall: 0.0000e+0
0 - val_loss: 0.4766 - val_recall: 0.0000e+00
Epoch 25/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4797 - recall: 0.0000e+0
0 - val_loss: 0.4752 - val_recall: 0.0000e+00
Epoch 26/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4783 - recall: 0.0000e+0
0 - val_loss: 0.4738 - val_recall: 0.0000e+00
Epoch 27/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4769 - recall: 0.0000e+0
0 - val_loss: 0.4725 - val_recall: 0.0000e+00
Epoch 28/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4756 - recall: 0.0000e+0
0 - val_loss: 0.4712 - val_recall: 0.0000e+00
Epoch 29/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4744 - recall: 0.0000e+0
0 - val_loss: 0.4699 - val_recall: 0.0000e+00
Epoch 30/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4731 - recall: 0.0000e+0
0 - val_loss: 0.4687 - val_recall: 0.0000e+00
Epoch 31/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4719 - recall: 7.6687e-0
4 - val_loss: 0.4675 - val_recall: 0.0000e+00
Epoch 32/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4708 - recall: 7.6687e-0
4 - val_loss: 0.4663 - val_recall: 0.0000e+00
Epoch 33/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4696 - recall: 7.6687e-0
4 - val_loss: 0.4652 - val_recall: 0.0000e+00
Epoch 34/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4685 - recall: 7.6687e-0
4 - val_loss: 0.4641 - val_recall: 0.0000e+00
Epoch 35/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4674 - recall: 7.6687e-0
4 - val_loss: 0.4630 - val_recall: 0.0000e+00
Epoch 36/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4664 - recall: 7.6687e-0
4 - val_loss: 0.4620 - val_recall: 0.0031
```

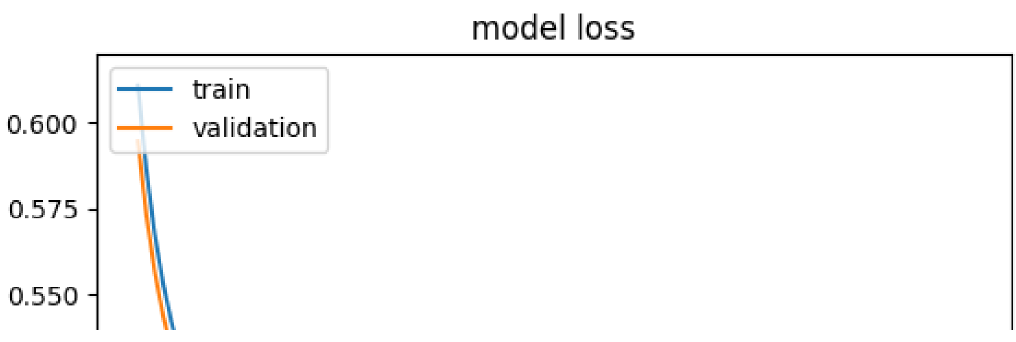
```
Epoch 37/100
100/100 [==============================] - 0s 5ms/step - loss: 0.4654 - recall: 0.0023 -
val_loss: 0.4610 - val_recall: 0.0031
Epoch 38/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4644 - recall: 0.0023 -
val_loss: 0.4600 - val_recall: 0.0031
Epoch 39/100
100/100 [==============================] - 0s 5ms/step - loss: 0.4634 - recall: 0.0023 -
val_loss: 0.4591 - val_recall: 0.0031
Epoch 40/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4624 - recall: 0.0031 -
val_loss: 0.4581 - val_recall: 0.0031
Epoch 41/100
100/100 [==============================] - 1s 5ms/step - loss: 0.4615 - recall: 0.0031 -
val_loss: 0.4572 - val_recall: 0.0031
Epoch 42/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4606 - recall: 0.0031 -
val_loss: 0.4563 - val_recall: 0.0031
Epoch 43/100
100/100 [==============================] - 0s 5ms/step - loss: 0.4597 - recall: 0.0038 -
val_loss: 0.4555 - val_recall: 0.0031
Epoch 44/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4588 - recall: 0.0038 -
val_loss: 0.4546 - val_recall: 0.0061
Epoch 45/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4580 - recall: 0.0038 -
val_loss: 0.4538 - val_recall: 0.0061
Epoch 46/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4571 - recall: 0.0054 -
val_loss: 0.4530 - val_recall: 0.0061
Epoch 47/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4563 - recall: 0.0054 -
val_loss: 0.4522 - val_recall: 0.0092
Epoch 48/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4555 - recall: 0.0077 -
val_loss: 0.4515 - val_recall: 0.0123
Epoch 49/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4548 - recall: 0.0084 -
val_loss: 0.4507 - val_recall: 0.0153
Epoch 50/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4540 - recall: 0.0084 -
val_loss: 0.4500 - val_recall: 0.0153
Epoch 51/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4533 - recall: 0.0084 -
val_loss: 0.4493 - val_recall: 0.0153
Epoch 52/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4526 - recall: 0.0084 -
val_loss: 0.4487 - val_recall: 0.0153
Epoch 53/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4519 - recall: 0.0092 -
val_loss: 0.4480 - val_recall: 0.0184
Epoch 54/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4512 - recall: 0.0092 -
val_loss: 0.4473 - val_recall: 0.0184
Epoch 55/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4505 - recall: 0.0115 -
val_loss: 0.4467 - val_recall: 0.0184
Epoch 56/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4498 - recall: 0.0115 -
val_loss: 0.4461 - val_recall: 0.0184
Epoch 57/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4492 - recall: 0.0123 -
val_loss: 0.4455 - val_recall: 0.0215
Epoch 58/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4486 - recall: 0.0138 -
val_loss: 0.4450 - val_recall: 0.0215
Epoch 59/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4479 - recall: 0.0161 -
val_loss: 0.4444 - val_recall: 0.0215
Epoch 60/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4473 - recall: 0.0199 -
val_loss: 0.4439 - val_recall: 0.0245
```

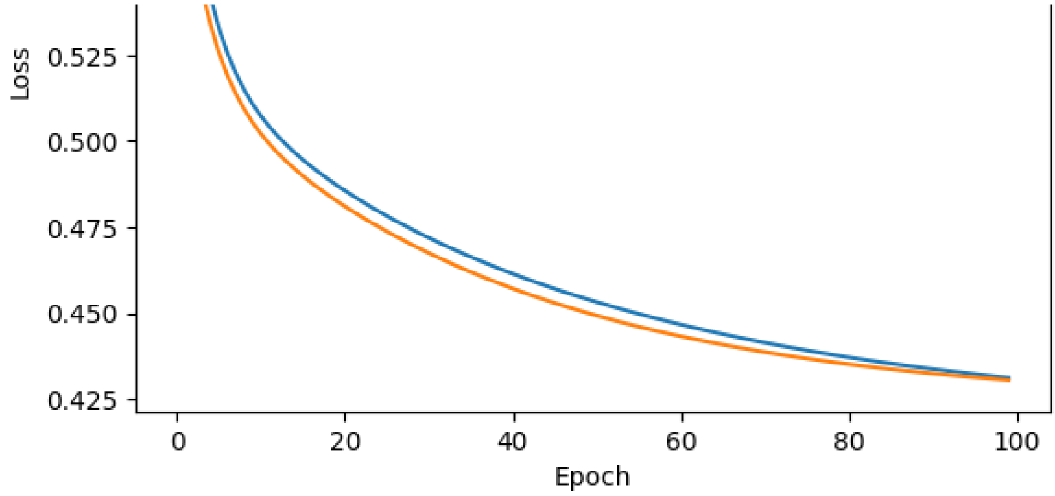
```
Epoch 61/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4467 - recall: 0.0199 -
val_loss: 0.4433 - val_recall: 0.0307
Epoch 62/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4462 - recall: 0.0230 -
val_loss: 0.4428 - val_recall: 0.0307
Epoch 63/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4456 - recall: 0.0253 -
val_loss: 0.4423 - val_recall: 0.0307
Epoch 64/100
100/100 [==============================] - 0s 5ms/step - loss: 0.4450 - recall: 0.0261 -
val_loss: 0.4418 - val_recall: 0.0337
Epoch 65/100
100/100 [==============================] - 0s 5ms/step - loss: 0.4445 - recall: 0.0299 -
val_loss: 0.4414 - val_recall: 0.0337
Epoch 66/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4440 - recall: 0.0314 -
val_loss: 0.4409 - val_recall: 0.0368
Epoch 67/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4434 - recall: 0.0337 -
val_loss: 0.4404 - val_recall: 0.0399
Epoch 68/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4429 - recall: 0.0337 -
val_loss: 0.4400 - val_recall: 0.0399
Epoch 69/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4424 - recall: 0.0345 -
val_loss: 0.4396 - val_recall: 0.0460
Epoch 70/100
100/100 [==============================] - 0s 3ms/step - loss: 0.4419 - recall: 0.0360 -
val_loss: 0.4392 - val_recall: 0.0491
Epoch 71/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4415 - recall: 0.0391 -
val_loss: 0.4388 - val_recall: 0.0521
Epoch 72/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4410 - recall: 0.0414 -
val_loss: 0.4384 - val_recall: 0.0552
Epoch 73/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4406 - recall: 0.0422 -
val_loss: 0.4380 - val_recall: 0.0583
Epoch 74/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4401 - recall: 0.0475 -
val_loss: 0.4376 - val_recall: 0.0613
Epoch 75/100
100/100 [==============================] - 1s 5ms/step - loss: 0.4397 - recall: 0.0491 -
val_loss: 0.4372 - val_recall: 0.0644
Epoch 76/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4392 - recall: 0.0521 -
val_loss: 0.4369 - val_recall: 0.0644
Epoch 77/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4388 - recall: 0.0544 -
val_loss: 0.4365 - val_recall: 0.0644
Epoch 78/100
100/100 [==============================] - 0s 5ms/step - loss: 0.4384 - recall: 0.0560 -
val_loss: 0.4362 - val_recall: 0.0736
Epoch 79/100
100/100 [==============================] - 1s 9ms/step - loss: 0.4380 - recall: 0.0621 -
val_loss: 0.4359 - val_recall: 0.0736
Epoch 80/100
100/100 [==============================] - 1s 10ms/step - loss: 0.4376 - recall: 0.0644 -
val_loss: 0.4355 - val_recall: 0.0736
Epoch 81/100
100/100 [==============================] - 1s 10ms/step - loss: 0.4372 - recall: 0.0660 -
val_loss: 0.4352 - val_recall: 0.0736
Epoch 82/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4369 - recall: 0.0683 -
val_loss: 0.4349 - val_recall: 0.0736
Epoch 83/100
100/100 [==============================] - 1s 8ms/step - loss: 0.4365 - recall: 0.0713 -
val_loss: 0.4346 - val_recall: 0.0736
Epoch 84/100
100/100 [==============================] - 1s 9ms/step - loss: 0.4361 - recall: 0.0736 -
val_loss: 0.4343 - val_recall: 0.0767
```

```
Epoch 85/100
100/100 [==============================] - 1s 10ms/step - loss: 0.4358 - recall: 0.0759 - val_loss: 0.4340 - val_recall: 0.0798
Epoch 86/100
100/100 [==============================] - 1s 7ms/step - loss: 0.4354 - recall: 0.0782 - val_loss: 0.4338 - val_recall: 0.0798
Epoch 87/100
100/100 [==============================] - 1s 8ms/step - loss: 0.4351 - recall: 0.0805 - val_loss: 0.4335 - val_recall: 0.0798
Epoch 88/100
100/100 [==============================] - 1s 10ms/step - loss: 0.4348 - recall: 0.0836 - val_loss: 0.4332 - val_recall: 0.0798
Epoch 89/100
100/100 [==============================] - 2s 15ms/step - loss: 0.4344 - recall: 0.0882 - val_loss: 0.4330 - val_recall: 0.0828
Epoch 90/100
100/100 [==============================] - 1s 10ms/step - loss: 0.4341 - recall: 0.0913 - val_loss: 0.4327 - val_recall: 0.0859
Epoch 91/100
100/100 [==============================] - 1s 13ms/step - loss: 0.4338 - recall: 0.0928 - val_loss: 0.4325 - val_recall: 0.0920
Epoch 92/100
100/100 [==============================] - 2s 15ms/step - loss: 0.4335 - recall: 0.0943 - val_loss: 0.4323 - val_recall: 0.0951
Epoch 93/100
100/100 [==============================] - 1s 13ms/step - loss: 0.4332 - recall: 0.0959 - val_loss: 0.4320 - val_recall: 0.0982
Epoch 94/100
100/100 [==============================] - 1s 13ms/step - loss: 0.4329 - recall: 0.0989 - val_loss: 0.4318 - val_recall: 0.1012
Epoch 95/100
100/100 [==============================] - 2s 18ms/step - loss: 0.4326 - recall: 0.1020 - val_loss: 0.4316 - val_recall: 0.1074
Epoch 96/100
100/100 [==============================] - 2s 21ms/step - loss: 0.4323 - recall: 0.1028 - val_loss: 0.4314 - val_recall: 0.1074
Epoch 97/100
100/100 [==============================] - 2s 17ms/step - loss: 0.4321 - recall: 0.1058 - val_loss: 0.4312 - val_recall: 0.1104
Epoch 98/100
100/100 [==============================] - 1s 9ms/step - loss: 0.4318 - recall: 0.1120 - val_loss: 0.4310 - val_recall: 0.1104
Epoch 99/100
100/100 [==============================] - 1s 12ms/step - loss: 0.4315 - recall: 0.1127 - val_loss: 0.4308 - val_recall: 0.1104
Epoch 100/100
100/100 [==============================] - 1s 14ms/step - loss: 0.4313 - recall: 0.1158 - val_loss: 0.4306 - val_recall: 0.1104
```

In [55]:

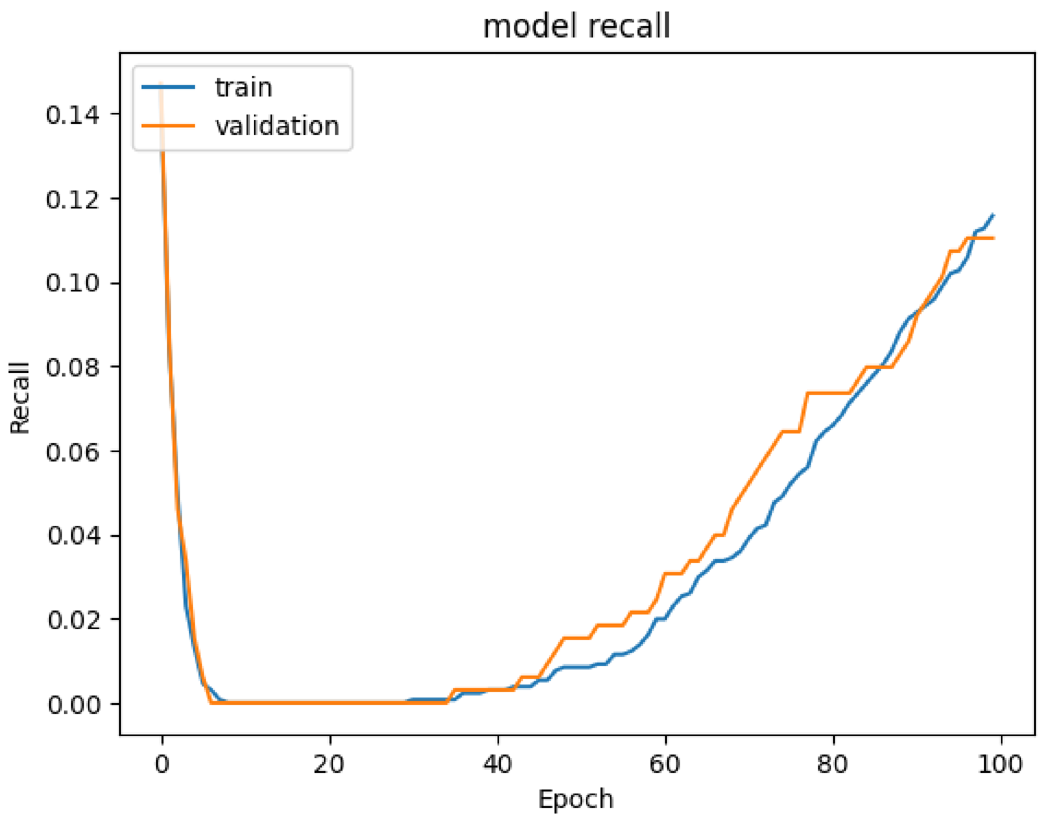
```
#Plotting Train Loss vs Validation Loss

plt.plot(model0_performace.history['loss'])
plt.plot(model0_performace.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





In [56]:

```
#Plotting Train recall vs Validation recall
plt.plot(model0_performace.history['recall'])
plt.plot(model0_performace.history['val_recall'])
plt.title('model recall')
plt.ylabel('Recall')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



The code provided involves using a trained neural network model to make predictions on training and validation data, then applying a threshold to convert these predictions into binary outcomes (0 or 1).

In [57]:

```
y_train_pred = model_0.predict(X_train)
y_train_pred = (y_train_pred > 0.5)
y_val_pred = model_0.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
```

```
200/200 [==============================] - 2s 6ms/step
50/50 [==============================] - 0s 7ms/step
```

LETS STORE THE RESULTS IN ORDER TO REVIEW THE OUTCOME LATER

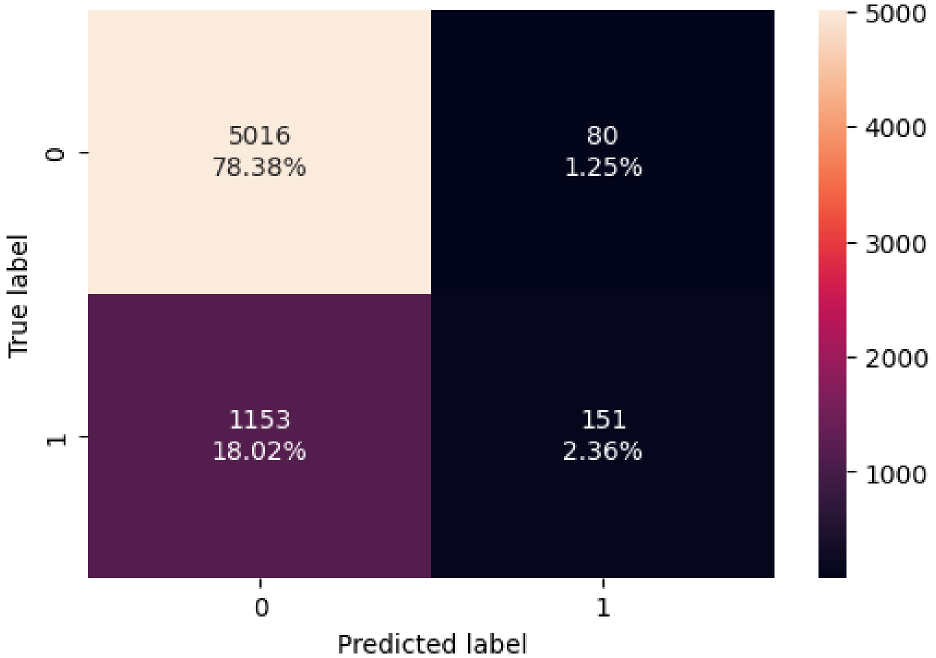
LETS STORE THE RESULTS IN ORDER TO REVIEW THE OUTCOME LATER.

In [58]:

```
model_name = "NN with SGD"

trainning_perf.loc[model_name] = recall_score(y_train,y_train_pred)
validation_perf.loc[model_name] = recall_score(y_val,y_val_pred)
```

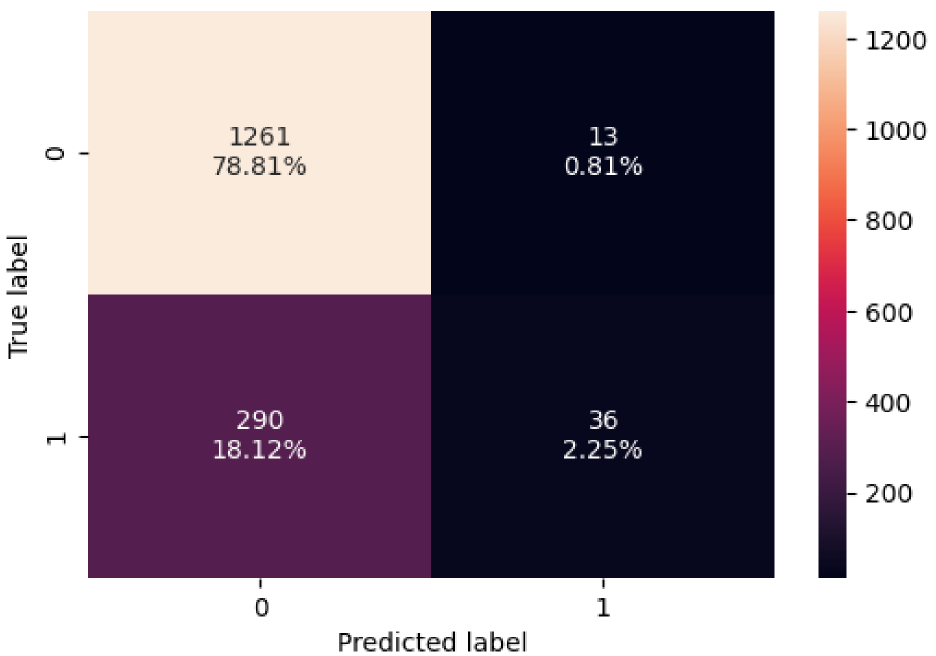
In [59]:

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



In [60]:

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



Model Performance Improvement

Neural Network with Adam Optimizer

In [61]:

```
backend.clear_session()
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

In [62]:

```
model_1 = Sequential()
model_1.add(Dense(64,activation='relu',input_dim = X_train.shape[1]))
model_1.add(Dense(32,activation='relu'))
model_1.add(Dense(1, activation = 'sigmoid'))
```

In [63]:

```
optimizer = tf.keras.optimizers.Adam()
metric = keras.metrics.Recall()
```

In [64]:

```
model_1.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [65]:

```
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 [66]:

```
ModelPerformace_1 = model_1.fit(
    X_train,y_train,
    batch_size=64,
    validation_data=(X_val,y_val),
    epochs=100,
    verbose=1
)
```

```
Epoch 1/100
100/100 [==============================] - 6s 18ms/step - loss: 0.4764 - recall: 0.0429 -
val_loss: 0.4331 - val_recall: 0.0982
Epoch 2/100
100/100 [==============================] - 1s 7ms/step - loss: 0.4237 - recall: 0.1994 -
val_loss: 0.4215 - val_recall: 0.1933
Epoch 3/100
100/100 [==============================] - 1s 9ms/step - loss: 0.4137 - recall: 0.2370 -
val_loss: 0.4167 - val_recall: 0.2730
Epoch 4/100
100/100 [==============================] - 1s 8ms/step - loss: 0.4068 - recall: 0.2937 -
val_loss: 0.4128 - val_recall: 0.3252
Epoch 5/100
100/100 [==============================] - 1s 12ms/step - loss: 0.3991 - recall: 0.2998 -
val_loss: 0.4107 - val_recall: 0.2454
Epoch 6/100
100/100 [==============================] - 1s 8ms/step - loss: 0.3937 - recall: 0.3321 -
val_loss: 0.4038 - val_recall: 0.3098
Epoch 7/100
```

```
100/100 [==============================] - 1s 10ms/step - loss: 0.3866 - recall: 0.3436 -
val_loss: 0.3984 - val_recall: 0.3221
Epoch 8/100
100/100 [==============================] - 1s 10ms/step - loss: 0.3800 - recall: 0.3727 -
val_loss: 0.4061 - val_recall: 0.2423
Epoch 9/100
100/100 [==============================] - 1s 11ms/step - loss: 0.3748 - recall: 0.3704 -
val_loss: 0.3838 - val_recall: 0.3804
Epoch 10/100
100/100 [==============================] - 1s 12ms/step - loss: 0.3690 - recall: 0.3857 -
val_loss: 0.3838 - val_recall: 0.4387
Epoch 11/100
100/100 [==============================] - 1s 11ms/step - loss: 0.3631 - recall: 0.4041 -
val_loss: 0.3862 - val_recall: 0.2791
Epoch 12/100
100/100 [==============================] - 2s 15ms/step - loss: 0.3587 - recall: 0.4256 -
val_loss: 0.3736 - val_recall: 0.4233
Epoch 13/100
100/100 [==============================] - 1s 10ms/step - loss: 0.3555 - recall: 0.4133 -
val_loss: 0.3723 - val_recall: 0.4387
Epoch 14/100
100/100 [==============================] - 1s 11ms/step - loss: 0.3491 - recall: 0.4310 -
val_loss: 0.3694 - val_recall: 0.4141
Epoch 15/100
100/100 [==============================] - 1s 10ms/step - loss: 0.3471 - recall: 0.4325 -
val_loss: 0.3672 - val_recall: 0.4417
Epoch 16/100
100/100 [==============================] - 1s 8ms/step - loss: 0.3429 - recall: 0.4494 -
val_loss: 0.3688 - val_recall: 0.3558
Epoch 17/100
100/100 [==============================] - 1s 14ms/step - loss: 0.3386 - recall: 0.4540 -
val_loss: 0.3622 - val_recall: 0.4049
Epoch 18/100
100/100 [==============================] - 1s 9ms/step - loss: 0.3368 - recall: 0.4509 -
val_loss: 0.3571 - val_recall: 0.3834
Epoch 19/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3323 - recall: 0.4678 -
val_loss: 0.3678 - val_recall: 0.3466
Epoch 20/100
100/100 [==============================] - 1s 9ms/step - loss: 0.3281 - recall: 0.4724 -
val_loss: 0.3605 - val_recall: 0.4417
Epoch 21/100
100/100 [==============================] - 1s 8ms/step - loss: 0.3264 - recall: 0.4793 -
val_loss: 0.3561 - val_recall: 0.4110
Epoch 22/100
100/100 [==============================] - 1s 9ms/step - loss: 0.3248 - recall: 0.4785 -
val_loss: 0.3547 - val_recall: 0.4387
Epoch 23/100
100/100 [==============================] - 1s 9ms/step - loss: 0.3212 - recall: 0.4816 -
val_loss: 0.3556 - val_recall: 0.4479
Epoch 24/100
100/100 [==============================] - 1s 8ms/step - loss: 0.3197 - recall: 0.4939 -
val_loss: 0.3578 - val_recall: 0.4110
Epoch 25/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3189 - recall: 0.4992 -
val_loss: 0.3528 - val_recall: 0.4509
Epoch 26/100
100/100 [==============================] - 1s 14ms/step - loss: 0.3154 - recall: 0.4939 -
val_loss: 0.3545 - val_recall: 0.4755
Epoch 27/100
100/100 [==============================] - 2s 19ms/step - loss: 0.3150 - recall: 0.5008 -
val_loss: 0.3565 - val_recall: 0.3988
Epoch 28/100
100/100 [==============================] - 2s 15ms/step - loss: 0.3119 - recall: 0.5092 -
val_loss: 0.3608 - val_recall: 0.4172
Epoch 29/100
100/100 [==============================] - 1s 12ms/step - loss: 0.3120 - recall: 0.5023 -
val_loss: 0.3570 - val_recall: 0.4847
Epoch 30/100
100/100 [==============================] - 1s 12ms/step - loss: 0.3101 - recall: 0.5153 -
val_loss: 0.3534 - val_recall: 0.4417
Epoch 31/100
```

```
100/100 [==============================] - 1s 7ms/step - loss: 0.3091 - recall: 0.5176 -
val_loss: 0.3564 - val_recall: 0.4202
Epoch 32/100
100/100 [==============================] - 1s 8ms/step - loss: 0.3065 - recall: 0.5092 -
val_loss: 0.3559 - val_recall: 0.4294
Epoch 33/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3062 - recall: 0.5161 -
val_loss: 0.3566 - val_recall: 0.4755
Epoch 34/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3079 - recall: 0.5222 -
val_loss: 0.3596 - val_recall: 0.3896
Epoch 35/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3040 - recall: 0.5184 -
val_loss: 0.3546 - val_recall: 0.4264
Epoch 36/100
100/100 [==============================] - 1s 8ms/step - loss: 0.3023 - recall: 0.5207 -
val_loss: 0.3565 - val_recall: 0.4325
Epoch 37/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3004 - recall: 0.5376 -
val_loss: 0.3584 - val_recall: 0.4356
Epoch 38/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3018 - recall: 0.5192 -
val_loss: 0.3658 - val_recall: 0.5307
Epoch 39/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3012 - recall: 0.5345 -
val_loss: 0.3563 - val_recall: 0.4356
Epoch 40/100
100/100 [==============================] - 1s 7ms/step - loss: 0.2992 - recall: 0.5245 -
val_loss: 0.3625 - val_recall: 0.4877
Epoch 41/100
100/100 [==============================] - 1s 6ms/step - loss: 0.2981 - recall: 0.5460 -
val_loss: 0.3618 - val_recall: 0.4110
Epoch 42/100
100/100 [==============================] - 1s 9ms/step - loss: 0.2988 - recall: 0.5238 -
val_loss: 0.3640 - val_recall: 0.4018
Epoch 43/100
100/100 [==============================] - 1s 8ms/step - loss: 0.2974 - recall: 0.5337 -
val_loss: 0.3608 - val_recall: 0.4571
Epoch 44/100
100/100 [==============================] - 1s 10ms/step - loss: 0.2964 - recall: 0.5291 -
val_loss: 0.3565 - val_recall: 0.4969
Epoch 45/100
100/100 [==============================] - 1s 12ms/step - loss: 0.2948 - recall: 0.5368 -
val_loss: 0.3697 - val_recall: 0.3896
Epoch 46/100
100/100 [==============================] - 1s 11ms/step - loss: 0.2957 - recall: 0.5261 -
val_loss: 0.3673 - val_recall: 0.3957
Epoch 47/100
100/100 [==============================] - 2s 16ms/step - loss: 0.2918 - recall: 0.5414 -
val_loss: 0.3612 - val_recall: 0.4847
Epoch 48/100
100/100 [==============================] - 1s 6ms/step - loss: 0.2916 - recall: 0.5537 -
val_loss: 0.3621 - val_recall: 0.4172
Epoch 49/100
100/100 [==============================] - 1s 6ms/step - loss: 0.2932 - recall: 0.5376 -
val_loss: 0.3622 - val_recall: 0.4387
Epoch 50/100
100/100 [==============================] - 1s 6ms/step - loss: 0.2912 - recall: 0.5506 -
val_loss: 0.3660 - val_recall: 0.4110
Epoch 51/100
100/100 [==============================] - 1s 6ms/step - loss: 0.2900 - recall: 0.5452 -
val_loss: 0.3613 - val_recall: 0.4233
Epoch 52/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2911 - recall: 0.5437 -
val_loss: 0.3734 - val_recall: 0.3650
Epoch 53/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2897 - recall: 0.5560 -
val_loss: 0.3715 - val_recall: 0.3834
Epoch 54/100
100/100 [==============================] - 1s 5ms/step - loss: 0.2883 - recall: 0.5483 -
val_loss: 0.3648 - val_recall: 0.4877
Epoch 55/100
```

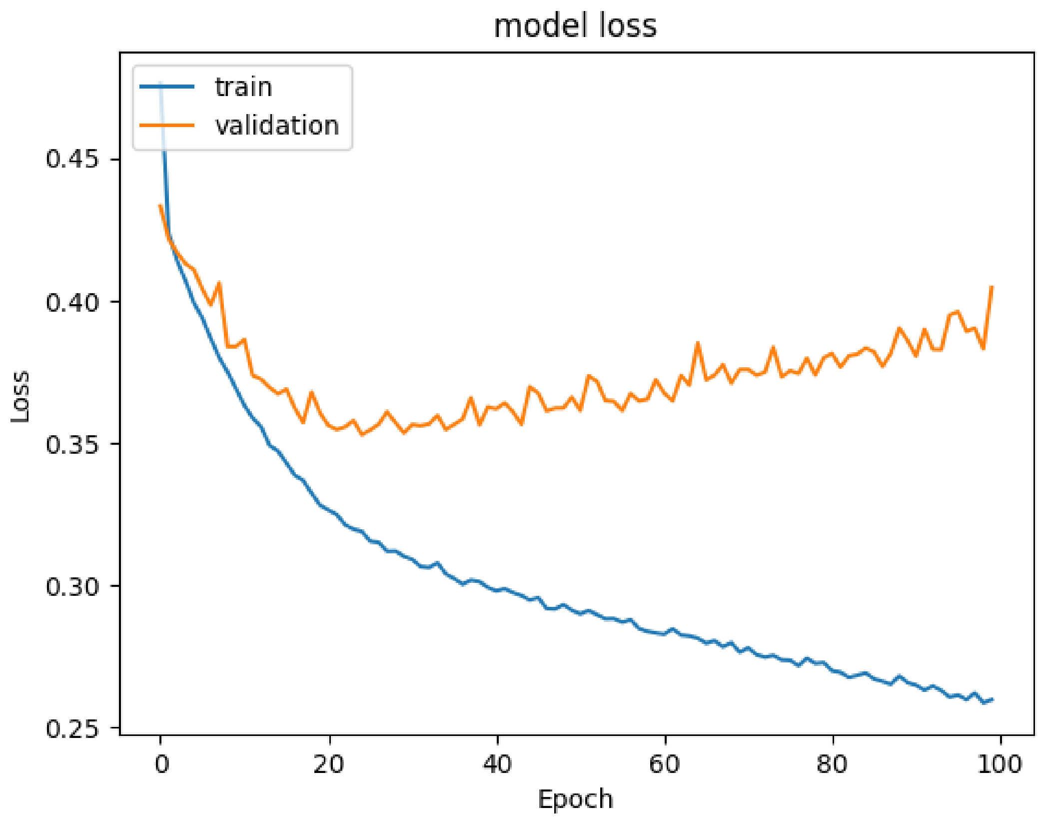
```
100/100 [==============================] - 0s 4ms/step - loss: 0.2883 - recall: 0.5598 -
val_loss: 0.3645 - val_recall: 0.4663
Epoch 56/100
100/100 [==============================] - 0s 3ms/step - loss: 0.2870 - recall: 0.5506 -
val_loss: 0.3613 - val_recall: 0.4816
Epoch 57/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2879 - recall: 0.5491 -
val_loss: 0.3673 - val_recall: 0.4448
Epoch 58/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2848 - recall: 0.5460 -
val_loss: 0.3647 - val_recall: 0.5092
Epoch 59/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2838 - recall: 0.5506 -
val_loss: 0.3653 - val_recall: 0.4479
Epoch 60/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2832 - recall: 0.5452 -
val_loss: 0.3721 - val_recall: 0.4387
Epoch 61/100
100/100 [==============================] - 0s 3ms/step - loss: 0.2828 - recall: 0.5629 -
val_loss: 0.3674 - val_recall: 0.4785
Epoch 62/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2847 - recall: 0.5460 -
val_loss: 0.3647 - val_recall: 0.4816
Epoch 63/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2825 - recall: 0.5567 -
val_loss: 0.3736 - val_recall: 0.4049
Epoch 64/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2822 - recall: 0.5544 -
val_loss: 0.3703 - val_recall: 0.5031
Epoch 65/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2814 - recall: 0.5621 -
val_loss: 0.3851 - val_recall: 0.3681
Epoch 66/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2797 - recall: 0.5644 -
val_loss: 0.3720 - val_recall: 0.4356
Epoch 67/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2805 - recall: 0.5560 -
val_loss: 0.3737 - val_recall: 0.4479
Epoch 68/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2784 - recall: 0.5713 -
val_loss: 0.3774 - val_recall: 0.4202
Epoch 69/100
100/100 [==============================] - 0s 3ms/step - loss: 0.2798 - recall: 0.5613 -
val_loss: 0.3710 - val_recall: 0.4509
Epoch 70/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2765 - recall: 0.5736 -
val_loss: 0.3757 - val_recall: 0.4325
Epoch 71/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2781 - recall: 0.5590 -
val_loss: 0.3758 - val_recall: 0.4571
Epoch 72/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2757 - recall: 0.5713 -
val_loss: 0.3737 - val_recall: 0.4785
Epoch 73/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2747 - recall: 0.5744 -
val_loss: 0.3748 - val_recall: 0.4632
Epoch 74/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2754 - recall: 0.5752 -
val_loss: 0.3835 - val_recall: 0.4110
Epoch 75/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2738 - recall: 0.5851 -
val_loss: 0.3732 - val_recall: 0.4540
Epoch 76/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2737 - recall: 0.5767 -
val_loss: 0.3753 - val_recall: 0.4294
Epoch 77/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2717 - recall: 0.5698 -
val_loss: 0.3743 - val_recall: 0.4509
Epoch 78/100
100/100 [==============================] - 1s 6ms/step - loss: 0.2744 - recall: 0.5767 -
val_loss: 0.3797 - val_recall: 0.4693
Epoch 79/100
```

```
100/100 [==============================] - 0s 5ms/step - loss: 0.2725 - recall: 0.5729 -
val_loss: 0.3738 - val_recall: 0.4509
Epoch 80/100
100/100 [==============================] - 1s 5ms/step - loss: 0.2729 - recall: 0.5874 -
val_loss: 0.3799 - val_recall: 0.4141
Epoch 81/100
100/100 [==============================] - 0s 5ms/step - loss: 0.2700 - recall: 0.5844 -
val_loss: 0.3813 - val_recall: 0.5153
Epoch 82/100
100/100 [==============================] - 0s 5ms/step - loss: 0.2694 - recall: 0.5836 -
val_loss: 0.3766 - val_recall: 0.4264
Epoch 83/100
100/100 [==============================] - 1s 5ms/step - loss: 0.2677 - recall: 0.5920 -
val_loss: 0.3806 - val_recall: 0.4601
Epoch 84/100
100/100 [==============================] - 1s 6ms/step - loss: 0.2684 - recall: 0.5913 -
val_loss: 0.3811 - val_recall: 0.5092
Epoch 85/100
100/100 [==============================] - 1s 7ms/step - loss: 0.2692 - recall: 0.5920 -
val_loss: 0.3832 - val_recall: 0.4141
Epoch 86/100
100/100 [==============================] - 1s 6ms/step - loss: 0.2671 - recall: 0.5867 -
val_loss: 0.3819 - val_recall: 0.5092
Epoch 87/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2663 - recall: 0.5851 -
val_loss: 0.3768 - val_recall: 0.4877
Epoch 88/100
100/100 [==============================] - 0s 3ms/step - loss: 0.2653 - recall: 0.5920 -
val_loss: 0.3814 - val_recall: 0.4939
Epoch 89/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2682 - recall: 0.5859 -
val_loss: 0.3903 - val_recall: 0.3988
Epoch 90/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2658 - recall: 0.5982 -
val_loss: 0.3858 - val_recall: 0.4049
Epoch 91/100
100/100 [==============================] - 0s 5ms/step - loss: 0.2649 - recall: 0.5867 -
val_loss: 0.3804 - val_recall: 0.4724
Epoch 92/100
100/100 [==============================] - 0s 3ms/step - loss: 0.2632 - recall: 0.6028 -
val_loss: 0.3898 - val_recall: 0.3957
Epoch 93/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2646 - recall: 0.5997 -
val_loss: 0.3828 - val_recall: 0.4632
Epoch 94/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2631 - recall: 0.6058 -
val_loss: 0.3827 - val_recall: 0.4264
Epoch 95/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2607 - recall: 0.6066 -
val_loss: 0.3948 - val_recall: 0.4080
Epoch 96/100
100/100 [==============================] - 0s 3ms/step - loss: 0.2614 - recall: 0.5897 -
val_loss: 0.3960 - val_recall: 0.4080
Epoch 97/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2598 - recall: 0.6081 -
val_loss: 0.3892 - val_recall: 0.4571
Epoch 98/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2622 - recall: 0.5928 -
val_loss: 0.3903 - val_recall: 0.4294
Epoch 99/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2588 - recall: 0.6020 -
val_loss: 0.3829 - val_recall: 0.5184
Epoch 100/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2598 - recall: 0.6074 -
val_loss: 0.4044 - val_recall: 0.3497
```

In [67]:

```
#Plotting Train Loss vs Validation Loss
plt.plot(ModelPerformace_1.history['loss'])
plt.plot(ModelPerformace_1.history['val_loss'])
```

```
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



In [68]:

```
y_train_pred = model_1.predict(X_train)
y_train_pred = (y_train_pred > 0.5)
y_val_pred = model_1.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
```

```
200/200 [==============================] - 1s 2ms/step
50/50 [==============================] - 0s 3ms/step
```

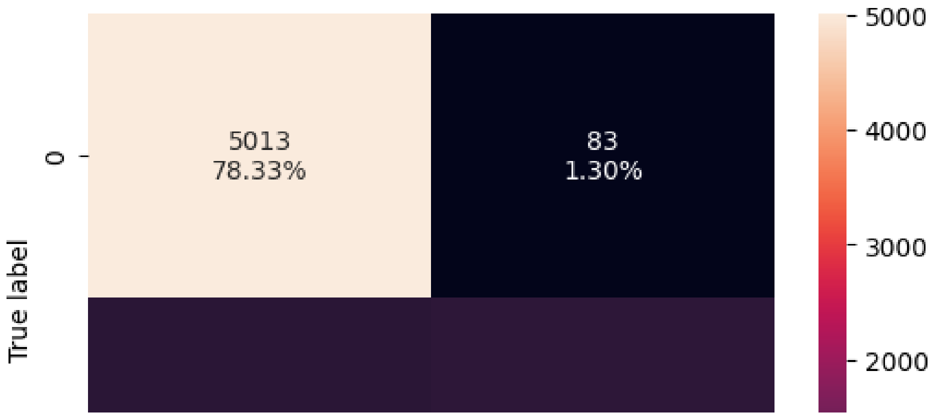
Lets store the results

In [69]:

```
model_name = "NN with Adam"

trainning_perf.loc[model_name] = recall_score(y_train,y_train_pred)
validation_perf.loc[model_name] = recall_score(y_val,y_val_pred)
```

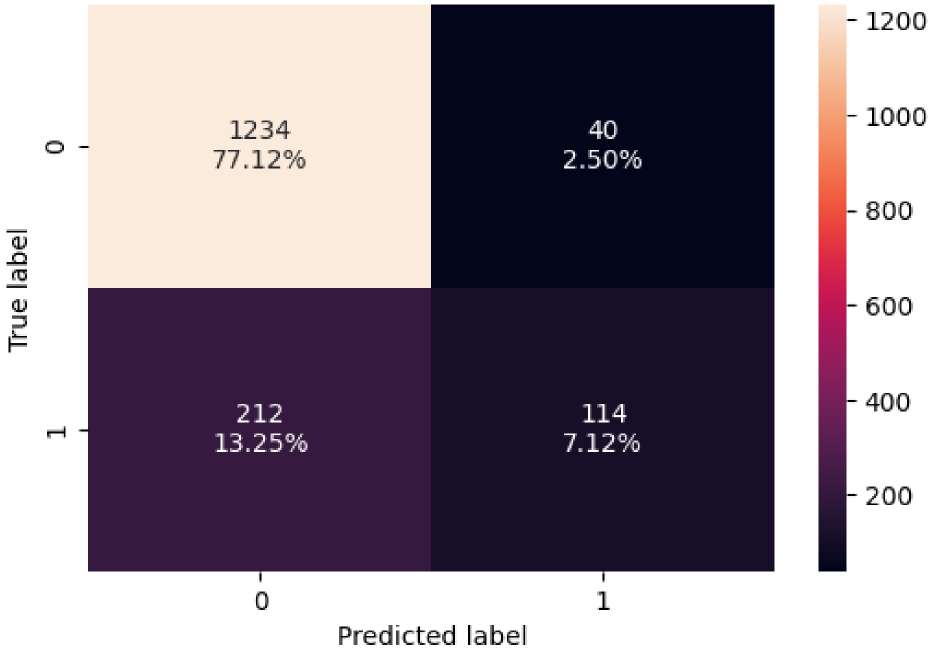
In [70]:

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



In [71]:

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



Neural Network with Adam Optimizer and Dropout

In [72]:

```
backend.clear_session()
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

In [73]:

```
#Initializing the neural network
model_2 = Sequential()
model_2.add(Dense(32,activation='relu',input_dim = X_train.shape[1]))
model_2.add(Dropout(0.2))
model_2.add(Dense(16,activation='relu'))
model_2.add(Dense(8,activation='relu'))
model_2.add(Dropout(0.1))
model_2.add(Dense(4,activation='relu'))
model_2.add(Dense(1, activation = 'sigmoid'))
```

In [74]:

```
optimizer = tf.keras.optimizers.Adam()
metric = keras.metrics.Recall()
```

In [75]:

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

In [76]:

```
model_2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 32)                384       
                                                                 
 dropout (Dropout)           (None, 32)                0         
                                                                 
 dense_1 (Dense)             (None, 16)                528       
                                                                 
 dense_2 (Dense)             (None, 8)                 136       
                                                                 
 dropout_1 (Dropout)         (None, 8)                 0         
                                                                 
 dense_3 (Dense)             (None, 4)                 36        
                                                                 
 dense_4 (Dense)             (None, 1)                 5         
                                                                 
=================================================================
Total params: 1089 (4.25 KB)
Trainable params: 1089 (4.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [77]:

```
#Fitting the ANN with batch_size = 32 and 100 epochs
ModelPerformace_2 = model_2.fit(
    X_train,y_train,
    batch_size=64,
    epochs=100,
    verbose=1,
    validation_data=(X_val,y_val)
)
```

```
Epoch 1/100
100/100 [==============================] - 2s 7ms/step - loss: 0.5592 - recall: 0.1273 - val_loss: 0.4697 - val_recall: 0.0000e+00
Epoch 2/100
100/100 [==============================] - 1s 7ms/step - loss: 0.4704 - recall: 0.0153 - val_loss: 0.4400 - val_recall: 0.0031
Epoch 3/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4551 - recall: 0.0199 - val_loss: 0.4334 - val_recall: 0.0092
Epoch 4/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4435 - recall: 0.0330 - val_loss: 0.4287 - val_recall: 0.0184
Epoch 5/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4425 - recall: 0.0590 - val_loss: 0.4269 - val_recall: 0.0276
Epoch 6/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4388 - recall: 0.1258 - val_loss: 0.4191 - val_recall: 0.0951
Epoch 7/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4292 - recall: 0.2032 - val_loss: 0.4130 - val_recall: 0.1840
Epoch 8/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4256 - recall: 0.2362 - val_loss: 0.4155 - val_recall: 0.1626
Epoch 9/100
100/100 [==============================] - 1s 6ms/step - loss: 0.4185 - recall: 0.2416 - val_loss: 0.4030 - val_recall: 0.2791
Epoch 10/100
100/100 [==============================] - 1s 5ms/step - loss: 0.4171 - recall: 0.3029 - val_loss: 0.3975 - val_recall: 0.3589
Epoch 11/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4119 - recall: 0.3198 - val_loss: 0.4052 - val_recall: 0.2025
Epoch 12/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4019 - recall: 0.3282 - val_loss: 0.3894 - val_recall: 0.3712
```

```
Epoch 13/100
100/100 [==============================] - 0s 4ms/step - loss: 0.4042 - recall: 0.3252 -
val_loss: 0.3895 - val_recall: 0.3221
Epoch 14/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3967 - recall: 0.3459 -
val_loss: 0.3890 - val_recall: 0.3344
Epoch 15/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3961 - recall: 0.3405 -
val_loss: 0.3803 - val_recall: 0.4049
Epoch 16/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3914 - recall: 0.3689 -
val_loss: 0.3798 - val_recall: 0.3681
Epoch 17/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3893 - recall: 0.3742 -
val_loss: 0.3725 - val_recall: 0.3773
Epoch 18/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3869 - recall: 0.3696 -
val_loss: 0.3709 - val_recall: 0.3681
Epoch 19/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3835 - recall: 0.3911 -
val_loss: 0.3751 - val_recall: 0.3528
Epoch 20/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3822 - recall: 0.3788 -
val_loss: 0.3683 - val_recall: 0.3558
Epoch 21/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3751 - recall: 0.4049 -
val_loss: 0.3661 - val_recall: 0.3742
Epoch 22/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3748 - recall: 0.3972 -
val_loss: 0.3652 - val_recall: 0.3589
Epoch 23/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3749 - recall: 0.3965 -
val_loss: 0.3631 - val_recall: 0.3436
Epoch 24/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3714 - recall: 0.3980 -
val_loss: 0.3576 - val_recall: 0.3681
Epoch 25/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3674 - recall: 0.3842 -
val_loss: 0.3571 - val_recall: 0.3926
Epoch 26/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3691 - recall: 0.4026 -
val_loss: 0.3548 - val_recall: 0.4080
Epoch 27/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3622 - recall: 0.4149 -
val_loss: 0.3585 - val_recall: 0.3650
Epoch 28/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3661 - recall: 0.4026 -
val_loss: 0.3593 - val_recall: 0.3681
Epoch 29/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3618 - recall: 0.4103 -
val_loss: 0.3514 - val_recall: 0.4080
Epoch 30/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3588 - recall: 0.4279 -
val_loss: 0.3511 - val_recall: 0.3896
Epoch 31/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3615 - recall: 0.4072 -
val_loss: 0.3490 - val_recall: 0.4018
Epoch 32/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3495 - recall: 0.4294 -
val_loss: 0.3500 - val_recall: 0.3926
Epoch 33/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3523 - recall: 0.4302 -
val_loss: 0.3483 - val_recall: 0.4325
Epoch 34/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3559 - recall: 0.4317 -
val_loss: 0.3513 - val_recall: 0.3957
Epoch 35/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3548 - recall: 0.4172 -
val_loss: 0.3502 - val_recall: 0.4049
Epoch 36/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3492 - recall: 0.4202 -
val_loss: 0.3496 - val_recall: 0.4356
```

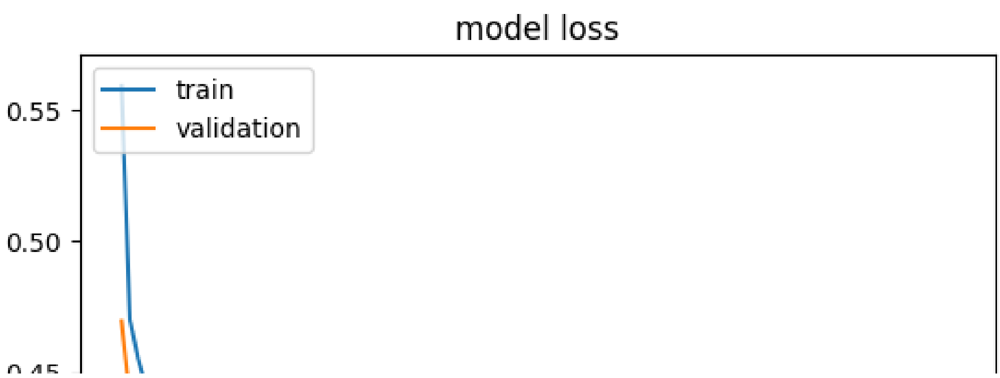
```
Epoch 37/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3483 - recall: 0.4525 -
val_loss: 0.3506 - val_recall: 0.4110
Epoch 38/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3485 - recall: 0.4233 -
val_loss: 0.3490 - val_recall: 0.4601
Epoch 39/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3474 - recall: 0.4494 -
val_loss: 0.3498 - val_recall: 0.4110
Epoch 40/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3502 - recall: 0.4348 -
val_loss: 0.3474 - val_recall: 0.4356
Epoch 41/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3450 - recall: 0.4540 -
val_loss: 0.3449 - val_recall: 0.4356
Epoch 42/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3480 - recall: 0.4363 -
val_loss: 0.3489 - val_recall: 0.4049
Epoch 43/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3475 - recall: 0.4294 -
val_loss: 0.3480 - val_recall: 0.4356
Epoch 44/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3415 - recall: 0.4471 -
val_loss: 0.3489 - val_recall: 0.4724
Epoch 45/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3441 - recall: 0.4709 -
val_loss: 0.3509 - val_recall: 0.3957
Epoch 46/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3488 - recall: 0.4440 -
val_loss: 0.3552 - val_recall: 0.3957
Epoch 47/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3406 - recall: 0.4440 -
val_loss: 0.3504 - val_recall: 0.4479
Epoch 48/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3423 - recall: 0.4701 -
val_loss: 0.3498 - val_recall: 0.4233
Epoch 49/100
100/100 [==============================] - 1s 8ms/step - loss: 0.3378 - recall: 0.4732 -
val_loss: 0.3503 - val_recall: 0.4540
Epoch 50/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3406 - recall: 0.4647 -
val_loss: 0.3541 - val_recall: 0.3742
Epoch 51/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3398 - recall: 0.4693 -
val_loss: 0.3509 - val_recall: 0.4264
Epoch 52/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3375 - recall: 0.4617 -
val_loss: 0.3518 - val_recall: 0.4294
Epoch 53/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3398 - recall: 0.4647 -
val_loss: 0.3562 - val_recall: 0.4018
Epoch 54/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3385 - recall: 0.4601 -
val_loss: 0.3526 - val_recall: 0.4325
Epoch 55/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3386 - recall: 0.4624 -
val_loss: 0.3526 - val_recall: 0.4294
Epoch 56/100
100/100 [==============================] - 0s 5ms/step - loss: 0.3336 - recall: 0.4785 -
val_loss: 0.3519 - val_recall: 0.4540
Epoch 57/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3329 - recall: 0.4877 -
val_loss: 0.3513 - val_recall: 0.4202
Epoch 58/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3354 - recall: 0.4816 -
val_loss: 0.3510 - val_recall: 0.4479
Epoch 59/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3342 - recall: 0.4893 -
val_loss: 0.3533 - val_recall: 0.4172
Epoch 60/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3334 - recall: 0.4831 -
val_loss: 0.3526 - val_recall: 0.4632
```

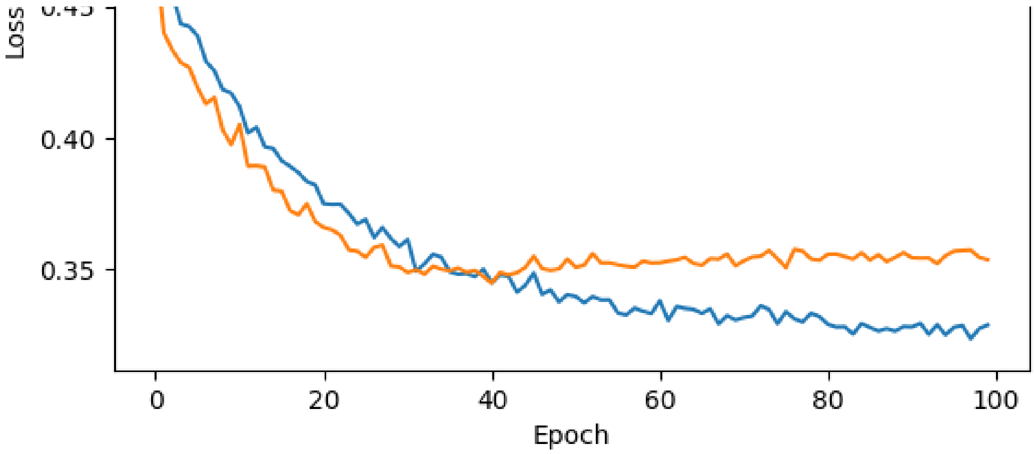
```
Epoch 61/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3382 - recall: 0.4739 -
val_loss: 0.3526 - val_recall: 0.4172
Epoch 62/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3308 - recall: 0.4647 -
val_loss: 0.3533 - val_recall: 0.4509
Epoch 63/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3360 - recall: 0.4755 -
val_loss: 0.3538 - val_recall: 0.4233
Epoch 64/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3353 - recall: 0.4762 -
val_loss: 0.3547 - val_recall: 0.4387
Epoch 65/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3350 - recall: 0.4816 -
val_loss: 0.3524 - val_recall: 0.4110
Epoch 66/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3334 - recall: 0.4601 -
val_loss: 0.3517 - val_recall: 0.4755
Epoch 67/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3351 - recall: 0.4762 -
val_loss: 0.3543 - val_recall: 0.4233
Epoch 68/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3294 - recall: 0.4908 -
val_loss: 0.3540 - val_recall: 0.4601
Epoch 69/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3326 - recall: 0.4762 -
val_loss: 0.3560 - val_recall: 0.4509
Epoch 70/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3309 - recall: 0.4900 -
val_loss: 0.3515 - val_recall: 0.4571
Epoch 71/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3319 - recall: 0.4655 -
val_loss: 0.3534 - val_recall: 0.4571
Epoch 72/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3324 - recall: 0.4762 -
val_loss: 0.3548 - val_recall: 0.4509
Epoch 73/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3363 - recall: 0.4770 -
val_loss: 0.3552 - val_recall: 0.4264
Epoch 74/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3349 - recall: 0.4785 -
val_loss: 0.3573 - val_recall: 0.4110
Epoch 75/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3295 - recall: 0.4755 -
val_loss: 0.3542 - val_recall: 0.4509
Epoch 76/100
100/100 [==============================] - 0s 5ms/step - loss: 0.3341 - recall: 0.4755 -
val_loss: 0.3508 - val_recall: 0.4724
Epoch 77/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3314 - recall: 0.4900 -
val_loss: 0.3577 - val_recall: 0.4294
Epoch 78/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3302 - recall: 0.4908 -
val_loss: 0.3571 - val_recall: 0.4908
Epoch 79/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3335 - recall: 0.4923 -
val_loss: 0.3541 - val_recall: 0.4571
Epoch 80/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3322 - recall: 0.4877 -
val_loss: 0.3536 - val_recall: 0.4571
Epoch 81/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3293 - recall: 0.4816 -
val_loss: 0.3559 - val_recall: 0.4877
Epoch 82/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3282 - recall: 0.4992 -
val_loss: 0.3560 - val_recall: 0.4356
Epoch 83/100
100/100 [==============================] - 0s 5ms/step - loss: 0.3283 - recall: 0.4908 -
val_loss: 0.3550 - val_recall: 0.4540
Epoch 84/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3255 - recall: 0.4977 -
val_loss: 0.3541 - val_recall: 0.4755
```

```
Epoch 85/100
100/100 [==============================] - 0s 5ms/step - loss: 0.3295 - recall: 0.4946 - val_loss: 0.3565 - val_recall: 0.4325
Epoch 86/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3280 - recall: 0.4992 - val_loss: 0.3537 - val_recall: 0.4632
Epoch 87/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3267 - recall: 0.4732 - val_loss: 0.3557 - val_recall: 0.4877
Epoch 88/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3276 - recall: 0.4931 - val_loss: 0.3531 - val_recall: 0.4479
Epoch 89/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3267 - recall: 0.4870 - val_loss: 0.3547 - val_recall: 0.4509
Epoch 90/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3284 - recall: 0.5008 - val_loss: 0.3565 - val_recall: 0.4325
Epoch 91/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3283 - recall: 0.4954 - val_loss: 0.3546 - val_recall: 0.4294
Epoch 92/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3296 - recall: 0.4724 - val_loss: 0.3544 - val_recall: 0.4294
Epoch 93/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3254 - recall: 0.4893 - val_loss: 0.3545 - val_recall: 0.4448
Epoch 94/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3291 - recall: 0.4939 - val_loss: 0.3523 - val_recall: 0.4387
Epoch 95/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3252 - recall: 0.4908 - val_loss: 0.3554 - val_recall: 0.4509
Epoch 96/100
100/100 [==============================] - 0s 4ms/step - loss: 0.3281 - recall: 0.4977 - val_loss: 0.3571 - val_recall: 0.4387
Epoch 97/100
100/100 [==============================] - 1s 5ms/step - loss: 0.3288 - recall: 0.4847 - val_loss: 0.3573 - val_recall: 0.4847
Epoch 98/100
100/100 [==============================] - 1s 7ms/step - loss: 0.3238 - recall: 0.5069 - val_loss: 0.3575 - val_recall: 0.4110
Epoch 99/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3276 - recall: 0.4908 - val_loss: 0.3547 - val_recall: 0.4264
Epoch 100/100
100/100 [==============================] - 1s 6ms/step - loss: 0.3290 - recall: 0.4939 - val_loss: 0.3539 - val_recall: 0.4663
```

In [78]:

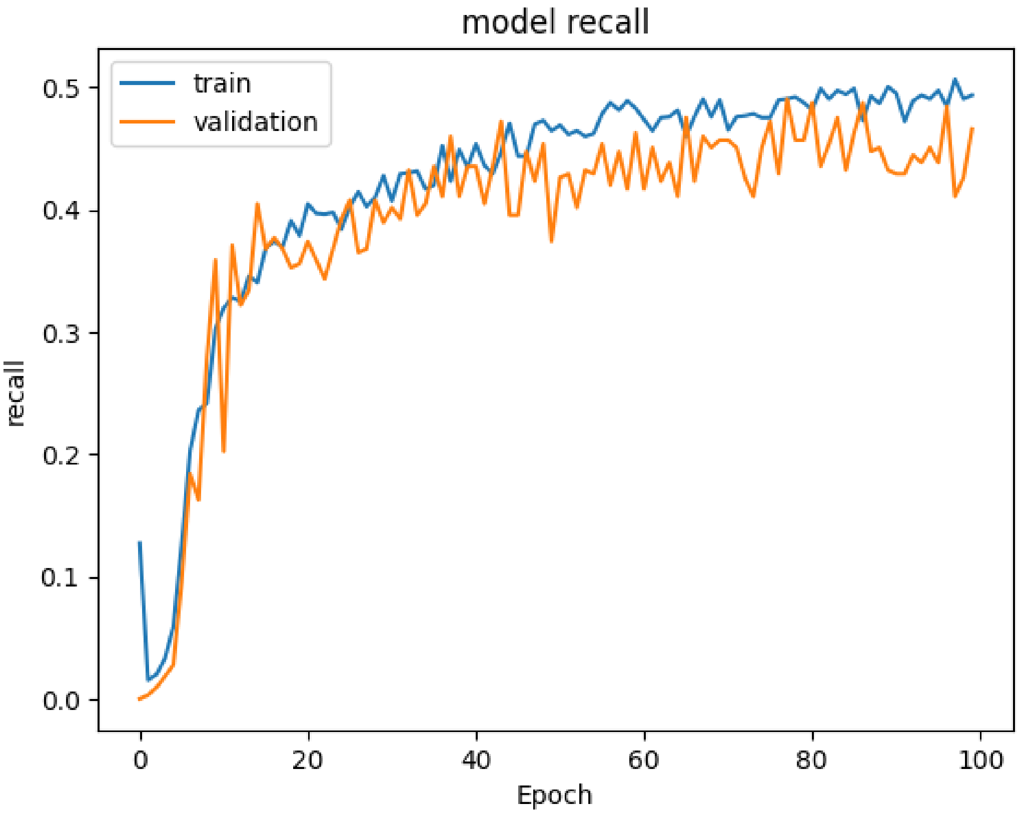
```
#Plotting Train Loss vs Validation Loss
plt.plot(ModelPerformace_2.history['loss'])
plt.plot(ModelPerformace_2.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





In [79]:

```
#Plotting Train recall vs Validation recall
plt.plot(ModelPerformace_2.history['recall'])
plt.plot(ModelPerformace_2.history['val_recall'])
plt.title('model recall')
plt.ylabel('recall')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



LETS STORE THE RESULTS

In [80]:

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

trainning_perf.loc[model_name] = recall_score(y_train,y_train_pred)
validation_perf.loc[model_name] = recall_score(y_val,y_val_pred)
```

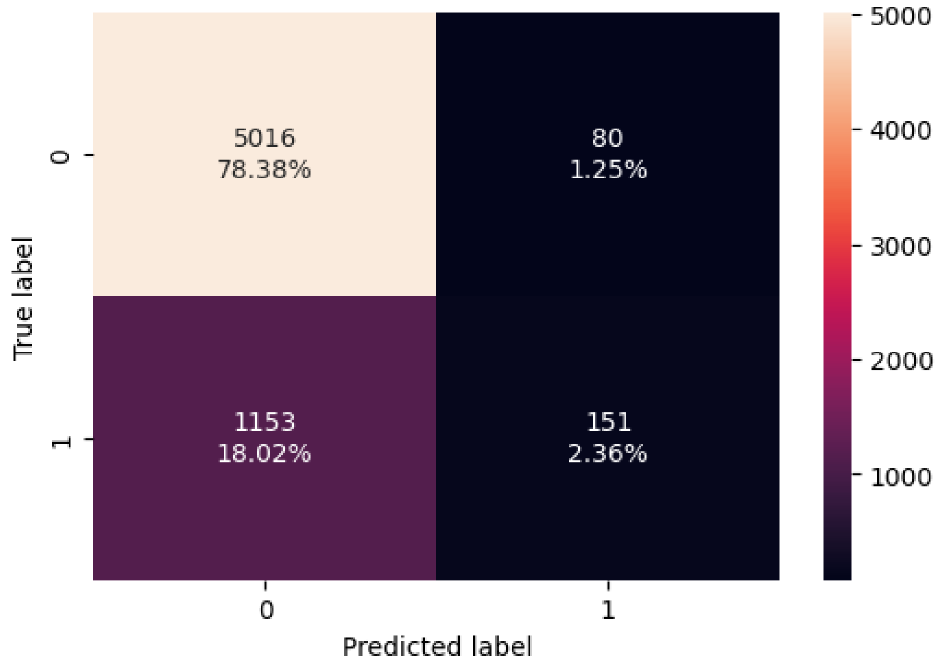
In [81]:

```
y_train_pred = model_0.predict(X_train)
y_train_pred = (y_train_pred > 0.5)
y_val_pred = model_0.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
```

```
200/200 [==============================] - 1s 3ms/step
50/50 [==============================] - 0s 3ms/step
```

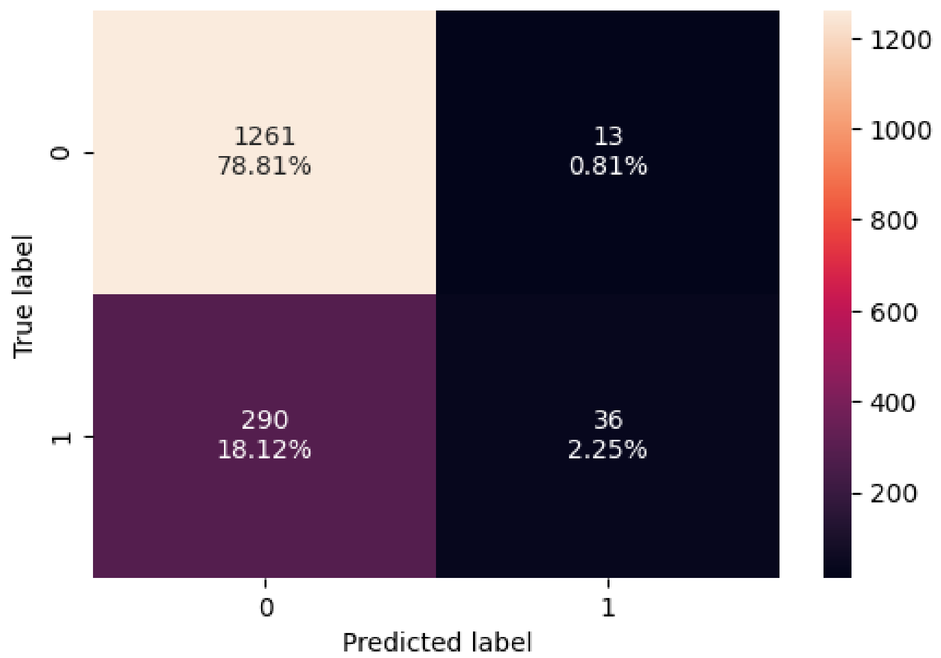
In [82]:

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



In [83]:

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



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

In [84]:

```
sm  = SMOTE(random_state=42)
#Complete the code to 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: (10192, 11)
After UpSampling, the shape of train_y: (10192,)
```

In [85]:

```
backend.clear_session()
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

In [86]:

```
#Initializing the model
model_3 = Sequential()
model_3.add(Dense(64,activation='relu',input_dim = X_train_smote.shape[1]))
model_3.add(Dense(32,activation='relu'))
model_3.add(Dense(16,activation='relu'))
model_3.add(Dense(1, activation = 'sigmoid'))
```

In [87]:

```
optimizer = tf.keras.optimizers.SGD(0.001)
metric = keras.metrics.Recall()
```

In [88]:

```
model_3.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [89]:

```
model_3.summary()
```

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

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 1)                 17

=================================================================
Total params: 3393 (13.25 KB)
Trainable params: 3393 (13.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [90]:

```
#Fitting the ANN
ModelPerformace_3 = model_3.fit(
    X_train_smote, y_train_smote,
    batch_size=64,
    epochs=100,
    verbose=1,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/100
160/160 [==============================] - 2s 5ms/step - loss: 0.6860 - recall: 0.6274 -
val_loss: 0.6898 - val_recall: 0.5859
Epoch 2/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6833 - recall: 0.6160 -
val_loss: 0.6842 - val_recall: 0.5675
Epoch 3/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6806 - recall: 0.5958 -
val_loss: 0.6799 - val_recall: 0.5736
Epoch 4/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6780 - recall: 0.5940 -
val_loss: 0.6755 - val_recall: 0.5644
```

```
Epoch 5/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6755 - recall: 0.5909 -
val_loss: 0.6714 - val_recall: 0.5552
Epoch 6/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6729 - recall: 0.5893 -
val_loss: 0.6677 - val_recall: 0.5583
Epoch 7/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6705 - recall: 0.5985 -
val_loss: 0.6640 - val_recall: 0.5613
Epoch 8/100
160/160 [==============================] - 1s 3ms/step - loss: 0.6680 - recall: 0.5995 -
val_loss: 0.6603 - val_recall: 0.5613
Epoch 9/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6655 - recall: 0.5991 -
val_loss: 0.6571 - val_recall: 0.5521
Epoch 10/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6630 - recall: 0.5995 -
val_loss: 0.6541 - val_recall: 0.5675
Epoch 11/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6605 - recall: 0.5993 -
val_loss: 0.6509 - val_recall: 0.5613
Epoch 12/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6580 - recall: 0.6026 -
val_loss: 0.6480 - val_recall: 0.5613
Epoch 13/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6554 - recall: 0.6052 -
val_loss: 0.6453 - val_recall: 0.5798
Epoch 14/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6529 - recall: 0.6069 -
val_loss: 0.6425 - val_recall: 0.5920
Epoch 15/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6502 - recall: 0.6136 -
val_loss: 0.6399 - val_recall: 0.6012
Epoch 16/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6475 - recall: 0.6181 -
val_loss: 0.6372 - val_recall: 0.6012
Epoch 17/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6448 - recall: 0.6221 -
val_loss: 0.6346 - val_recall: 0.6043
Epoch 18/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6420 - recall: 0.6264 -
val_loss: 0.6315 - val_recall: 0.6043
Epoch 19/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6392 - recall: 0.6307 -
val_loss: 0.6290 - val_recall: 0.6012
Epoch 20/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6363 - recall: 0.6372 -
val_loss: 0.6256 - val_recall: 0.6012
Epoch 21/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6334 - recall: 0.6391 -
val_loss: 0.6229 - val_recall: 0.6074
Epoch 22/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6305 - recall: 0.6423 -
val_loss: 0.6202 - val_recall: 0.6074
Epoch 23/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6276 - recall: 0.6480 -
val_loss: 0.6173 - val_recall: 0.6104
Epoch 24/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6247 - recall: 0.6495 -
val_loss: 0.6147 - val_recall: 0.6227
Epoch 25/100
160/160 [==============================] - 1s 3ms/step - loss: 0.6219 - recall: 0.6546 -
val_loss: 0.6119 - val_recall: 0.6196
Epoch 26/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6191 - recall: 0.6574 -
val_loss: 0.6094 - val_recall: 0.6227
Epoch 27/100
160/160 [==============================] - 1s 5ms/step - loss: 0.6162 - recall: 0.6593 -
val_loss: 0.6065 - val_recall: 0.6227
Epoch 28/100
160/160 [==============================] - 1s 6ms/step - loss: 0.6135 - recall: 0.6609 -
val_loss: 0.6042 - val_recall: 0.6258
```

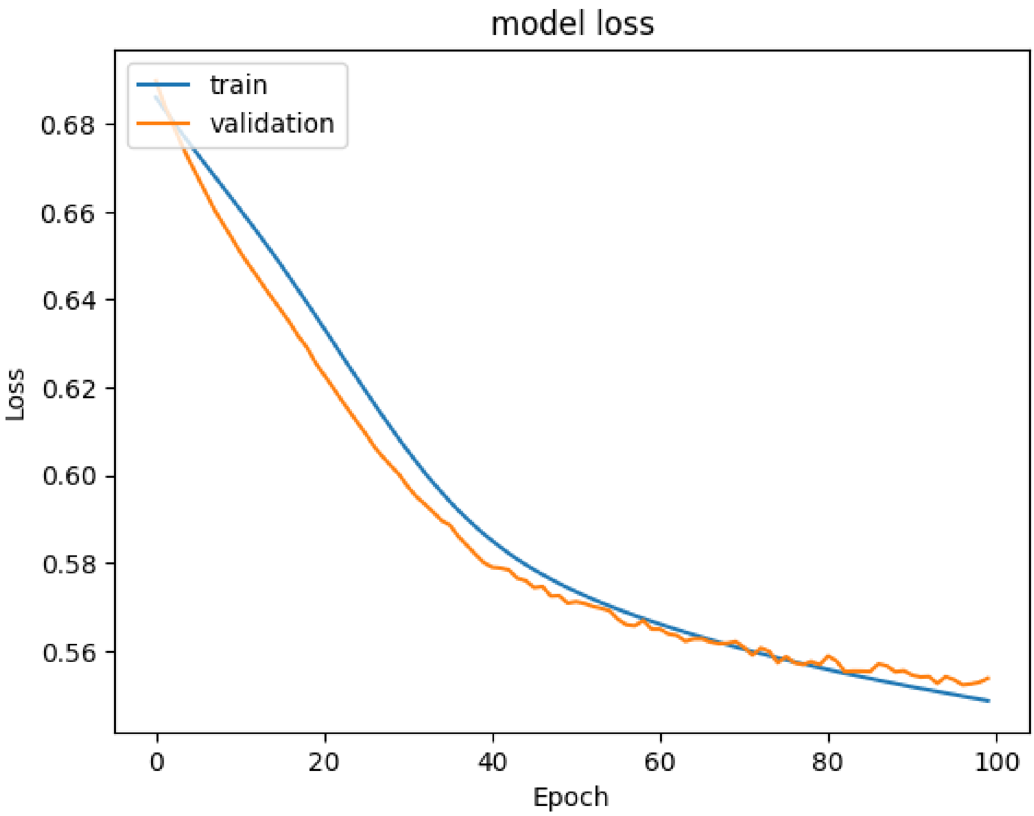
```
Epoch 29/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6107 - recall: 0.6607 -
val_loss: 0.6021 - val_recall: 0.6288
Epoch 30/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6081 - recall: 0.6642 -
val_loss: 0.6001 - val_recall: 0.6350
Epoch 31/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6055 - recall: 0.6674 -
val_loss: 0.5973 - val_recall: 0.6350
Epoch 32/100
160/160 [==============================] - 1s 8ms/step - loss: 0.6030 - recall: 0.6701 -
val_loss: 0.5951 - val_recall: 0.6411
Epoch 33/100
160/160 [==============================] - 1s 3ms/step - loss: 0.6006 - recall: 0.6688 -
val_loss: 0.5935 - val_recall: 0.6411
Epoch 34/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5983 - recall: 0.6746 -
val_loss: 0.5916 - val_recall: 0.6442
Epoch 35/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5961 - recall: 0.6764 -
val_loss: 0.5897 - val_recall: 0.6442
Epoch 36/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5940 - recall: 0.6776 -
val_loss: 0.5887 - val_recall: 0.6472
Epoch 37/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5920 - recall: 0.6829 -
val_loss: 0.5861 - val_recall: 0.6472
Epoch 38/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5901 - recall: 0.6817 -
val_loss: 0.5841 - val_recall: 0.6503
Epoch 39/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5884 - recall: 0.6837 -
val_loss: 0.5821 - val_recall: 0.6472
Epoch 40/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5867 - recall: 0.6876 -
val_loss: 0.5802 - val_recall: 0.6503
Epoch 41/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5851 - recall: 0.6852 -
val_loss: 0.5791 - val_recall: 0.6534
Epoch 42/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5837 - recall: 0.6862 -
val_loss: 0.5789 - val_recall: 0.6534
Epoch 43/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5823 - recall: 0.6874 -
val_loss: 0.5785 - val_recall: 0.6595
Epoch 44/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5809 - recall: 0.6911 -
val_loss: 0.5766 - val_recall: 0.6534
Epoch 45/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5797 - recall: 0.6907 -
val_loss: 0.5760 - val_recall: 0.6534
Epoch 46/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5785 - recall: 0.6925 -
val_loss: 0.5745 - val_recall: 0.6534
Epoch 47/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5774 - recall: 0.6927 -
val_loss: 0.5747 - val_recall: 0.6564
Epoch 48/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5764 - recall: 0.6966 -
val_loss: 0.5725 - val_recall: 0.6503
Epoch 49/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5754 - recall: 0.6933 -
val_loss: 0.5726 - val_recall: 0.6503
Epoch 50/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5744 - recall: 0.6968 -
val_loss: 0.5709 - val_recall: 0.6503
Epoch 51/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5735 - recall: 0.6947 -
val_loss: 0.5712 - val_recall: 0.6534
Epoch 52/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5727 - recall: 0.6972 -
val_loss: 0.5709 - val_recall: 0.6595
```

```
Epoch 53/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5718 - recall: 0.6968 -
val_loss: 0.5702 - val_recall: 0.6595
Epoch 54/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5710 - recall: 0.6978 -
val_loss: 0.5697 - val_recall: 0.6595
Epoch 55/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5703 - recall: 0.6980 -
val_loss: 0.5692 - val_recall: 0.6595
Epoch 56/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5695 - recall: 0.7021 -
val_loss: 0.5672 - val_recall: 0.6564
Epoch 57/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5688 - recall: 0.7009 -
val_loss: 0.5660 - val_recall: 0.6595
Epoch 58/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5681 - recall: 0.7000 -
val_loss: 0.5658 - val_recall: 0.6595
Epoch 59/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5674 - recall: 0.7004 -
val_loss: 0.5670 - val_recall: 0.6626
Epoch 60/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5668 - recall: 0.7047 -
val_loss: 0.5650 - val_recall: 0.6595
Epoch 61/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5662 - recall: 0.7031 -
val_loss: 0.5650 - val_recall: 0.6626
Epoch 62/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5655 - recall: 0.7047 -
val_loss: 0.5639 - val_recall: 0.6626
Epoch 63/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5649 - recall: 0.7035 -
val_loss: 0.5636 - val_recall: 0.6626
Epoch 64/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5643 - recall: 0.7049 -
val_loss: 0.5623 - val_recall: 0.6595
Epoch 65/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5637 - recall: 0.7041 -
val_loss: 0.5628 - val_recall: 0.6687
Epoch 66/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5632 - recall: 0.7051 -
val_loss: 0.5628 - val_recall: 0.6687
Epoch 67/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5626 - recall: 0.7062 -
val_loss: 0.5621 - val_recall: 0.6687
Epoch 68/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5620 - recall: 0.7076 -
val_loss: 0.5617 - val_recall: 0.6656
Epoch 69/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5615 - recall: 0.7074 -
val_loss: 0.5618 - val_recall: 0.6656
Epoch 70/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5610 - recall: 0.7078 -
val_loss: 0.5622 - val_recall: 0.6656
Epoch 71/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5605 - recall: 0.7102 -
val_loss: 0.5609 - val_recall: 0.6656
Epoch 72/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5599 - recall: 0.7104 -
val_loss: 0.5591 - val_recall: 0.6626
Epoch 73/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5595 - recall: 0.7084 -
val_loss: 0.5607 - val_recall: 0.6626
Epoch 74/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5590 - recall: 0.7131 -
val_loss: 0.5600 - val_recall: 0.6626
Epoch 75/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5585 - recall: 0.7157 -
val_loss: 0.5574 - val_recall: 0.6626
Epoch 76/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5580 - recall: 0.7125 -
val_loss: 0.5588 - val_recall: 0.6656
```

```
Epoch 77/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5576 - recall: 0.7151 -
val_loss: 0.5574 - val_recall: 0.6656
Epoch 78/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5571 - recall: 0.7141 -
val_loss: 0.5569 - val_recall: 0.6656
Epoch 79/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5567 - recall: 0.7117 -
val_loss: 0.5577 - val_recall: 0.6656
Epoch 80/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5562 - recall: 0.7139 -
val_loss: 0.5569 - val_recall: 0.6656
Epoch 81/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5558 - recall: 0.7127 -
val_loss: 0.5589 - val_recall: 0.6687
Epoch 82/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5554 - recall: 0.7178 -
val_loss: 0.5577 - val_recall: 0.6687
Epoch 83/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5550 - recall: 0.7188 -
val_loss: 0.5554 - val_recall: 0.6687
Epoch 84/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5546 - recall: 0.7157 -
val_loss: 0.5555 - val_recall: 0.6687
Epoch 85/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5542 - recall: 0.7170 -
val_loss: 0.5554 - val_recall: 0.6687
Epoch 86/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5538 - recall: 0.7192 -
val_loss: 0.5554 - val_recall: 0.6687
Epoch 87/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5534 - recall: 0.7174 -
val_loss: 0.5571 - val_recall: 0.6718
Epoch 88/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5530 - recall: 0.7202 -
val_loss: 0.5566 - val_recall: 0.6718
Epoch 89/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5526 - recall: 0.7229 -
val_loss: 0.5553 - val_recall: 0.6718
Epoch 90/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5523 - recall: 0.7202 -
val_loss: 0.5555 - val_recall: 0.6718
Epoch 91/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5519 - recall: 0.7215 -
val_loss: 0.5546 - val_recall: 0.6718
Epoch 92/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5515 - recall: 0.7231 -
val_loss: 0.5541 - val_recall: 0.6718
Epoch 93/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5512 - recall: 0.7215 -
val_loss: 0.5542 - val_recall: 0.6718
Epoch 94/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5508 - recall: 0.7247 -
val_loss: 0.5527 - val_recall: 0.6718
Epoch 95/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5505 - recall: 0.7223 -
val_loss: 0.5542 - val_recall: 0.6748
Epoch 96/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5501 - recall: 0.7259 -
val_loss: 0.5535 - val_recall: 0.6779
Epoch 97/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5498 - recall: 0.7263 -
val_loss: 0.5523 - val_recall: 0.6779
Epoch 98/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5494 - recall: 0.7253 -
val_loss: 0.5525 - val_recall: 0.6810
Epoch 99/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5491 - recall: 0.7255 -
val_loss: 0.5529 - val_recall: 0.6840
Epoch 100/100
160/160 [==============================] - 1s 3ms/step - loss: 0.5487 - recall: 0.7251 -
val_loss: 0.5538 - val_recall: 0.6840
```

In [91]:

```
#Plotting Train Loss vs Validation Loss
plt.plot(ModelPerformace_3.history['loss'])
plt.plot(ModelPerformace_3.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



In [92]:

```
y_train_pred = model_3.predict(X_train_smote)
y_train_pred = (y_train_pred > 0.5)
y_val_pred = model_3.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
```

```
319/319 [==============================] - 1s 3ms/step
50/50 [==============================] - 0s 2ms/step
```

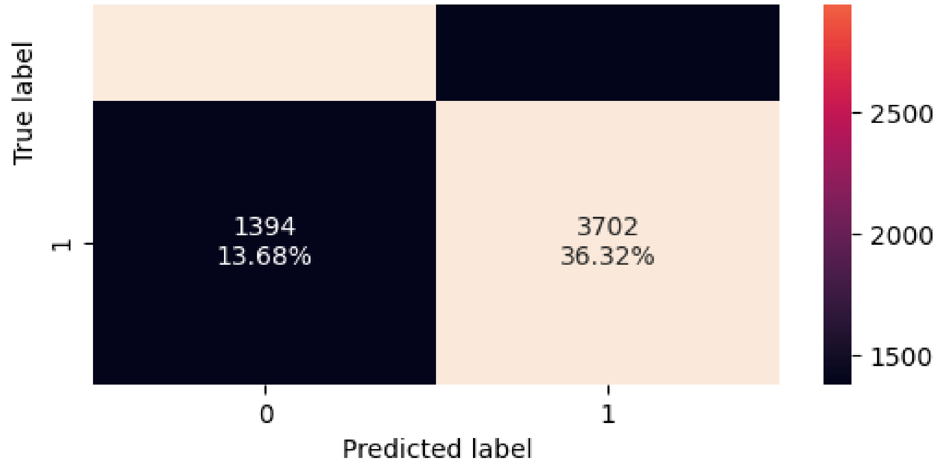
Lets Store the results

In [93]:

```
model_name = "NN with SMOTE and SGD"

trainning_perf.loc[model_name] = recall_score(y_train_smote,y_train_pred)
validation_perf.loc[model_name] = recall_score(y_val,y_val_pred)
```

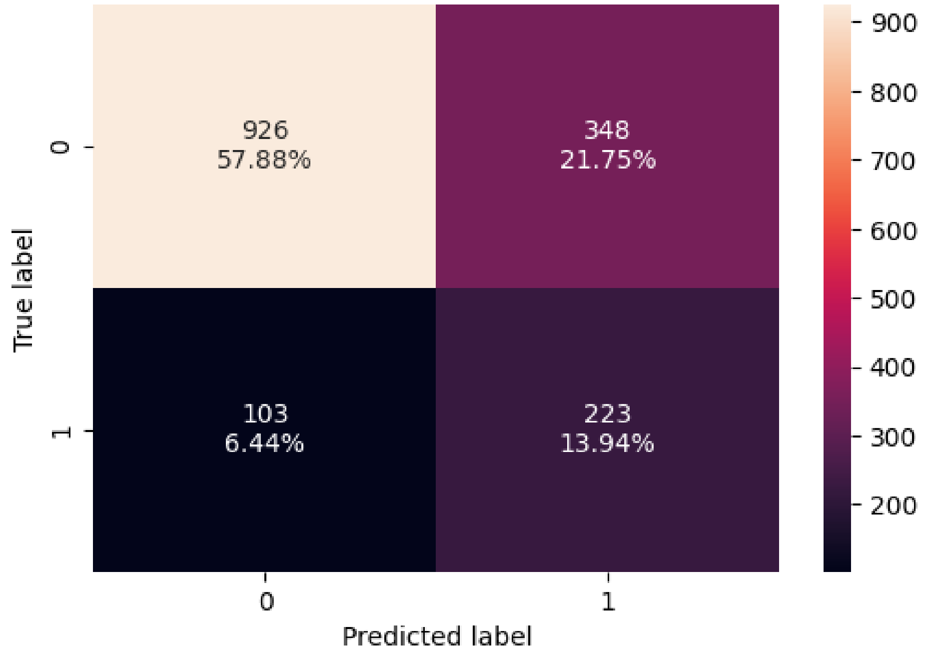
In [94]:

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



In [95]:

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



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

In [96]:

```
backend.clear_session()
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

In [97]:

```
model_4 = Sequential()
model_4.add(Dense(64,activation='relu',input_dim = X_train_smote.shape[1]))
model_4.add(Dense(32,activation='relu'))
model_4.add(Dense(16,activation='relu'))
model_4.add(Dense(1, activation = 'sigmoid'))
```

In [98]:

```
model_4.summary()
```

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

```
 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 1)                 17

=================================================================
Total params: 3393 (13.25 KB)
Trainable params: 3393 (13.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [99]:

```
optimizer = tf.keras.optimizers.Adam()
metric = keras.metrics.Recall()
```

In [100]:

```
model_4.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [101]:

```
#Fitting the ANN

ModelPerformance4 = model_4.fit(
    X_train_smote,y_train_smote,
    batch_size=64,
    epochs=100,
    verbose=1,
    validation_data = (X_val,y_val)
)
```

```
Epoch 1/100
160/160 [==============================] - 3s 9ms/step - loss: 0.5883 - recall: 0.6919 -
val_loss: 0.5587 - val_recall: 0.6871
Epoch 2/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5396 - recall: 0.7162 -
val_loss: 0.5383 - val_recall: 0.6810
Epoch 3/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5173 - recall: 0.7347 -
val_loss: 0.5390 - val_recall: 0.6902
Epoch 4/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4936 - recall: 0.7537 -
val_loss: 0.4800 - val_recall: 0.6350
Epoch 5/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4773 - recall: 0.7647 -
val_loss: 0.4728 - val_recall: 0.6687
Epoch 6/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4606 - recall: 0.7783 -
val_loss: 0.5082 - val_recall: 0.7393
Epoch 7/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4467 - recall: 0.7922 -
val_loss: 0.4957 - val_recall: 0.7393
Epoch 8/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4295 - recall: 0.8059 -
val_loss: 0.4806 - val_recall: 0.7362
Epoch 9/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4155 - recall: 0.8138 -
val_loss: 0.4700 - val_recall: 0.6994
Epoch 10/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4027 - recall: 0.8228 -
val_loss: 0.5256 - val_recall: 0.7730
Epoch 11/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3948 - recall: 0.8271 -
val_loss: 0.4342 - val_recall: 0.6779
Epoch 12/100
160/160 [==============================] - 1s 5ms/step - loss: 0.3851 - recall: 0.8269 -
val_loss: 0.4730 - val_recall: 0.7178
Epoch 13/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3788 - recall: 0.8295 -
```

```
val_loss: 0.4699 - val_recall: 0.7025
Epoch 14/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3738 - recall: 0.8299 -
val_loss: 0.4747 - val_recall: 0.7239
Epoch 15/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3663 - recall: 0.8416 -
val_loss: 0.4848 - val_recall: 0.7209
Epoch 16/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3594 - recall: 0.8450 -
val_loss: 0.4724 - val_recall: 0.7055
Epoch 17/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3535 - recall: 0.8483 -
val_loss: 0.4479 - val_recall: 0.6595
Epoch 18/100
160/160 [==============================] - 2s 10ms/step - loss: 0.3458 - recall: 0.8524 -
val_loss: 0.4442 - val_recall: 0.6871
Epoch 19/100
160/160 [==============================] - 1s 5ms/step - loss: 0.3393 - recall: 0.8560 -
val_loss: 0.4990 - val_recall: 0.7362
Epoch 20/100
160/160 [==============================] - 1s 6ms/step - loss: 0.3343 - recall: 0.8638 -
val_loss: 0.4676 - val_recall: 0.7025
Epoch 21/100
160/160 [==============================] - 1s 6ms/step - loss: 0.3310 - recall: 0.8630 -
val_loss: 0.4618 - val_recall: 0.6902
Epoch 22/100
160/160 [==============================] - 1s 6ms/step - loss: 0.3252 - recall: 0.8693 -
val_loss: 0.4237 - val_recall: 0.5859
Epoch 23/100
160/160 [==============================] - 1s 5ms/step - loss: 0.3222 - recall: 0.8744 -
val_loss: 0.4502 - val_recall: 0.6534
Epoch 24/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3207 - recall: 0.8666 -
val_loss: 0.5034 - val_recall: 0.7178
Epoch 25/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3163 - recall: 0.8752 -
val_loss: 0.4793 - val_recall: 0.6810
Epoch 26/100
160/160 [==============================] - 1s 5ms/step - loss: 0.3080 - recall: 0.8776 -
val_loss: 0.4462 - val_recall: 0.6104
Epoch 27/100
160/160 [==============================] - 1s 6ms/step - loss: 0.3064 - recall: 0.8744 -
val_loss: 0.4667 - val_recall: 0.6748
Epoch 28/100
160/160 [==============================] - 1s 5ms/step - loss: 0.3003 - recall: 0.8828 -
val_loss: 0.5025 - val_recall: 0.7055
Epoch 29/100
160/160 [==============================] - 1s 4ms/step - loss: 0.3029 - recall: 0.8832 -
val_loss: 0.4626 - val_recall: 0.6288
Epoch 30/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2940 - recall: 0.8842 -
val_loss: 0.4716 - val_recall: 0.6288
Epoch 31/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2929 - recall: 0.8893 -
val_loss: 0.4534 - val_recall: 0.6012
Epoch 32/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2891 - recall: 0.8889 -
val_loss: 0.4609 - val_recall: 0.6380
Epoch 33/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2814 - recall: 0.8958 -
val_loss: 0.4666 - val_recall: 0.6166
Epoch 34/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2790 - recall: 0.8913 -
val_loss: 0.4857 - val_recall: 0.6104
Epoch 35/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2798 - recall: 0.8907 -
val_loss: 0.4966 - val_recall: 0.6687
Epoch 36/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2789 - recall: 0.8942 -
val_loss: 0.4773 - val_recall: 0.6012
Epoch 37/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2736 - recall: 0.8958 -
```

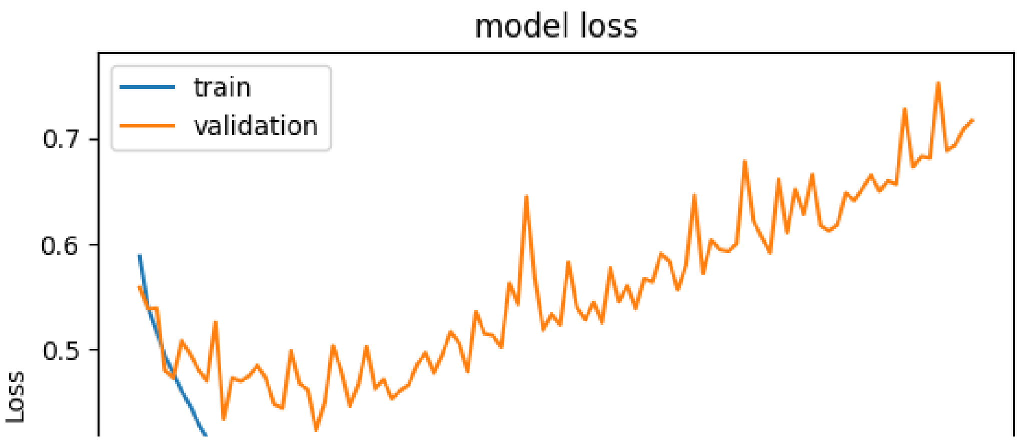
```
val_loss: 0.4955 - val_recall: 0.6595
Epoch 38/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2655 - recall: 0.9027 -
val_loss: 0.5167 - val_recall: 0.6534
Epoch 39/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2676 - recall: 0.9003 -
val_loss: 0.5058 - val_recall: 0.6564
Epoch 40/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2599 - recall: 0.9064 -
val_loss: 0.4789 - val_recall: 0.5276
Epoch 41/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2639 - recall: 0.8978 -
val_loss: 0.5355 - val_recall: 0.6534
Epoch 42/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2605 - recall: 0.9052 -
val_loss: 0.5151 - val_recall: 0.6534
Epoch 43/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2592 - recall: 0.9036 -
val_loss: 0.5133 - val_recall: 0.6288
Epoch 44/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2535 - recall: 0.9109 -
val_loss: 0.5022 - val_recall: 0.6319
Epoch 45/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2520 - recall: 0.9070 -
val_loss: 0.5626 - val_recall: 0.6626
Epoch 46/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2474 - recall: 0.9099 -
val_loss: 0.5424 - val_recall: 0.6779
Epoch 47/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2448 - recall: 0.9097 -
val_loss: 0.6447 - val_recall: 0.7178
Epoch 48/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2482 - recall: 0.9107 -
val_loss: 0.5672 - val_recall: 0.6534
Epoch 49/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2464 - recall: 0.9101 -
val_loss: 0.5186 - val_recall: 0.6043
Epoch 50/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2390 - recall: 0.9164 -
val_loss: 0.5337 - val_recall: 0.5982
Epoch 51/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2405 - recall: 0.9080 -
val_loss: 0.5233 - val_recall: 0.5951
Epoch 52/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2402 - recall: 0.9076 -
val_loss: 0.5826 - val_recall: 0.6718
Epoch 53/100
160/160 [==============================] - 1s 8ms/step - loss: 0.2313 - recall: 0.9184 -
val_loss: 0.5400 - val_recall: 0.6012
Epoch 54/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2311 - recall: 0.9188 -
val_loss: 0.5281 - val_recall: 0.5767
Epoch 55/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2319 - recall: 0.9176 -
val_loss: 0.5446 - val_recall: 0.6074
Epoch 56/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2253 - recall: 0.9215 -
val_loss: 0.5253 - val_recall: 0.5215
Epoch 57/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2263 - recall: 0.9166 -
val_loss: 0.5772 - val_recall: 0.6227
Epoch 58/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2266 - recall: 0.9184 -
val_loss: 0.5454 - val_recall: 0.5460
Epoch 59/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2230 - recall: 0.9225 -
val_loss: 0.5604 - val_recall: 0.6074
Epoch 60/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2187 - recall: 0.9299 -
val_loss: 0.5383 - val_recall: 0.4939
Epoch 61/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2200 - recall: 0.9235 -
```

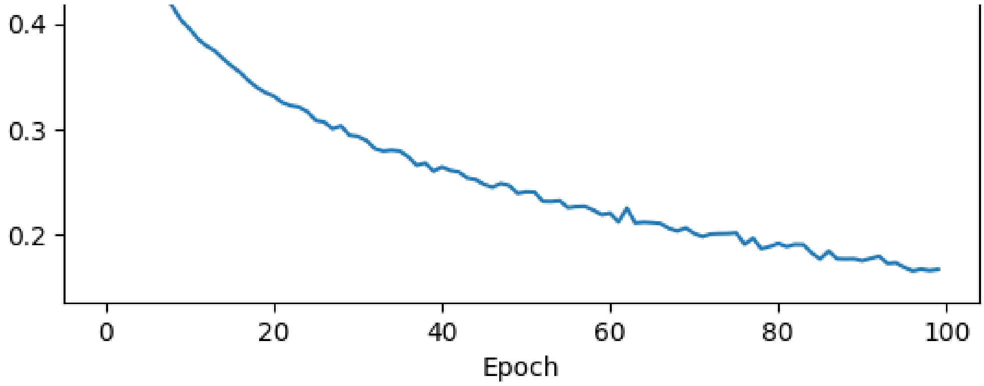
```
val_loss: 0.5669 - val_recall: 0.5920
Epoch 62/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2119 - recall: 0.9274 -
val_loss: 0.5641 - val_recall: 0.5828
Epoch 63/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2250 - recall: 0.9186 -
val_loss: 0.5907 - val_recall: 0.6442
Epoch 64/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2108 - recall: 0.9299 -
val_loss: 0.5833 - val_recall: 0.5552
Epoch 65/100
160/160 [==============================] - 1s 6ms/step - loss: 0.2116 - recall: 0.9264 -
val_loss: 0.5565 - val_recall: 0.5736
Epoch 66/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2111 - recall: 0.9290 -
val_loss: 0.5799 - val_recall: 0.5429
Epoch 67/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2103 - recall: 0.9272 -
val_loss: 0.6463 - val_recall: 0.6380
Epoch 68/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2057 - recall: 0.9264 -
val_loss: 0.5720 - val_recall: 0.5276
Epoch 69/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2031 - recall: 0.9317 -
val_loss: 0.6033 - val_recall: 0.5859
Epoch 70/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2061 - recall: 0.9292 -
val_loss: 0.5947 - val_recall: 0.5920
Epoch 71/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2008 - recall: 0.9311 -
val_loss: 0.5929 - val_recall: 0.5828
Epoch 72/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1978 - recall: 0.9341 -
val_loss: 0.6000 - val_recall: 0.4663
Epoch 73/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2003 - recall: 0.9258 -
val_loss: 0.6780 - val_recall: 0.6442
Epoch 74/100
160/160 [==============================] - 1s 5ms/step - loss: 0.2006 - recall: 0.9325 -
val_loss: 0.6217 - val_recall: 0.5736
Epoch 75/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2008 - recall: 0.9299 -
val_loss: 0.6060 - val_recall: 0.4571
Epoch 76/100
160/160 [==============================] - 1s 4ms/step - loss: 0.2014 - recall: 0.9297 -
val_loss: 0.5916 - val_recall: 0.5337
Epoch 77/100
160/160 [==============================] - 1s 5ms/step - loss: 0.1906 - recall: 0.9347 -
val_loss: 0.6612 - val_recall: 0.5460
Epoch 78/100
160/160 [==============================] - 1s 6ms/step - loss: 0.1963 - recall: 0.9337 -
val_loss: 0.6106 - val_recall: 0.5460
Epoch 79/100
160/160 [==============================] - 1s 6ms/step - loss: 0.1864 - recall: 0.9376 -
val_loss: 0.6515 - val_recall: 0.6350
Epoch 80/100
160/160 [==============================] - 1s 6ms/step - loss: 0.1885 - recall: 0.9380 -
val_loss: 0.6280 - val_recall: 0.5123
Epoch 81/100
160/160 [==============================] - 1s 7ms/step - loss: 0.1915 - recall: 0.9319 -
val_loss: 0.6655 - val_recall: 0.5736
Epoch 82/100
160/160 [==============================] - 1s 6ms/step - loss: 0.1885 - recall: 0.9366 -
val_loss: 0.6173 - val_recall: 0.5583
Epoch 83/100
160/160 [==============================] - 1s 7ms/step - loss: 0.1905 - recall: 0.9374 -
val_loss: 0.6121 - val_recall: 0.5460
Epoch 84/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1903 - recall: 0.9339 -
val_loss: 0.6184 - val_recall: 0.5245
Epoch 85/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1823 - recall: 0.9405 -
```

```
val_loss: 0.6482 - val_recall: 0.5337
Epoch 86/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1766 - recall: 0.9435 -
val_loss: 0.6409 - val_recall: 0.5736
Epoch 87/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1842 - recall: 0.9341 -
val_loss: 0.6527 - val_recall: 0.5644
Epoch 88/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1768 - recall: 0.9427 -
val_loss: 0.6648 - val_recall: 0.5951
Epoch 89/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1766 - recall: 0.9431 -
val_loss: 0.6498 - val_recall: 0.5307
Epoch 90/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1768 - recall: 0.9411 -
val_loss: 0.6597 - val_recall: 0.5736
Epoch 91/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1750 - recall: 0.9419 -
val_loss: 0.6563 - val_recall: 0.5337
Epoch 92/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1771 - recall: 0.9398 -
val_loss: 0.7276 - val_recall: 0.6380
Epoch 93/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1792 - recall: 0.9398 -
val_loss: 0.6727 - val_recall: 0.5307
Epoch 94/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1724 - recall: 0.9435 -
val_loss: 0.6830 - val_recall: 0.6380
Epoch 95/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1730 - recall: 0.9443 -
val_loss: 0.6814 - val_recall: 0.5920
Epoch 96/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1687 - recall: 0.9433 -
val_loss: 0.7521 - val_recall: 0.6411
Epoch 97/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1653 - recall: 0.9445 -
val_loss: 0.6881 - val_recall: 0.5644
Epoch 98/100
160/160 [==============================] - 1s 4ms/step - loss: 0.1671 - recall: 0.9454 -
val_loss: 0.6936 - val_recall: 0.5368
Epoch 99/100
160/160 [==============================] - 1s 6ms/step - loss: 0.1657 - recall: 0.9462 -
val_loss: 0.7087 - val_recall: 0.5491
Epoch 100/100
160/160 [==============================] - 1s 6ms/step - loss: 0.1668 - recall: 0.9447 -
val_loss: 0.7167 - val_recall: 0.5920
```

In [102]:

```
#Plotting Train Loss vs Validation Loss
plt.plot(ModelPerformance4.history['loss'])
plt.plot(ModelPerformance4.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





In [103]:

```
y_train_pred = model_4.predict(X_train_smote)
y_train_pred = (y_train_pred > 0.5)
y_val_pred = model_4.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
```

```
319/319 [==============================] - 1s 3ms/step
50/50 [==============================] - 0s 4ms/step
```

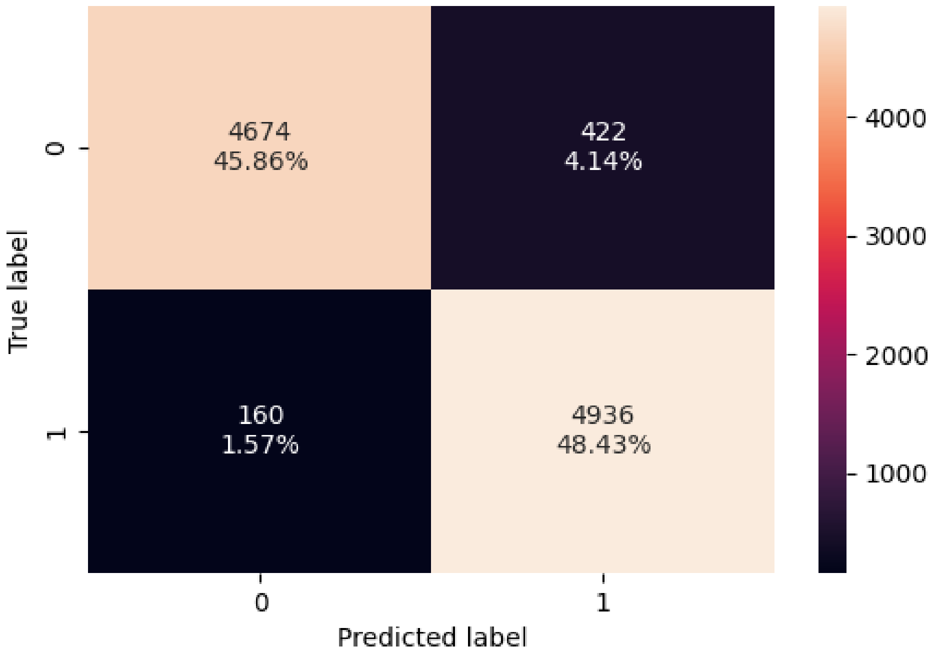
Lets store the results

In [104]:

```
model_name = "NN with SMOTE & Adam"

trainning_perf.loc[model_name] = recall_score(y_train_smote,y_train_pred)
validation_perf.loc[model_name] = recall_score(y_val,y_val_pred)
```

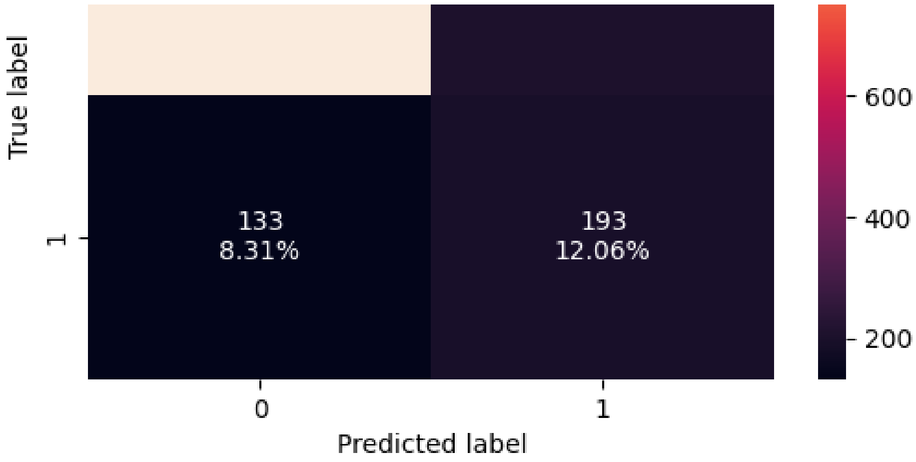
In [105]:

```
make_confusion_matrix(y_train_smote, y_train_pred)
```



In [106]:

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



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

In [107]:

```
backend.clear_session()
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

In [108]:

```
#Initializing the model
model_5 = Sequential()
model_5.add(Dense(64,activation='relu',input_dim = X_train_smote.shape[1]))
model_5.add(Dropout(0.5))
model_5.add(Dense(32,activation='relu'))
model_5.add(Dropout(0.5))
model_5.add(Dense(8,activation='relu'))
model_5.add(Dense(1, activation = 'sigmoid'))
```

In [109]:

```
optimizer = tf.keras.optimizers.Adam()
metric = keras.metrics.Recall()
```

In [110]:

```
model_5.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [111]:

```
model_5.summary()
```

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

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 8)                 264

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 3121 (12.19 KB)
Trainable params: 3121 (12.19 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
Non-trainable params: 0 (0.00 Byte)
```

In [112]:

```
ModelPerformace_5 = model_5.fit(
    X_train_smote, y_train_smote,
    batch_size=64,
    epochs=100,
    verbose=1,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/100
160/160 [==============================] - 2s 6ms/step - loss: 0.6930 - recall: 0.8350 -
val_loss: 0.6607 - val_recall: 0.5000
Epoch 2/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6388 - recall: 0.6436 -
val_loss: 0.5620 - val_recall: 0.5000
Epoch 3/100
160/160 [==============================] - 1s 4ms/step - loss: 0.6039 - recall: 0.6115 -
val_loss: 0.5412 - val_recall: 0.6350
Epoch 4/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5806 - recall: 0.6837 -
val_loss: 0.5232 - val_recall: 0.6350
Epoch 5/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5790 - recall: 0.6917 -
val_loss: 0.5197 - val_recall: 0.6411
Epoch 6/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5691 - recall: 0.6896 -
val_loss: 0.5361 - val_recall: 0.6595
Epoch 7/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5638 - recall: 0.7029 -
val_loss: 0.5007 - val_recall: 0.6319
Epoch 8/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5539 - recall: 0.7104 -
val_loss: 0.5243 - val_recall: 0.6687
Epoch 9/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5499 - recall: 0.7288 -
val_loss: 0.5136 - val_recall: 0.6626
Epoch 10/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5459 - recall: 0.7268 -
val_loss: 0.5159 - val_recall: 0.6656
Epoch 11/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5378 - recall: 0.7429 -
val_loss: 0.5246 - val_recall: 0.6748
Epoch 12/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5336 - recall: 0.7280 -
val_loss: 0.5171 - val_recall: 0.6626
Epoch 13/100
160/160 [==============================] - 1s 4ms/step - loss: 0.5271 - recall: 0.7266 -
val_loss: 0.5085 - val_recall: 0.6564
Epoch 14/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5246 - recall: 0.7363 -
val_loss: 0.4966 - val_recall: 0.6656
Epoch 15/100
160/160 [==============================] - 1s 7ms/step - loss: 0.5177 - recall: 0.7425 -
val_loss: 0.4924 - val_recall: 0.6626
Epoch 16/100
160/160 [==============================] - 1s 7ms/step - loss: 0.5090 - recall: 0.7498 -
val_loss: 0.4911 - val_recall: 0.6871
Epoch 17/100
160/160 [==============================] - 1s 8ms/step - loss: 0.5095 - recall: 0.7516 -
val_loss: 0.4902 - val_recall: 0.6933
Epoch 18/100
160/160 [==============================] - 1s 6ms/step - loss: 0.5058 - recall: 0.7535 -
val_loss: 0.4803 - val_recall: 0.6963
Epoch 19/100
160/160 [==============================] - 1s 5ms/step - loss: 0.5037 - recall: 0.7490 -
val_loss: 0.4921 - val_recall: 0.7055
Epoch 20/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4959 - recall: 0.7649 -
val_loss: 0.4661 - val_recall: 0.6994
```

```
val_loss: 0.4661 - val_recall: 0.6994
Epoch 21/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4935 - recall: 0.7673 -
val_loss: 0.4777 - val_recall: 0.7331
Epoch 22/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4903 - recall: 0.7714 -
val_loss: 0.4627 - val_recall: 0.7086
Epoch 23/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4846 - recall: 0.7700 -
val_loss: 0.4811 - val_recall: 0.7454
Epoch 24/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4808 - recall: 0.7796 -
val_loss: 0.4467 - val_recall: 0.7025
Epoch 25/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4755 - recall: 0.7710 -
val_loss: 0.4559 - val_recall: 0.7423
Epoch 26/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4743 - recall: 0.7810 -
val_loss: 0.4556 - val_recall: 0.7239
Epoch 27/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4720 - recall: 0.7832 -
val_loss: 0.4470 - val_recall: 0.7423
Epoch 28/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4658 - recall: 0.7912 -
val_loss: 0.4595 - val_recall: 0.7485
Epoch 29/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4600 - recall: 0.7788 -
val_loss: 0.4640 - val_recall: 0.7270
Epoch 30/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4619 - recall: 0.7851 -
val_loss: 0.4520 - val_recall: 0.7638
Epoch 31/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4569 - recall: 0.7920 -
val_loss: 0.4571 - val_recall: 0.7577
Epoch 32/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4536 - recall: 0.7940 -
val_loss: 0.4277 - val_recall: 0.7178
Epoch 33/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4519 - recall: 0.8059 -
val_loss: 0.4485 - val_recall: 0.7699
Epoch 34/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4473 - recall: 0.8067 -
val_loss: 0.4329 - val_recall: 0.7423
Epoch 35/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4468 - recall: 0.8024 -
val_loss: 0.4493 - val_recall: 0.7699
Epoch 36/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4440 - recall: 0.8030 -
val_loss: 0.4669 - val_recall: 0.7761
Epoch 37/100
160/160 [==============================] - 1s 7ms/step - loss: 0.4470 - recall: 0.7998 -
val_loss: 0.4369 - val_recall: 0.7454
Epoch 38/100
160/160 [==============================] - 1s 7ms/step - loss: 0.4484 - recall: 0.8032 -
val_loss: 0.4399 - val_recall: 0.7515
Epoch 39/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4387 - recall: 0.8091 -
val_loss: 0.4432 - val_recall: 0.7362
Epoch 40/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4425 - recall: 0.8042 -
val_loss: 0.4257 - val_recall: 0.7209
Epoch 41/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4443 - recall: 0.8034 -
val_loss: 0.4397 - val_recall: 0.7301
Epoch 42/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4426 - recall: 0.8010 -
val_loss: 0.4363 - val_recall: 0.7362
Epoch 43/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4355 - recall: 0.8102 -
val_loss: 0.4469 - val_recall: 0.7485
Epoch 44/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4375 - recall: 0.8134 -
val_loss: 0.4452 - val_recall: 0.7239
```

```
val_loss: 0.4452 - val_recall: 0.7239
Epoch 45/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4352 - recall: 0.8047 -
val_loss: 0.4388 - val_recall: 0.7362
Epoch 46/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4369 - recall: 0.8071 -
val_loss: 0.4486 - val_recall: 0.7669
Epoch 47/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4357 - recall: 0.8057 -
val_loss: 0.4403 - val_recall: 0.7638
Epoch 48/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4303 - recall: 0.8203 -
val_loss: 0.4381 - val_recall: 0.7178
Epoch 49/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4392 - recall: 0.8099 -
val_loss: 0.4351 - val_recall: 0.7423
Epoch 50/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4342 - recall: 0.8138 -
val_loss: 0.4452 - val_recall: 0.7699
Epoch 51/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4307 - recall: 0.8171 -
val_loss: 0.4434 - val_recall: 0.7577
Epoch 52/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4292 - recall: 0.8203 -
val_loss: 0.4314 - val_recall: 0.7331
Epoch 53/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4283 - recall: 0.8206 -
val_loss: 0.4615 - val_recall: 0.7730
Epoch 54/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4323 - recall: 0.8159 -
val_loss: 0.4442 - val_recall: 0.7485
Epoch 55/100
160/160 [==============================] - 1s 7ms/step - loss: 0.4315 - recall: 0.8122 -
val_loss: 0.4427 - val_recall: 0.7577
Epoch 56/100
160/160 [==============================] - 1s 7ms/step - loss: 0.4271 - recall: 0.8214 -
val_loss: 0.4306 - val_recall: 0.7331
Epoch 57/100
160/160 [==============================] - 1s 7ms/step - loss: 0.4274 - recall: 0.8246 -
val_loss: 0.4344 - val_recall: 0.7270
Epoch 58/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4307 - recall: 0.8193 -
val_loss: 0.4435 - val_recall: 0.7638
Epoch 59/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4296 - recall: 0.8269 -
val_loss: 0.4546 - val_recall: 0.7883
Epoch 60/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4237 - recall: 0.8314 -
val_loss: 0.4261 - val_recall: 0.7331
Epoch 61/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4301 - recall: 0.8183 -
val_loss: 0.4390 - val_recall: 0.7362
Epoch 62/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4301 - recall: 0.8222 -
val_loss: 0.4285 - val_recall: 0.7423
Epoch 63/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4284 - recall: 0.8226 -
val_loss: 0.4323 - val_recall: 0.7270
Epoch 64/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4272 - recall: 0.8279 -
val_loss: 0.4261 - val_recall: 0.7209
Epoch 65/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4249 - recall: 0.8224 -
val_loss: 0.4523 - val_recall: 0.7577
Epoch 66/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4295 - recall: 0.8261 -
val_loss: 0.4369 - val_recall: 0.7362
Epoch 67/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4215 - recall: 0.8342 -
val_loss: 0.4316 - val_recall: 0.7362
Epoch 68/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4234 - recall: 0.8271 -
val_loss: 0.4440 - val_recall: 0.7669
```

```
val_loss: 0.4440 - val_recall: 0.7669
Epoch 69/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4286 - recall: 0.8254 -
val_loss: 0.4422 - val_recall: 0.7485
Epoch 70/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4289 - recall: 0.8177 -
val_loss: 0.4419 - val_recall: 0.7546
Epoch 71/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4235 - recall: 0.8303 -
val_loss: 0.4523 - val_recall: 0.7638
Epoch 72/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4276 - recall: 0.8340 -
val_loss: 0.4270 - val_recall: 0.7331
Epoch 73/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4233 - recall: 0.8257 -
val_loss: 0.4485 - val_recall: 0.7669
Epoch 74/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4216 - recall: 0.8354 -
val_loss: 0.4289 - val_recall: 0.7270
Epoch 75/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4204 - recall: 0.8269 -
val_loss: 0.4203 - val_recall: 0.7055
Epoch 76/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4200 - recall: 0.8271 -
val_loss: 0.4318 - val_recall: 0.7546
Epoch 77/100
160/160 [==============================] - 1s 7ms/step - loss: 0.4204 - recall: 0.8363 -
val_loss: 0.4246 - val_recall: 0.7086
Epoch 78/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4188 - recall: 0.8371 -
val_loss: 0.4215 - val_recall: 0.7117
Epoch 79/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4213 - recall: 0.8312 -
val_loss: 0.4118 - val_recall: 0.7025
Epoch 80/100
160/160 [==============================] - 1s 7ms/step - loss: 0.4209 - recall: 0.8301 -
val_loss: 0.4337 - val_recall: 0.7393
Epoch 81/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4185 - recall: 0.8303 -
val_loss: 0.4382 - val_recall: 0.7454
Epoch 82/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4164 - recall: 0.8409 -
val_loss: 0.4363 - val_recall: 0.7485
Epoch 83/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4211 - recall: 0.8308 -
val_loss: 0.4277 - val_recall: 0.7117
Epoch 84/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4232 - recall: 0.8261 -
val_loss: 0.4357 - val_recall: 0.7423
Epoch 85/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4184 - recall: 0.8277 -
val_loss: 0.4525 - val_recall: 0.7761
Epoch 86/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4171 - recall: 0.8346 -
val_loss: 0.4502 - val_recall: 0.7669
Epoch 87/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4193 - recall: 0.8348 -
val_loss: 0.4376 - val_recall: 0.7515
Epoch 88/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4168 - recall: 0.8336 -
val_loss: 0.4334 - val_recall: 0.7485
Epoch 89/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4211 - recall: 0.8338 -
val_loss: 0.4427 - val_recall: 0.7577
Epoch 90/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4209 - recall: 0.8295 -
val_loss: 0.4376 - val_recall: 0.7546
Epoch 91/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4179 - recall: 0.8279 -
val_loss: 0.4377 - val_recall: 0.7393
Epoch 92/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4204 - recall: 0.8352 -
val_loss: 0.4360 - val_recall: 0.7086
```

```
Epoch 93/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4156 - recall: 0.8401 -
val_loss: 0.4434 - val_recall: 0.7577
Epoch 94/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4148 - recall: 0.8430 -
val_loss: 0.4210 - val_recall: 0.7239
Epoch 95/100
160/160 [==============================] - 1s 4ms/step - loss: 0.4126 - recall: 0.8371 -
val_loss: 0.4403 - val_recall: 0.7454
Epoch 96/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4164 - recall: 0.8338 -
val_loss: 0.4211 - val_recall: 0.6994
Epoch 97/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4172 - recall: 0.8369 -
val_loss: 0.4239 - val_recall: 0.7270
Epoch 98/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4202 - recall: 0.8277 -
val_loss: 0.4311 - val_recall: 0.7301
Epoch 99/100
160/160 [==============================] - 1s 5ms/step - loss: 0.4160 - recall: 0.8385 -
val_loss: 0.4188 - val_recall: 0.7055
Epoch 100/100
160/160 [==============================] - 1s 6ms/step - loss: 0.4171 - recall: 0.8334 -
val_loss: 0.4440 - val_recall: 0.7362
```

In [113]:

```
#Plotting Train Loss vs Validation Loss
plt.plot(ModelPerformace_5.history['loss'])
plt.plot(ModelPerformace_5.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

In [114]:

```
y_train_pred = model_5.predict(X_train_smote)
y_train_pred = (y_train_pred > 0.5)
y_val_pred = model_5.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
```

```
319/319 [==============================] - 1s 3ms/step
50/50 [==============================] - 0s 3ms/step
```

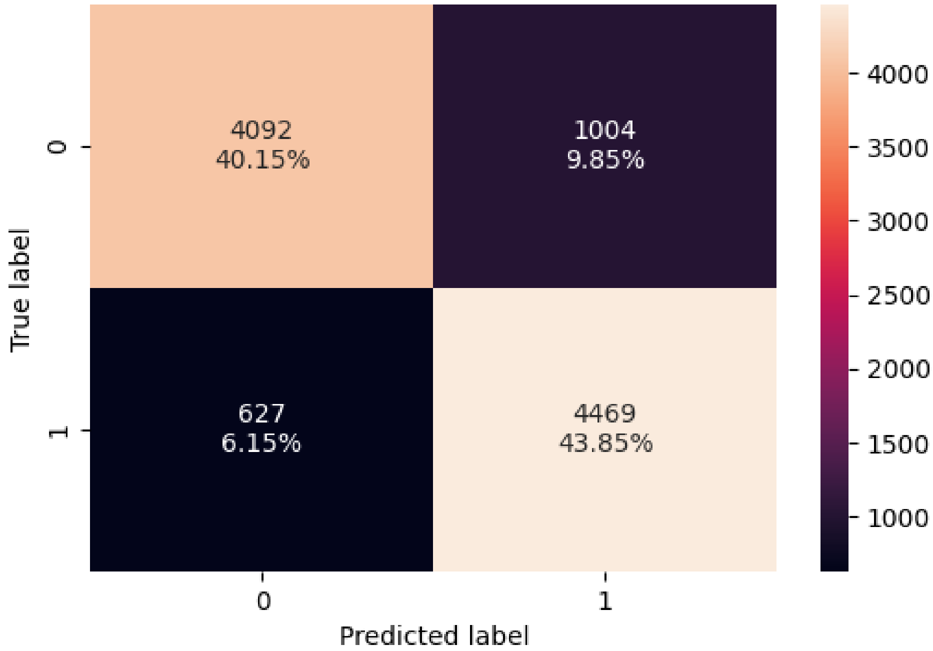
Lets store the results

In [115]:

```
model_name = "NN with SMOTE- Adam with dropout"

trainning_perf.loc[model_name] = recall_score(y_train_smote,y_train_pred)
validation_perf.loc[model_name] = recall_score(y_val,y_val_pred)
```

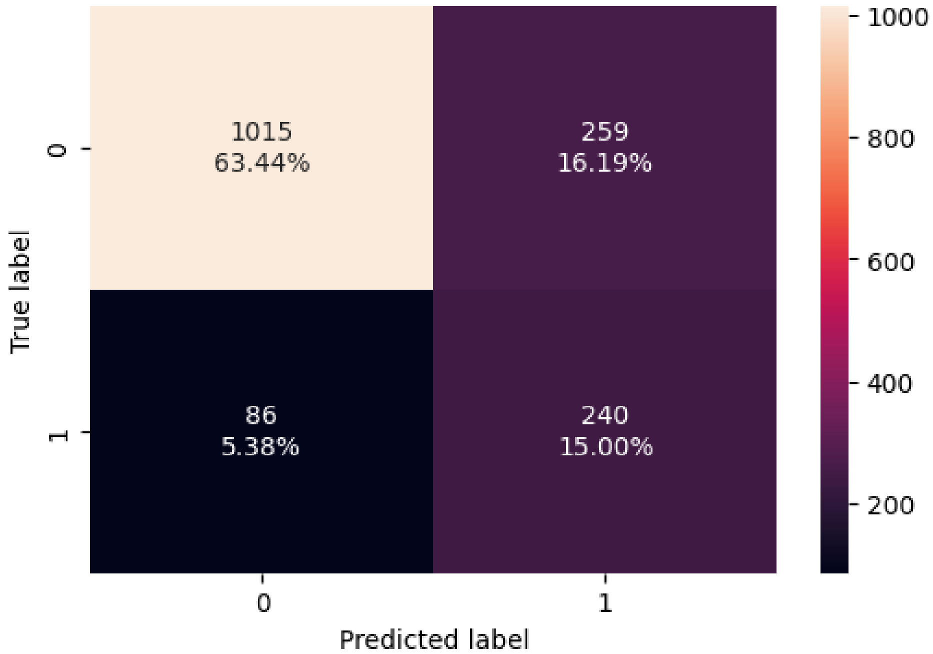
In [116]:

```
make_confusion_matrix(y_train_smote, y_train_pred)
```



In [117]:

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



Model Performance Comparison and Final Model Selection

In [118]:

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

Training performance comparison

Out[118]:

	recall
NN with SGD	0.115798
NN with Adam	0.509202
NN with Adam and Dropout	0.509202
NN with SMOTE and SGD	0.726452
NN with SMOTE & Adam	0.968603
NN with SMOTE- Adam with dropout	0.876962

In [119]:

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

Validation set performance comparison

Out[119]:

	recall
NN with SGD	0.110429
NN with Adam	0.349693
NN with Adam and Dropout	0.349693
NN with SMOTE and SGD	0.684049
NN with SMOTE & Adam	0.592025
NN with SMOTE- Adam with dropout	0.736196

In [120]:

```
trainning_perf - validation_perf
```

Out[120]:

	recall
NN with SGD	0.005368
NN with Adam	0.159509
NN with Adam and Dropout	0.159509
NN with SMOTE and SGD	0.042403
NN with SMOTE & Adam	0.376578
NN with SMOTE- Adam with dropout	0.140766

Actionable Insights and Business Recommendations

Neural Network Model Performance Evaluation (Based on Recall)

The performance of different neural network models was evaluated using **Recall** as the primary metric. Recall measures a model's ability to correctly identify positive instances (true positives), which is particularly important in scenarios where false negatives are costly (e.g., customer churn prediction). Below is a comparison of various models based on their recall scores.

Model Performance Results:

- **NN with SGD:**
  - **Recall:** 0.0920
  - This model demonstrates a very low recall score, indicating poor performance in identifying positive instances. The model is likely biased toward predicting the negative class or is unable to capture key patterns in the data.
- **NN with Adam:**
  - **Recall:** 0.3497
  - The Adam optimizer significantly improves recall compared to SGD, though the recall is still relatively low. This indicates that the model can better identify positive instances but still needs improvement in handling positive class examples.
- **NN with Adam and Dropout:**
  - **Recall:** 0.3497
  - Adding Dropout to the model with Adam does not affect recall. Dropout, a regularization technique to prevent overfitting, may improve generalization but does not seem to strongly impact recall in this case.
- **NN with SMOTE and SGD:**
  - **Recall:** 0.6840
  - SMOTE (Synthetic Minority Over-sampling Technique) balances the dataset by generating synthetic samples for the minority class, leading to a significant improvement in recall. This shows that addressing class imbalance positively affects the model's ability to identify the positive class.
- **NN with SMOTE & Adam:**
  - **Recall:** 0.5920
  - The combination of SMOTE and Adam shows an improvement over the Adam-only model, but it still lags behind the model with SGD and SMOTE. While SMOTE helps balance the dataset, Adam alone doesn't achieve the recall levels seen with SGD and SMOTE.
- **NN with SMOTE, Adam, and Dropout:**
  - **Recall:** 0.7362
  - This model achieves the highest recall score. The combination of SMOTE, Adam optimizer, and Dropout produces the best performance in identifying positive instances. Dropout likely aids in improving generalization by preventing overfitting, while SMOTE ensures better handling of class imbalance, leading to the optimal recall score.

Key Insights:

- **SMOTE Significantly Improves Recall**:
  Models using SMOTE (to address class imbalance) show a notable increase in recall, particularly when combined with optimizers like SGD and Adam. This highlights the importance of addressing class imbalance in improving model performance, especially in binary classification tasks.
- **Dropout's Effect**:
  While Dropout does not seem to improve recall when applied to the Adam optimizer alone, it has a more significant effect when used alongside SMOTE and Adam. This suggests that Dropout helps improve generalization in models that already handle class imbalance effectively.
- **Optimizer Choice Matters**:
  The model with **SGD and SMOTE** outperforms the model with **Adam and SMOTE**, indicating that the choice of optimizer significantly impacts model performance. SGD may be more suitable for this task because its noisier updates can help escape local minima and better identify positive samples.

Conclusion:

- The **best-performing model** in terms of recall is **NN with SMOTE, Adam, and Dropout**, which achieves a recall of **0.7362**. This indicates strong performance in correctly identifying positive instances, which is crucial for applications like churn prediction and fraud detection.
- The **NN with SGD and SMOTE** model, with a recall of **0.6840**, also performs well, but the addition of Adam and Dropout further improves recall, especially in imbalanced datasets.
- Models **without SMOTE** show much poorer recall, underscoring the significant impact of addressing class imbalance for improving model performance.

imbalance for improving model performance.

Business Recommendations

To address the problem of customer attrition from the bank service, we can take a targeted, data-driven approach to develop strategies that tackle the issues mentioned. Here's how you can address each specific problem:

1. Higher Exits from Germany:

- **Solution:**
    - **Localized Promotions:** Germany-specific offers, such as tailored banking products or region-specific financial benefits, could help retain customers.
    - **Cultural Understanding:** Implement surveys or focus groups in Germany to understand the specific reasons behind their exit. Factors like service quality, product offerings, or even geopolitical influences could play a role.
    - **Customer Support Enhancement:** Improve customer support with German-speaking agents, better support channels, and understanding of local needs.
    - **Targeted Retention Campaigns:** Offer personalized retention campaigns for German customers, such as loyalty rewards, discounts, or exclusive services for long-term customers.

2. Females Exiting More than Males:

- **Solution:**
    - **Promote Gender-Specific Benefits:** Promote services that are specifically attractive to female customers. This could include:
        - **Financial Planning for Women:** Tailored financial planning services, such as retirement savings, maternity leave planning, or women-focused investment options.
        - **Female-Friendly Products:** Partner with brands offering discounts or benefits for women (e.g., lifestyle products, wellness services).
    - **Customer Engagement:** Use personalized communication, such as targeted emails or SMS, highlighting products and services that cater to women's financial needs, such as low-fee accounts, savings programs, and female-focused financial advice.
    - **Improved Customer Experience:** Female customers may feel more comfortable with a certain type of service, so consider improving or offering dedicated support for female customers, particularly in financial advisory services.

3. Age and Exiting Bank Service:

- **Solution:**
    - **Understand the Subtle Trends:** Even though there's no clear signal that age is a significant factor, there may still be an underlying issue with older customers. Older customers may feel disconnected from modern banking services, so:
        - **Senior-Friendly Services:** Introduce products tailored to older customers, such as simplified account management, low-fee services, or senior discount packages.
        - **Increased Communication:** Use direct mail, phone calls, or in-branch assistance to communicate more effectively with older clients who may be less tech-savvy.
        - **Customer Education:** Offer workshops or webinars for older customers to educate them on digital banking, online services, and new technology that can make banking easier for them.

4. Inactive Members Leaving Credit Card Service:

- **Solution:**
    - **Re-engagement Campaigns:** Target inactive credit card holders with personalized re-engagement offers, such as:
        - **Bonus Points:** Offer loyalty points, cashback, or other rewards for using the card again.
        - **Exclusive Promotions:** Provide limited-time offers for spending in certain categories (e.g., groceries, dining).
        - **Upgrade Offers:** Offer an upgraded version of the card with additional benefits (such as higher credit limits, better rewards, or exclusive access to events).

limits, better rewards, or exclusive access to events).

- **Behavioral Targeting**: Send reminders or alerts when a customer hasn't used their card in a while, with incentives to start using it again.

5. More Bank Credit Cards Leading to Exits:

- **Solution:**
  - **Analyze Credit Card Overload:** People with multiple credit cards may leave if they feel overwhelmed or burdened by too many offers. Address this issue with:
    - **Simplified Offerings:** Evaluate if a consolidation or simplification of product offerings could help. Consider offering fewer but more attractive card options with better benefits.
    - **Loyalty or Retention Programs**: Reward customers who have multiple bank cards but stay loyal, offering personalized benefits like higher cashback, exclusive offers, or lower interest rates.
    - **Debt Management Solutions**: For customers with multiple cards, provide debt consolidation options or low-interest transfer plans to make the service more manageable.
  - **Personalized Communication:** Target customers with a higher number of credit cards to offer a customized solution, such as consolidating their cards into a single product with added benefits or reduced fees.

6. General Recommendations:

- **Customer Feedback**: Collect more data about why customers are leaving through exit surveys, focus groups, or customer service interactions. This will provide more clarity on specific pain points.
- **Tailored Marketing:** Use machine learning and customer segmentation techniques to develop hyper-targeted marketing campaigns. For example, use a mix of email, SMS, and app notifications to keep customers engaged based on their individual behavior.
- **Proactive Customer Service**: Instead of waiting for customers to exit, implement proactive customer service initiatives that reach out to customers at risk of leaving based on behavioral triggers, such as inactivity or dissatisfaction with certain services.

Conclusion:

By analyzing customer data and implementing targeted interventions, the bank can address these issues, enhance customer retention, and improve customer satisfaction. It's crucial to combine personalized service, localized campaigns, and customized offerings to prevent further exits and foster long-term loyalty.

*

Power Ahead