Bank Churn Prediction

By Raghuram Palaniappan

Problem Statement

Context

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

Objective

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

Data Dictionary

- CustomerId: Unique ID which is assigned to each customer
- Surname: Last name of the customer
- CreditScore: It defines the credit history of the customer.
- Geography: A customer’s location
- Gender: It defines the Gender of the customer
- Age: Age of the customer
- Tenure: Number of years for which the customer has been with the bank
- NumOfProducts: refers to the number of products that a customer has purchased through the bank.
- Balance: Account balance
- HasCrCard: It is a categorical variable which decides whether the customer has credit card or not.
- EstimatedSalary: Estimated salary

- IsActiveMember: Is is a categorical variable which decides whether the customer is active member of the bank or not (Active member in the sense, using bank products regularly, making transactions etc)
- Exited : whether or not the customer left the bank within six month. It can take two values
  - 0 = No (Customer did not leave the bank)
  - 1 = Yes (Customer left the bank)

Importing necessary libraries

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

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

# Library to split data
from sklearn.model_selection import train_test_split

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

# importing different functions to build models
import tensorflow as tf
from tensorflow import keras
from keras import backend
from keras.models import Sequential
from keras.layers import Dense, Dropout

# importing SMOTE
from imblearn.over_sampling import SMOTE

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

import random

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

```
WARNING:tensorflow:From C:\Users\Raghuram\AppData\Roaming\Python\Python311\site-packa
ges\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is depr
ecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Loading the dataset

In [ ]:
```
ds = pd.read_csv("Churn.csv")
```

In [ ]:
```
df = ds.copy()
```

Data Overview

In [ ]:
```
ds.head()
```

Out[ ]:

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

In [ ]:
```
ds.shape
```

Out[ ]:
```
(10000, 14)
```

In [ ]:
```
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 [ ]:
```
ds.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	
RowNumber	10000.0	5.000500e+03	2886.895680	1.00	2500.75	5.000500e+03	7.50025
CustomerId	10000.0	1.569094e+07	71936.186123	15565701.00	15628528.25	1.569074e+07	1.57532
CreditScore	10000.0	6.505288e+02	96.653299	350.00	584.00	6.520000e+02	7.18000
Age	10000.0	3.892180e+01	10.487806	18.00	32.00	3.700000e+01	4.40000
Tenure	10000.0	5.012800e+00	2.892174	0.00	3.00	5.000000e+00	7.00000
Balance	10000.0	7.648589e+04	62397.405202	0.00	0.00	9.719854e+04	1.27644
NumOfProducts	10000.0	1.530200e+00	0.581654	1.00	1.00	1.000000e+00	2.00000
HasCrCard	10000.0	7.055000e-01	0.455840	0.00	0.00	1.000000e+00	1.00000
IsActiveMember	10000.0	5.151000e-01	0.499797	0.00	0.00	1.000000e+00	1.00000
EstimatedSalary	10000.0	1.000902e+05	57510.492818	11.58	51002.11	1.001939e+05	1.49388
Exited	10000.0	2.037000e-01	0.402769	0.00	0.00	0.000000e+00	0.00000

Checking for Missing Values

In [ ]:

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

Out[ ]:

```
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
```

Checking for unique values for each of the column

In [ ]:

```
ds.nunique()
```

Out[ ]:
```
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
```

Observations: The data set is 14 variables and 10,000 rows. There are no missing values. The target dependent variable is the Exited column that determines if a customer left the bank of not within 6 months.

In [ ]:
```
#RowNumber , CustomerId and Surname are unique hence dropping it
ds = ds.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
```

Exploratory Data Analysis

Univariate Analysis

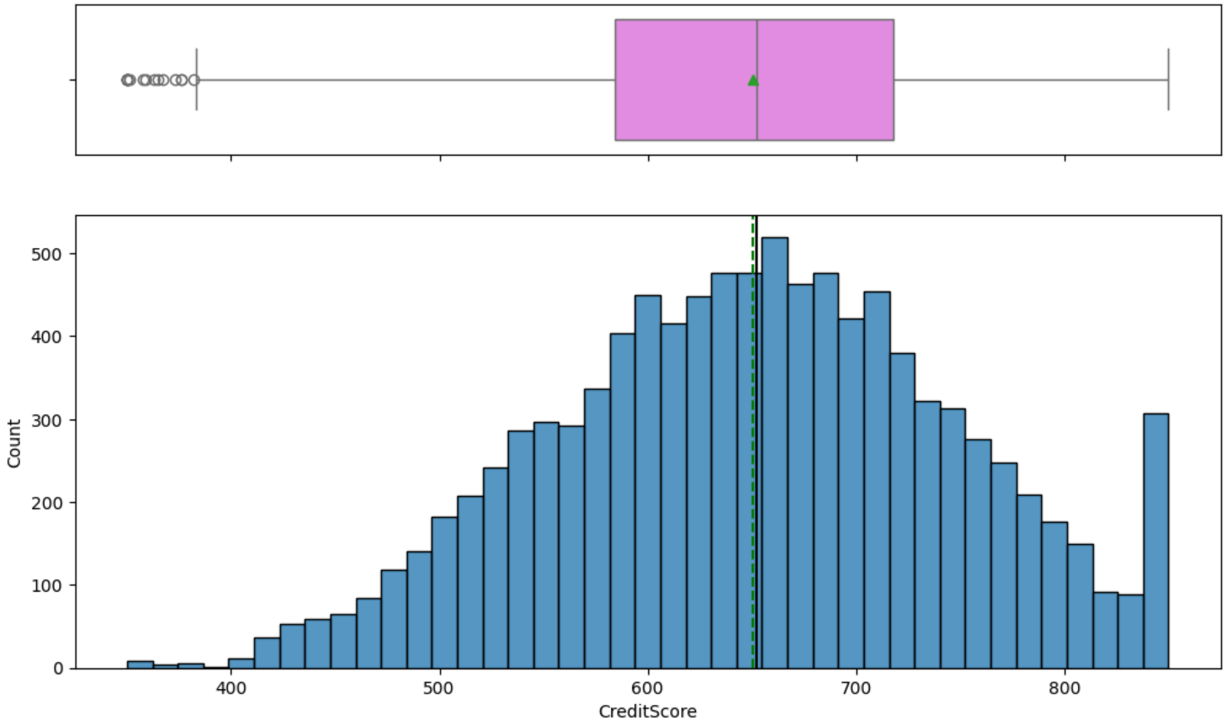
In [ ]:
```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
```

```
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [ ]:
```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

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

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

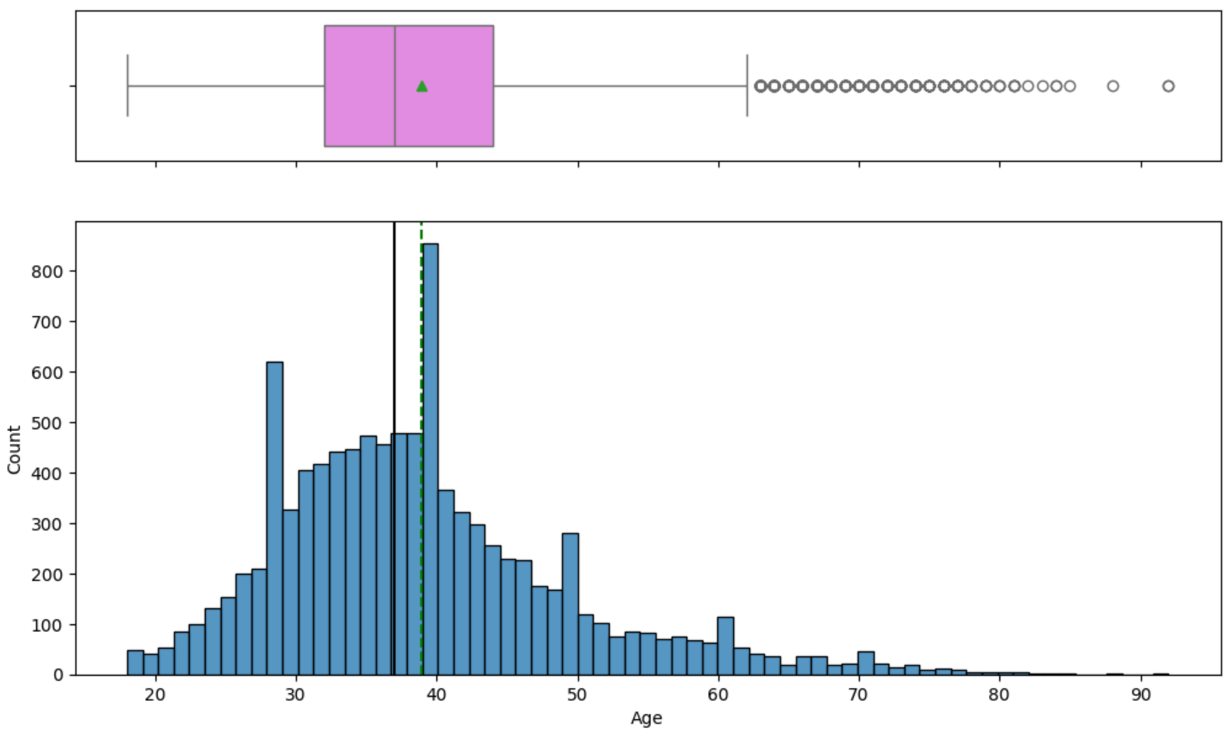
Observations on CreditScore: The distribution is close to symmetrical with outliers on the left side of the data.

In [ ]:
```
histogram_boxplot(ds,'CreditScore')
```



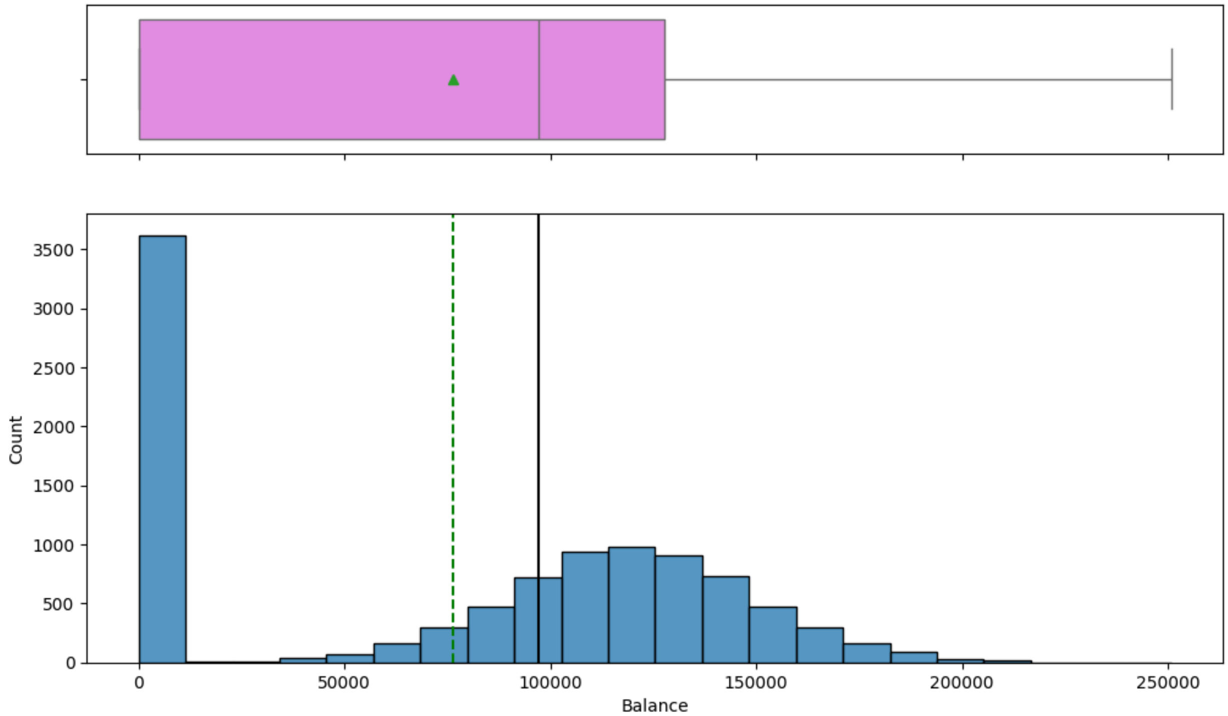
Observations on Age: The data is right skewed due with a lot of outliers in the right side of the data.

In [ ]:
```
histogram_boxplot(ds,'Age')
```



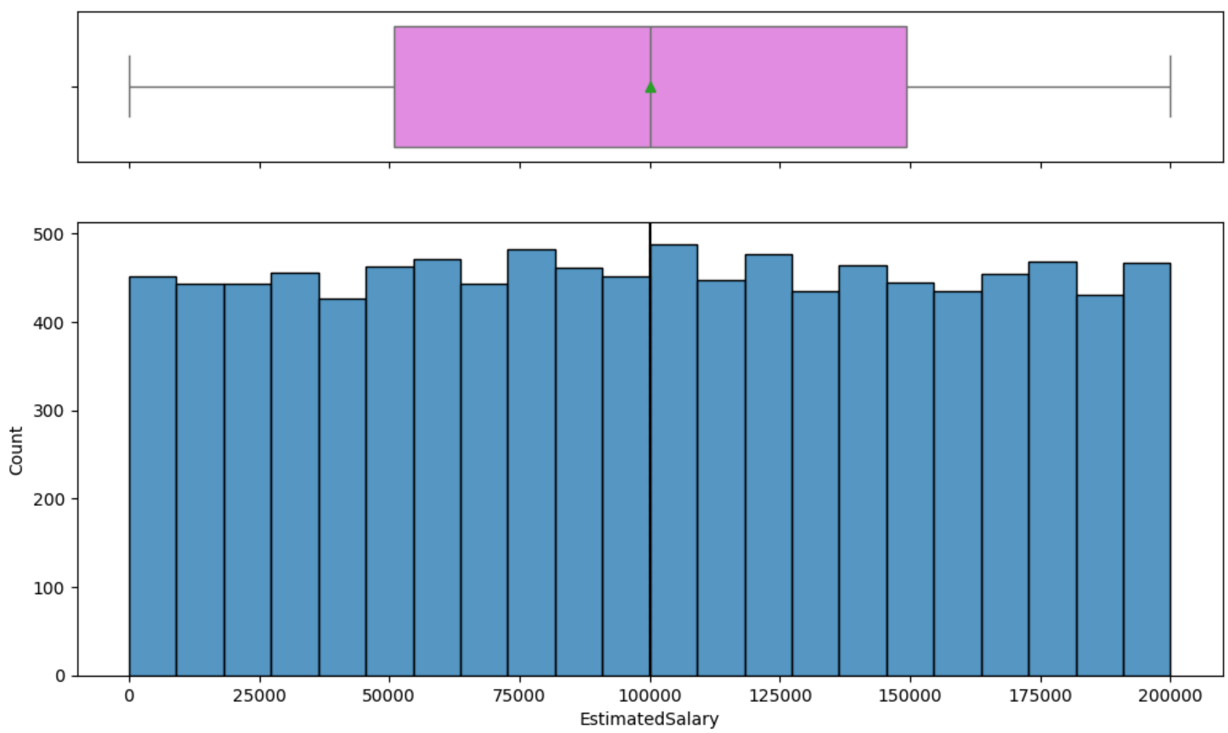
Observations on Balance: Outside of the data points around $0 the distribution is symmetrical. There seem to be a good amount of people with little to no balance in their accounts. The overall data is right skewed.

In [ ]:
```
histogram_boxplot(ds,'Balance')
```



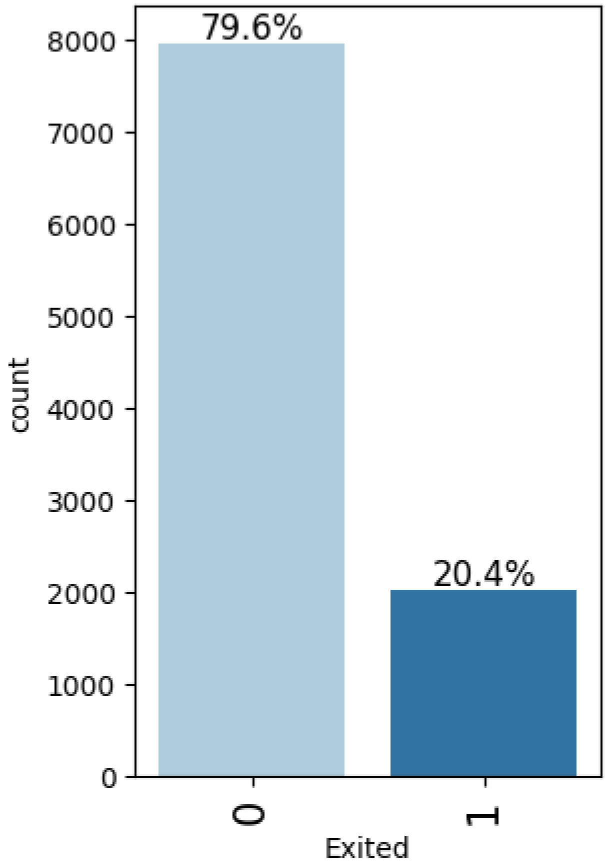
Observations on Estimated Salary: The distribution is pretty uniform. The data is spread relatively even.

In [ ]:
```
histogram_boxplot(ds,'EstimatedSalary')
```



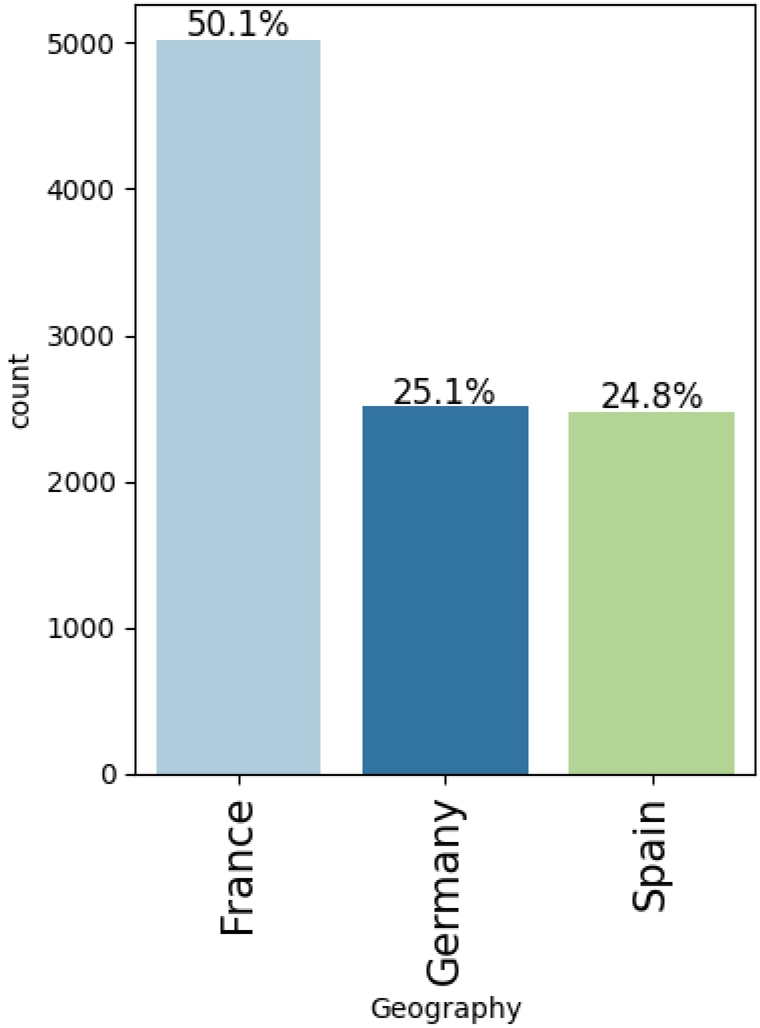
Observations on Exited: Around 20% of customers of the bank have left within the first 6 months.

In [ ]:
```
labeled_barplot(ds, "Exited", perc=True)
```



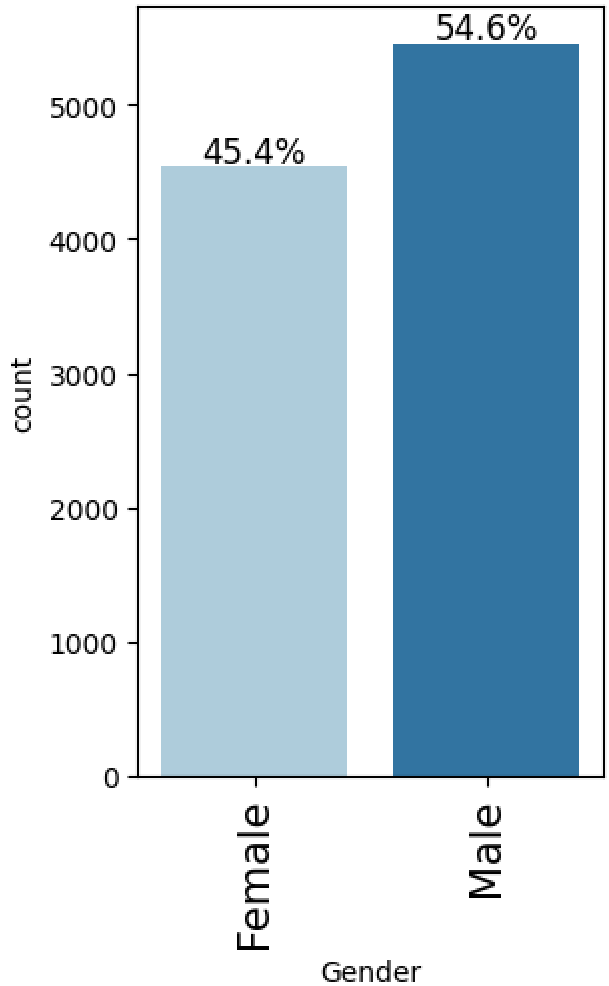
Observations on Geography: Half of the customers come from France with Germany and Spain taking up a quarter each.

In [ ]:
```
labeled_barplot(ds, "Geography", perc=True)
```



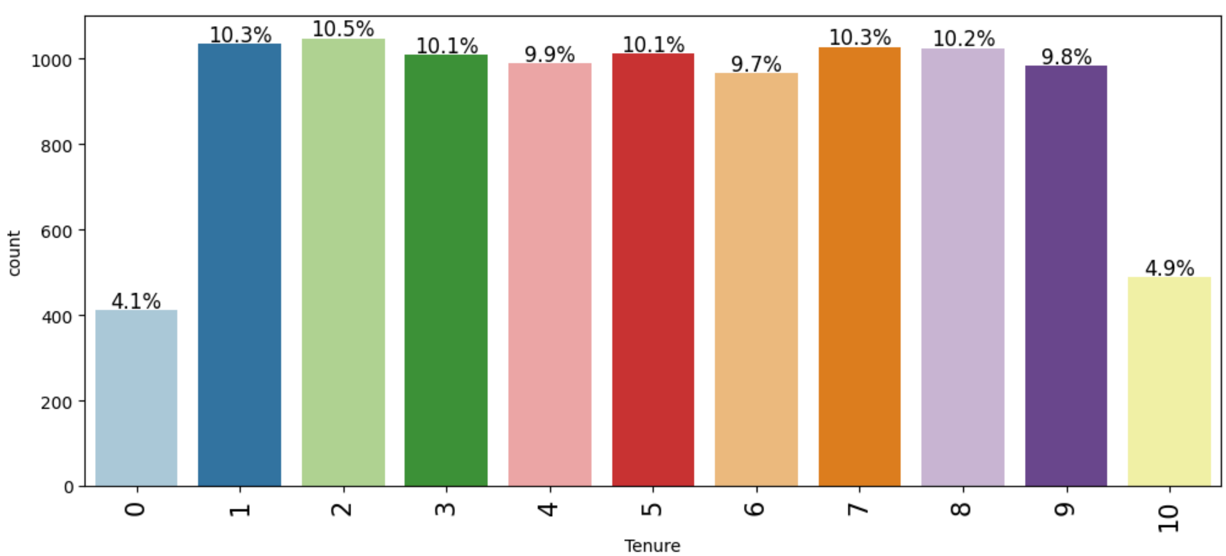
Observations on Gender: More men at 54.6% are customers of the bank.

In [ ]:
```
labeled_barplot(ds, "Gender", perc=True)
```



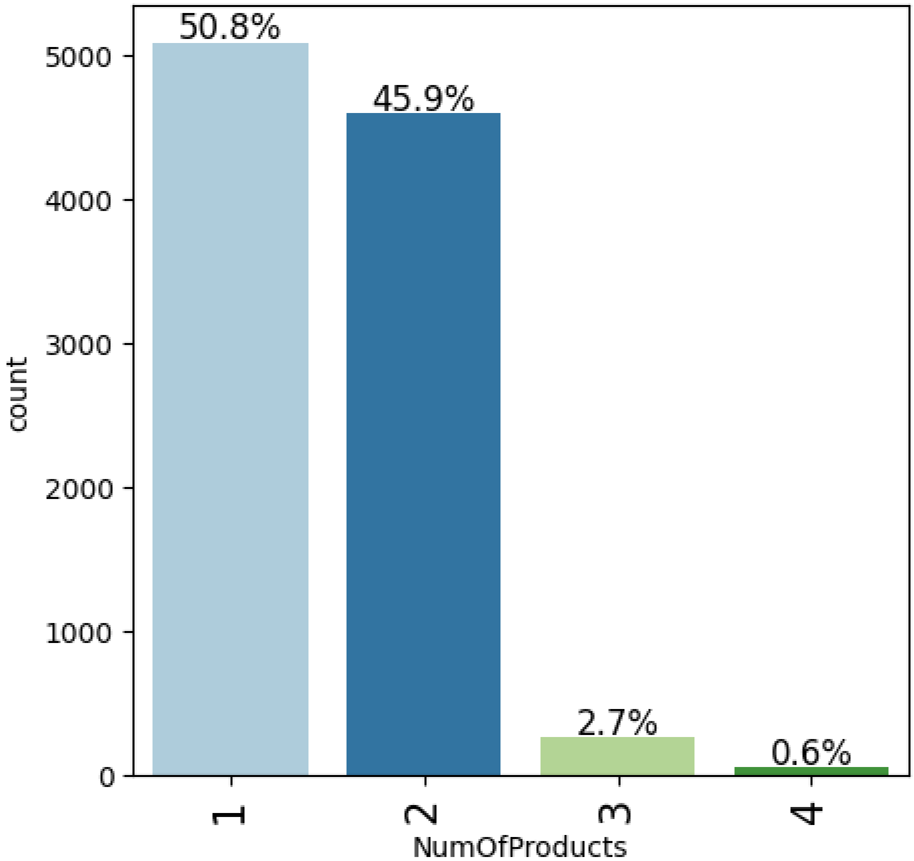
Observations on Tenure: For 1-9 years, at every year there is around 10% of customers who have that long of a tenure.

In [ ]:
```
labeled_barplot(ds, "Tenure", perc=True)
```



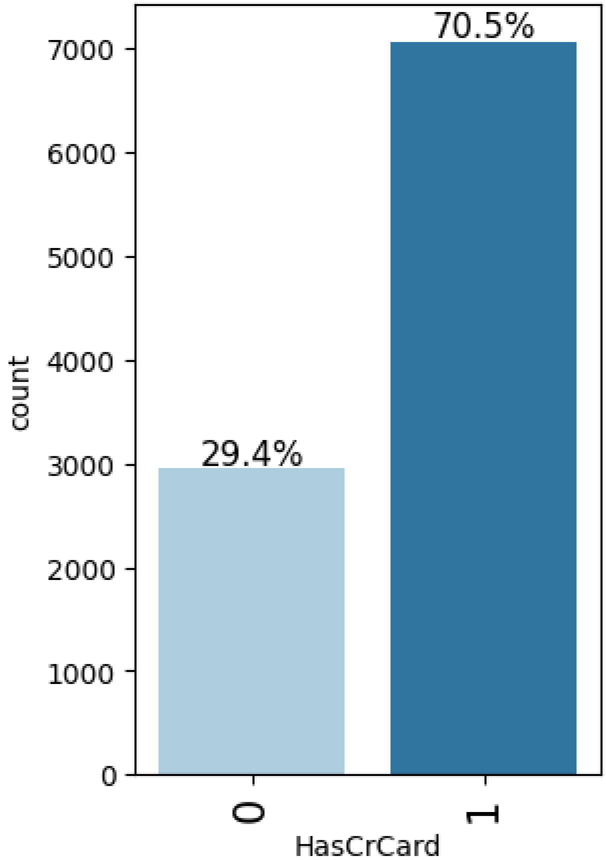
Observations on Number of Products: Half of the customers have purchased 1 product throught the bank and another 45% have purchased exactly 2 products.

In [ ]:
```
labeled_barplot(ds, "NumOfProducts", perc=True)
```



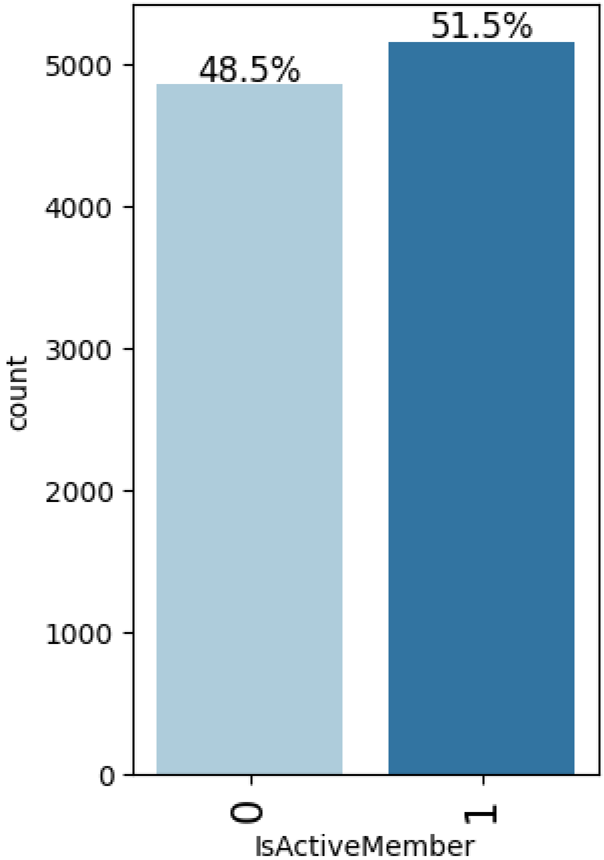
Observations on Has Credit Card: Majority of the customers at 70.5% have credit cards issued by the bank.

In [ ]:
```
labeled_barplot(ds, "HasCrCard", perc=True)
```



Observations on Is Active Member: Slightly over half at 51.5% are active customers at the banks.

In [ ]:
```
labeled_barplot(ds, "IsActiveMember", perc=True)
```



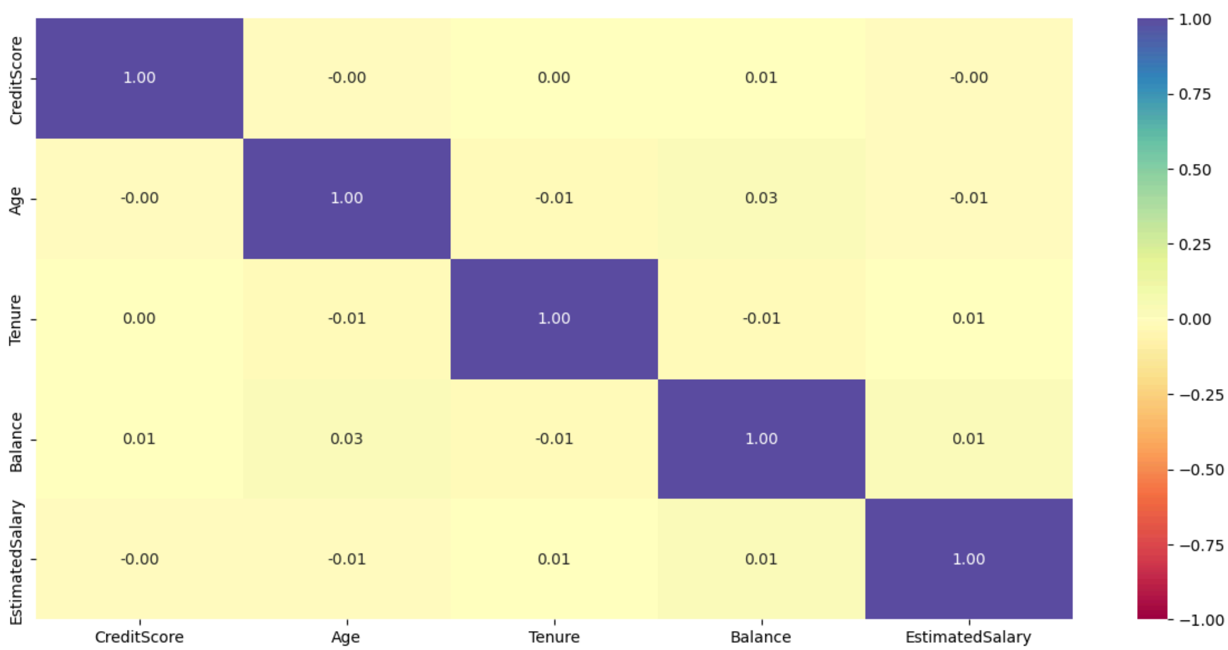
Bivariate Analysis

In [ ]:
```
# function to plot stacked bar chart


def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values(
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 1, 5))
    plt.legend(
        loc="lower left",
        frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

Correlation plot

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

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

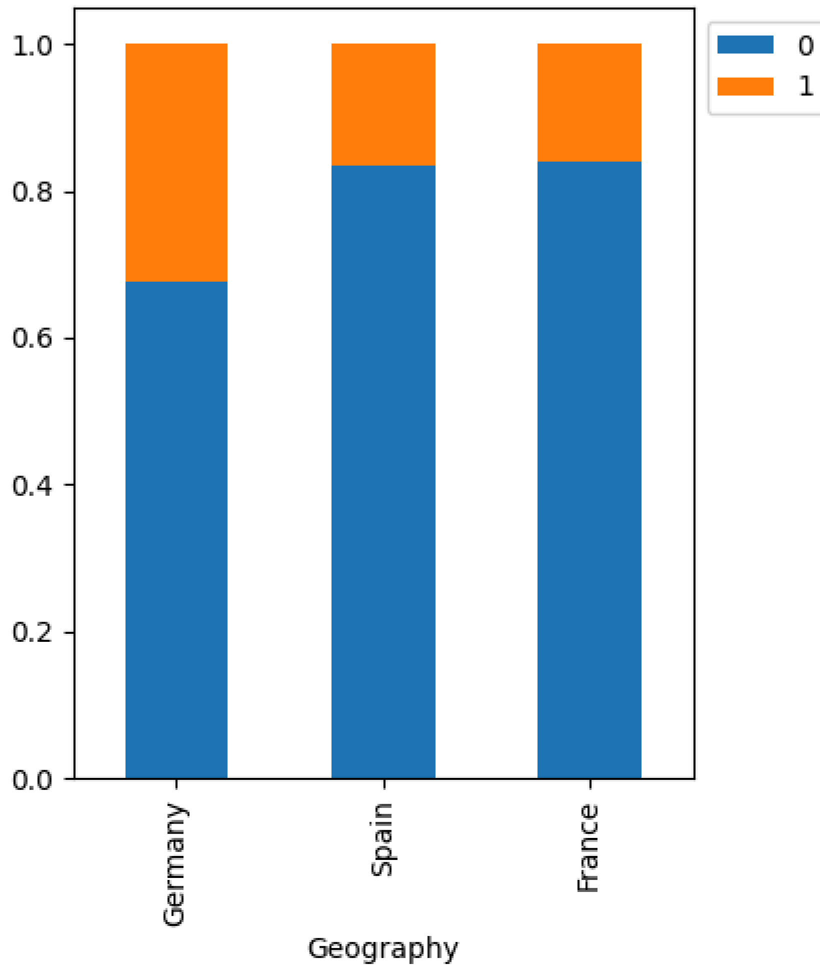


Observation: These variables are excellent due to practically no correlation between each other. This shows no multicollinearity.

Exited Vs Geography

In [ ]:
```
stacked_barplot(ds, "Geography", "Exited")
```

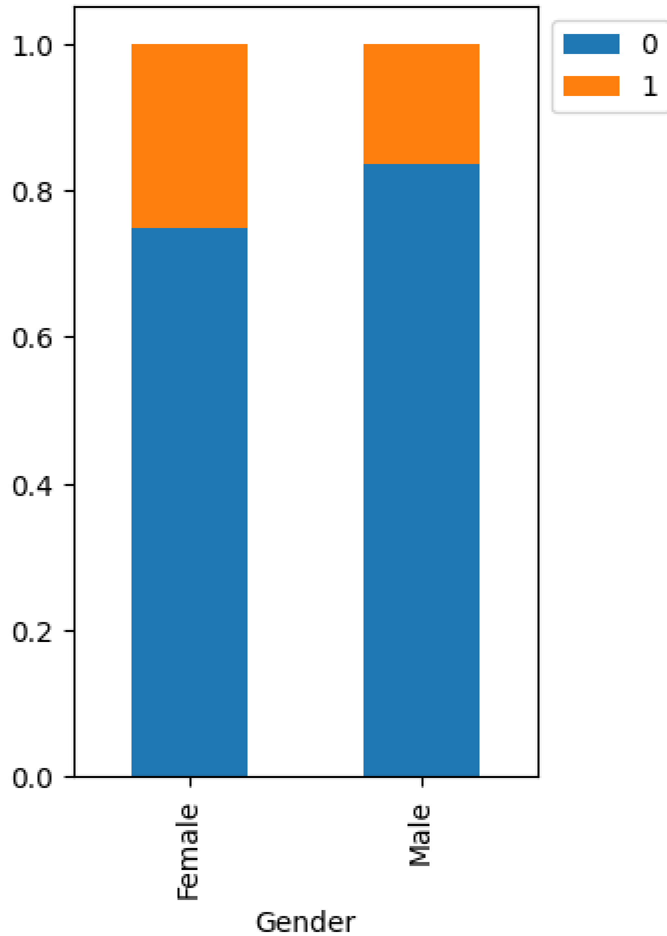
```
Exited        0     1    All
Geography
All        7963  2037  10000
Germany    1695   814   2509
France     4204   810   5014
Spain      2064   413   2477
------------------------------------------------------------------------------------------------------------------------
```



Exited Vs Gender

In [ ]:
```
stacked_barplot(ds, "Gender", "Exited")
```

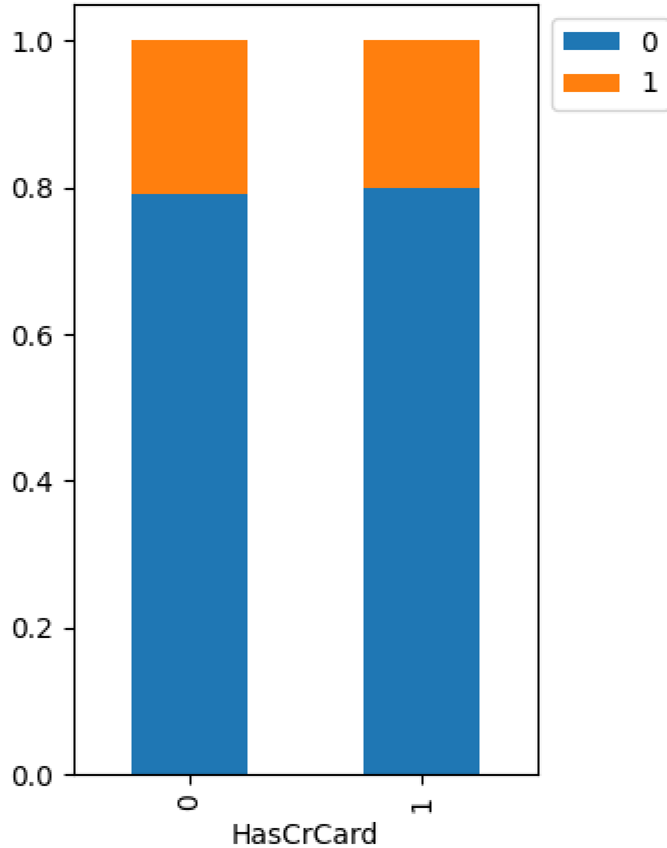
```
Exited     0     1    All
Gender
All     7963  2037  10000
Female  3404  1139   4543
Male    4559   898   5457
------------------------------------------------------------------------------------------------------------------------
```



Exited Vs Has Credit Card

In [ ]:
```
stacked_barplot(ds,"HasCrCard", "Exited")
```

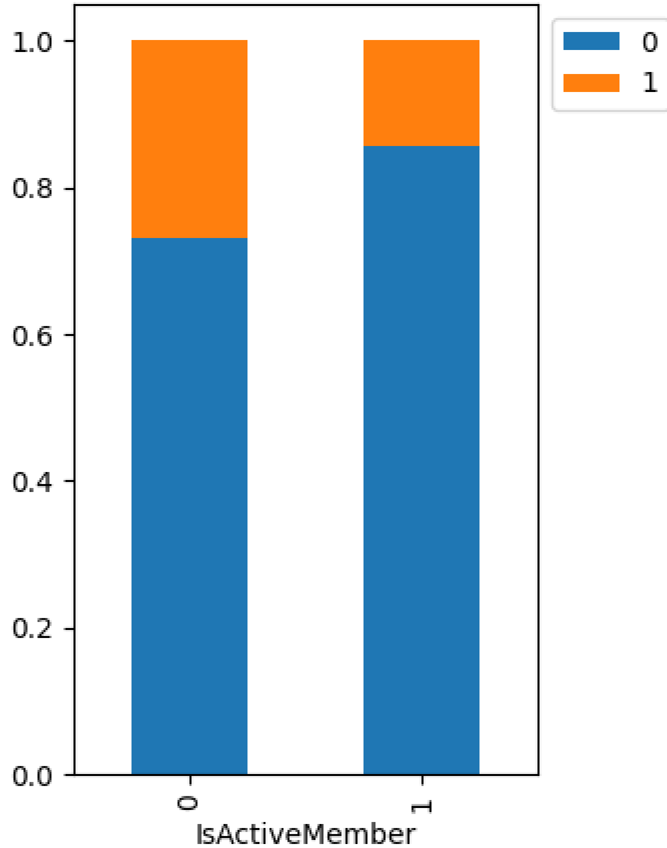
```
Exited        0     1    All
HasCrCard
All        7963  2037  10000
1          5631  1424   7055
0          2332   613   2945
------------------------------------------------------------------------------------
----------------------------------
```



Exited Vs Active Member

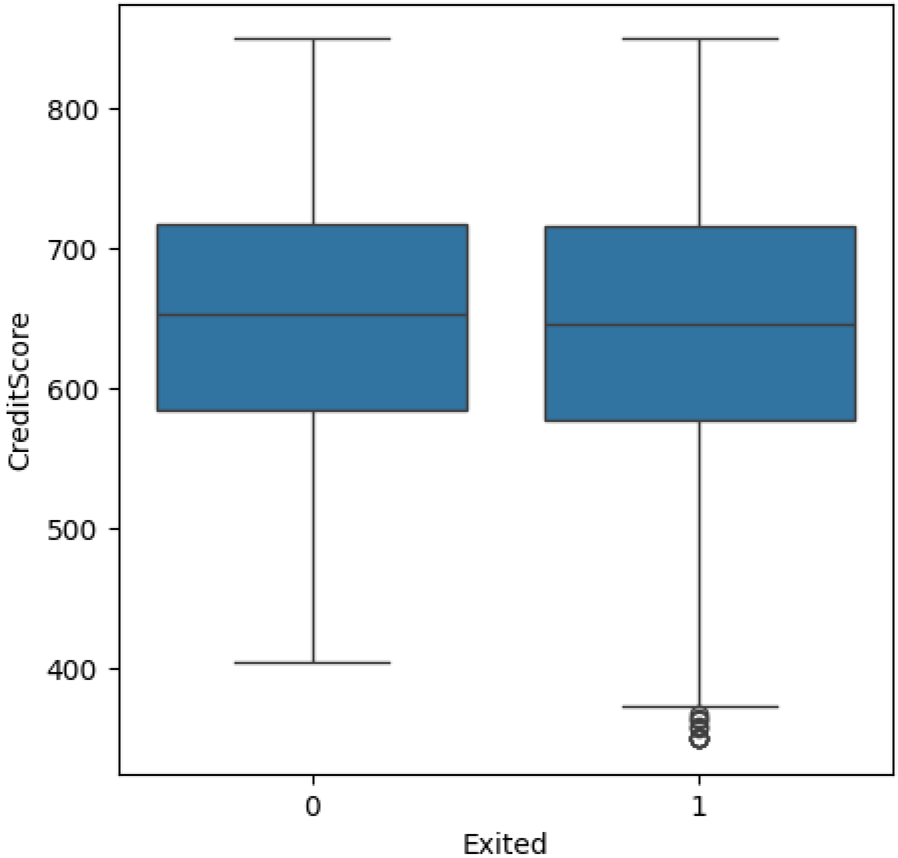
```
In [ ]: stacked_barplot(ds, "IsActiveMember", "Exited")
```

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



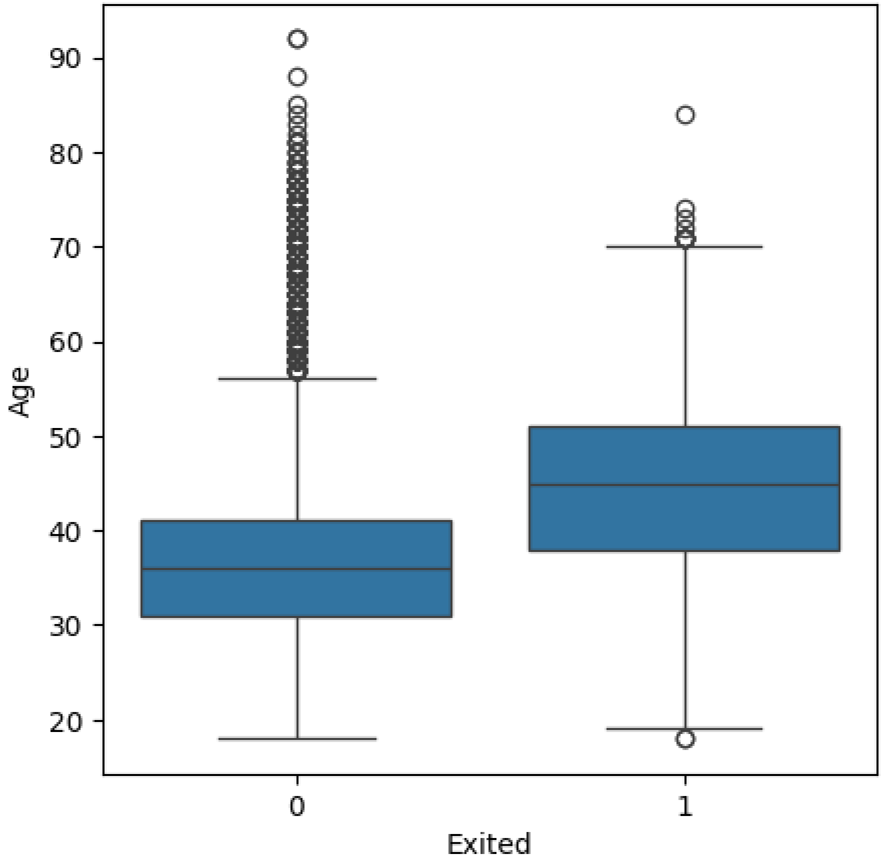
Exited Vs Credit Score

In [ ]:
```
plt.figure(figsize=(5,5))
sns.boxplot(y='CreditScore',x='Exited',data=ds)
plt.show()
```



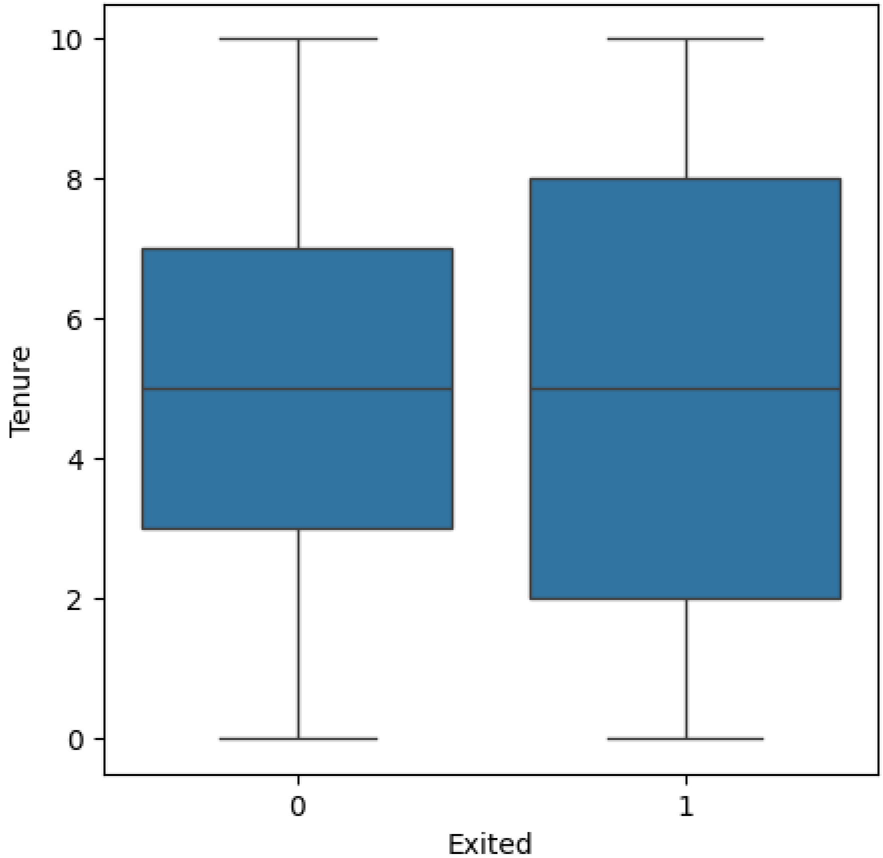
Exited Vs Age

In [ ]:
```
plt.figure(figsize=(5,5))
sns.boxplot(y='Age',x='Exited',data=ds)
plt.show()
```



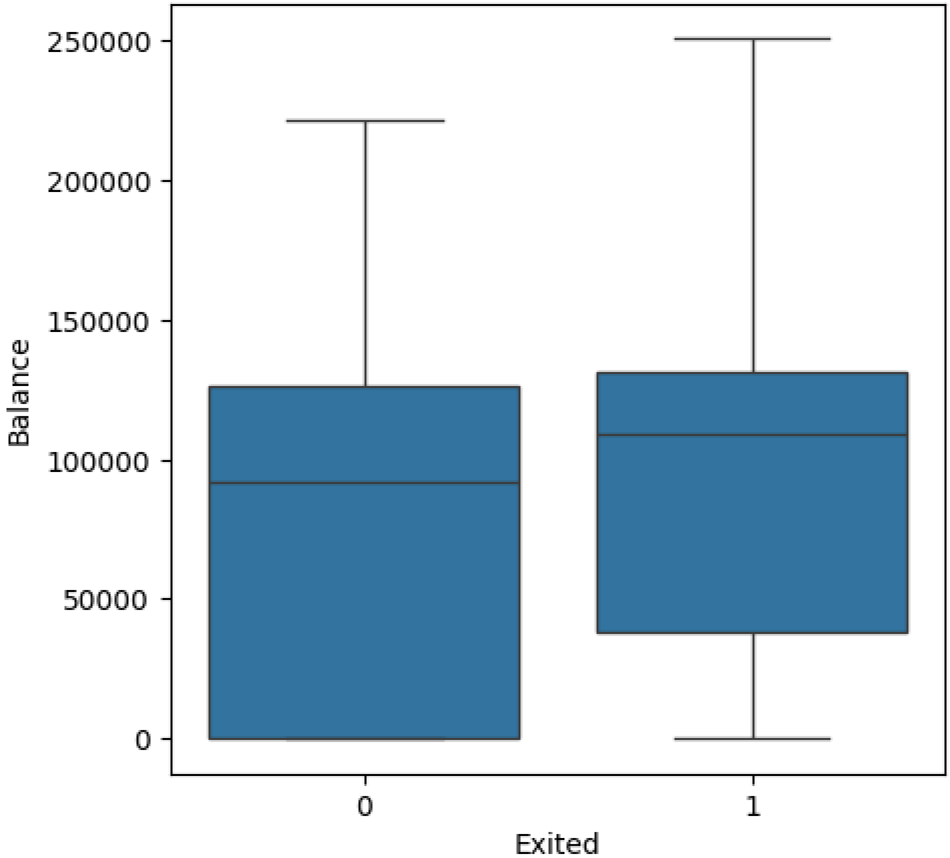
Exited Vs Tenure

In [ ]:
```
plt.figure(figsize=(5,5))
sns.boxplot(y='Tenure',x='Exited',data=ds)
plt.show()
```



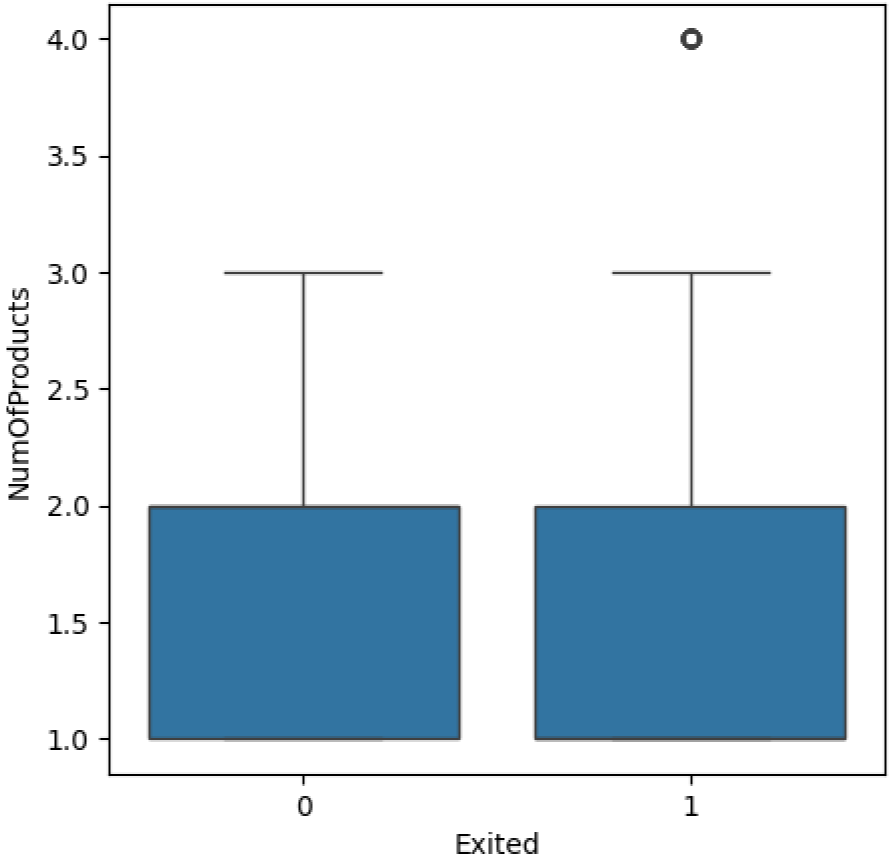
Exited Vs Balance

In [ ]:
```
plt.figure(figsize=(5,5))
sns.boxplot(y='Balance',x='Exited',data=ds)
plt.show()
```



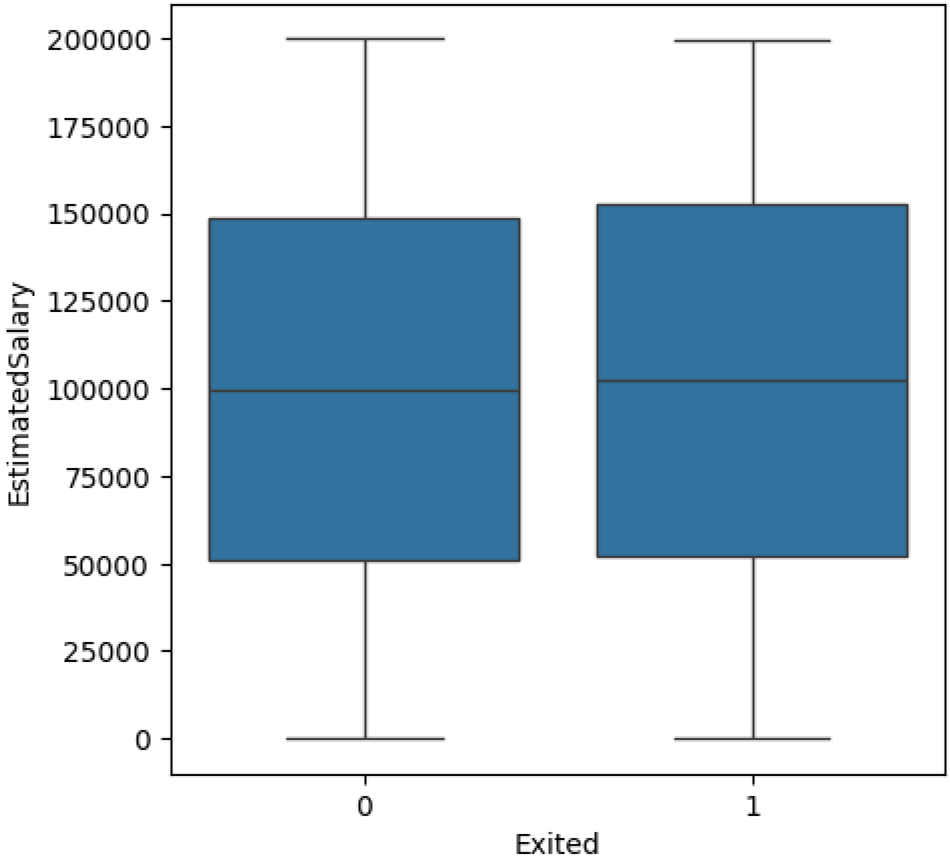
Exited Vs Number of Products

In [ ]:
```
plt.figure(figsize=(5,5))
sns.boxplot(y='NumOfProducts',x='Exited',data=ds)
plt.show()
```



Exited Vs Estimated Salary

In [ ]:
```
plt.figure(figsize=(5,5))
sns.boxplot(y='EstimatedSalary',x='Exited',data=ds)
plt.show()
```



Observations:

- Geography: The highest ratio of exited customers are from Germany. This variable is a good predictor.
- Gender: The majority of exited customers are females, hence this variable also a good predictor.
- IsActiveMember: A higher ratio of exited customers is observed for non-active customers compared to active customers. This is a good predictor.

EDA Observations

- From the Univariate Analysis we conclude:
  - All bank customers come are based in France, Germany or Spain.
  - The majority of customers are from France, are Male, and have not exited the bank
- From the Bivariate Anslysis we conclude:
  - There is no multicollinearity between features.
  - Good predictors: Geography, Gender, and IsActiveMember

Data Preprocessing

Dummy Variable Creation

In [ ]:
```
ds = pd.get_dummies(ds,columns=ds.select_dtypes(include=["object"]).columns.tolist(),d
```

Train-validation-test Split

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

In [ ]:
```
# Splitting the dataset into the Training and Testing set.

X_large, X_test, y_large, y_test = train_test_split(X, y, test_size = 0.2, random_stat
```

In [ ]:
```
# Splitting the dataset into the Training and Testing set.

X_train, X_val, y_train, y_val = train_test_split(X_large, y_large, test_size = 0.2, r
```

In [ ]:
```
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(6400, 11) (1600, 11) (2000, 11)
```

In [ ]:
```
print(y_train.shape, y_val.shape, y_test.shape)
```

```
(6400,) (1600,) (2000,)
```

Data Normalization

Since all the numerical values are on a different scale, so we will be scaling all the numerical values to bring them to the same scale.

In [ ]:
```
# creating an instance of the standard scaler
sc = StandardScaler()

cols_list = ['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary']

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

Observations: The code has now been split to train test validation sets and the columns have all been scaled properly.

Model Building

Model Evaluation Criterion

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

- Our main goal is to reduce false negatives so Recall is our best Metric to select the best models

Let's create a function for plotting the confusion matrix

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

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

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

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

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

Neural Network with SGD Optimizer

In [ ]:
```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the sam
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

```
WARNING:tensorflow:From C:\Users\Raghuram\AppData\Roaming\Python\Python311\site-packa
ges\keras\src\backend.py:277: The name tf.reset_default_graph is deprecated. Please u
se tf.compat.v1.reset_default_graph instead.
```

In [ ]:
```
#Initializing the neural network
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 [ ]:
```
# SGD Optimizer
optimizer = tf.keras.optimizers.SGD(0.001)
metric = keras.metrics.Recall()
```

In [ ]:
```
## Model with binary cross entropy as loss function and recall as the metric.
model_0.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[metric])
```

In [ ]:
```
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 [ ]:

```
# Fitting the ANN
history_0 = model_0.fit(
    X_train, y_train,
    batch_size=32,
    validation_data=(X_val,y_val),
    epochs=20,
    verbose=1
)
```

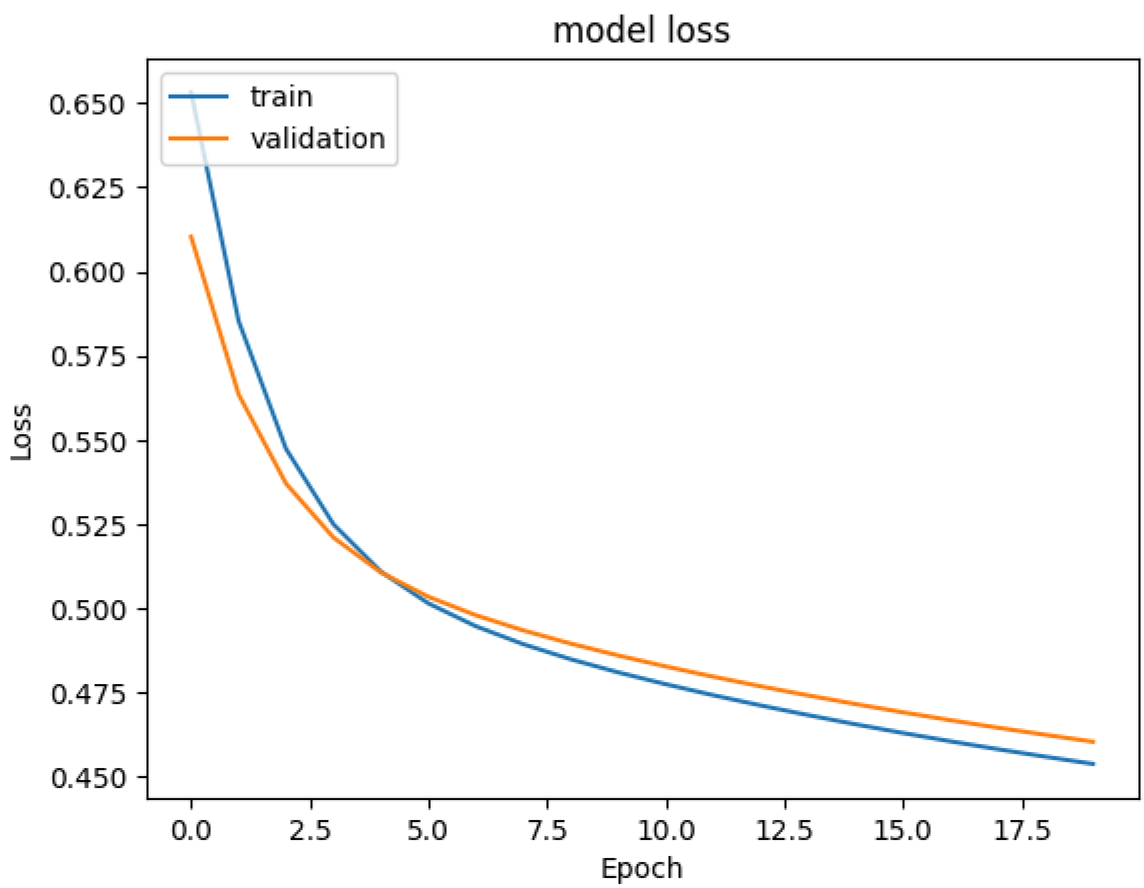
```
Epoch 1/20
WARNING:tensorflow:From C:\Users\Raghuram\AppData\Roaming\Python\Python311\site-packa
ges\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecat
ed. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

200/200 [==============================] - 8s 16ms/step - loss: 0.6531 - recall: 0.21
24 - val_loss: 0.6103 - val_recall: 0.0429
Epoch 2/20
200/200 [==============================] - 2s 9ms/step - loss: 0.5853 - recall: 0.016
9 - val_loss: 0.5635 - val_recall: 0.0000e+00
Epoch 3/20
200/200 [==============================] - 3s 16ms/step - loss: 0.5473 - recall: 0.00
00e+00 - val_loss: 0.5370 - val_recall: 0.0000e+00
Epoch 4/20
200/200 [==============================] - 2s 12ms/step - loss: 0.5249 - recall: 0.00
00e+00 - val_loss: 0.5210 - val_recall: 0.0000e+00
Epoch 5/20
200/200 [==============================] - 1s 4ms/step - loss: 0.5109 - recall: 0.000
0e+00 - val_loss: 0.5106 - val_recall: 0.0000e+00
Epoch 6/20
200/200 [==============================] - 3s 16ms/step - loss: 0.5015 - recall: 0.00
00e+00 - val_loss: 0.5034 - val_recall: 0.0000e+00
Epoch 7/20
200/200 [==============================] - 4s 22ms/step - loss: 0.4947 - recall: 0.00
00e+00 - val_loss: 0.4980 - val_recall: 0.0000e+00
Epoch 8/20
200/200 [==============================] - 4s 22ms/step - loss: 0.4894 - recall: 0.00
00e+00 - val_loss: 0.4934 - val_recall: 0.0000e+00
Epoch 9/20
200/200 [==============================] - 4s 19ms/step - loss: 0.4849 - recall: 0.00
00e+00 - val_loss: 0.4895 - val_recall: 0.0000e+00
Epoch 10/20
200/200 [==============================] - 4s 21ms/step - loss: 0.4810 - recall: 0.00
00e+00 - val_loss: 0.4860 - val_recall: 0.0000e+00
Epoch 11/20
200/200 [==============================] - 3s 16ms/step - loss: 0.4775 - recall: 0.00
00e+00 - val_loss: 0.4828 - val_recall: 0.0000e+00
Epoch 12/20
200/200 [==============================] - 4s 21ms/step - loss: 0.4742 - recall: 0.00
00e+00 - val_loss: 0.4797 - val_recall: 0.0000e+00
Epoch 13/20
200/200 [==============================] - 4s 22ms/step - loss: 0.4712 - recall: 0.00
00e+00 - val_loss: 0.4769 - val_recall: 0.0000e+00
Epoch 14/20
200/200 [==============================] - 4s 22ms/step - loss: 0.4683 - recall: 0.00
00e+00 - val_loss: 0.4742 - val_recall: 0.0000e+00
Epoch 15/20
200/200 [==============================] - 4s 20ms/step - loss: 0.4656 - recall: 0.00
00e+00 - val_loss: 0.4716 - val_recall: 0.0000e+00
Epoch 16/20
200/200 [==============================] - 2s 11ms/step - loss: 0.4630 - recall: 0.00
00e+00 - val_loss: 0.4691 - val_recall: 0.0000e+00
Epoch 17/20
200/200 [==============================] - 3s 15ms/step - loss: 0.4606 - recall: 0.00
00e+00 - val_loss: 0.4668 - val_recall: 0.0000e+00
Epoch 18/20
200/200 [==============================] - 4s 21ms/step - loss: 0.4582 - recall: 0.00
00e+00 - val_loss: 0.4646 - val_recall: 0.0000e+00
Epoch 19/20
200/200 [==============================] - 4s 20ms/step - loss: 0.4560 - recall: 0.00
```

```
00e+00 - val_loss: 0.4625 - val_recall: 0.0000e+00
Epoch 20/20
200/200 [==============================] - 4s 22ms/step - loss: 0.4539 - recall: 0.00
00e+00 - val_loss: 0.4604 - val_recall: 0.0031
```

Loss function

In [ ]:

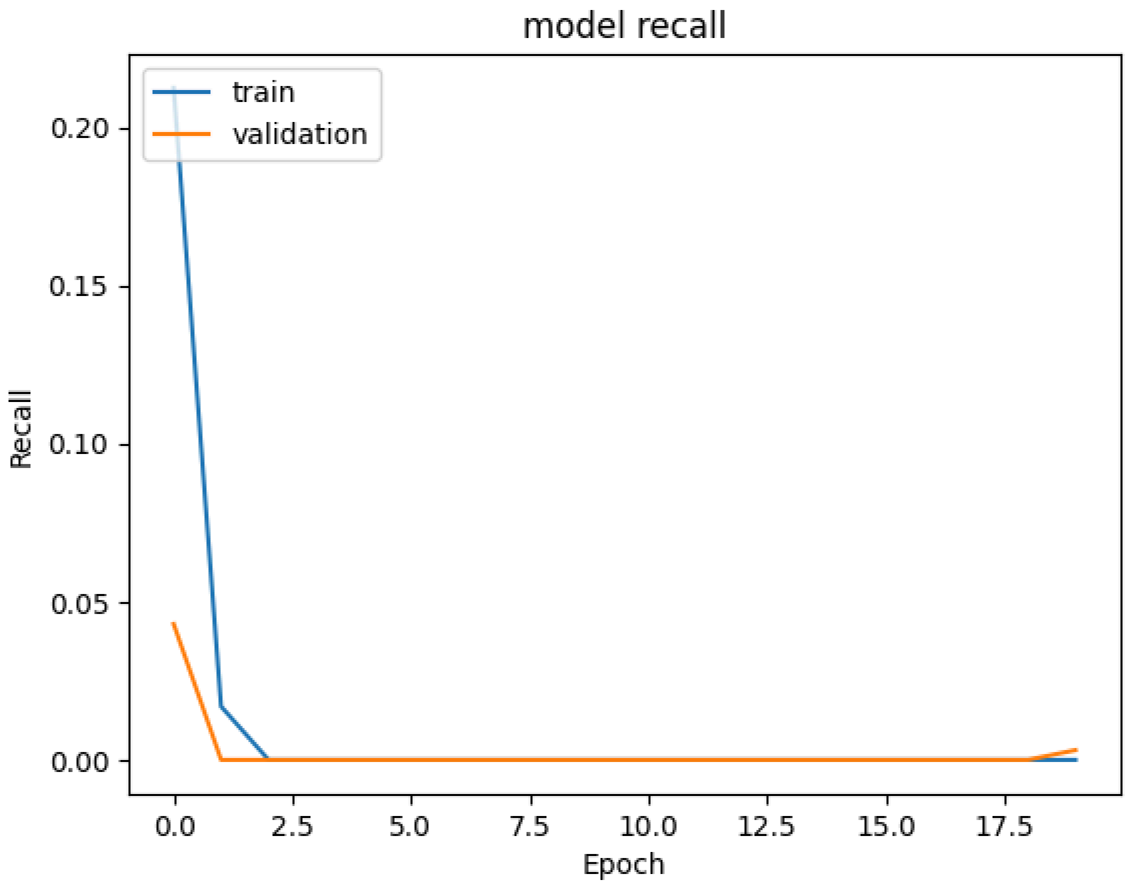
```python
#Plotting Train Loss vs Validation Loss
plt.plot(history_0.history['loss'])
plt.plot(history_0.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



Recall

In [ ]:

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



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

Out[ ]:
```
200/200 [==============================] - 4s 13ms/step
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [ ]:
```
#Predicting the results using best as a threshold
y_val_pred = model_0.predict(X_val)
y_val_pred = (y_val_pred > 0.5)
y_val_pred
```

Out[ ]:
```
50/50 [==============================] - 1s 13ms/step
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [ ]:
```
model_name = "NN with SGD"
```

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

Classification report

In [ ]:
```
#lassification report
cr = classification_report(y_train, y_train_pred)
print(cr)
```

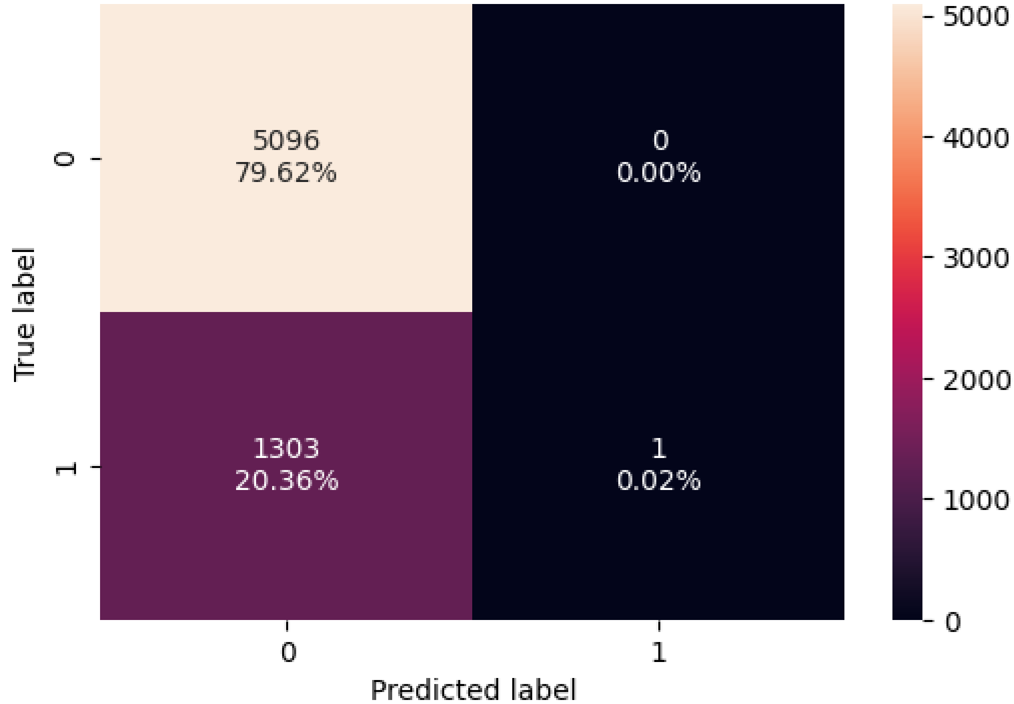
```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      5096
           1       1.00      0.00      0.00      1304

    accuracy                           0.80      6400
   macro avg       0.90      0.50      0.44      6400
weighted avg       0.84      0.80      0.71      6400
```

In [ ]:
```
#classification report
cr=classification_report(y_val, y_val_pred)
print(cr)
```

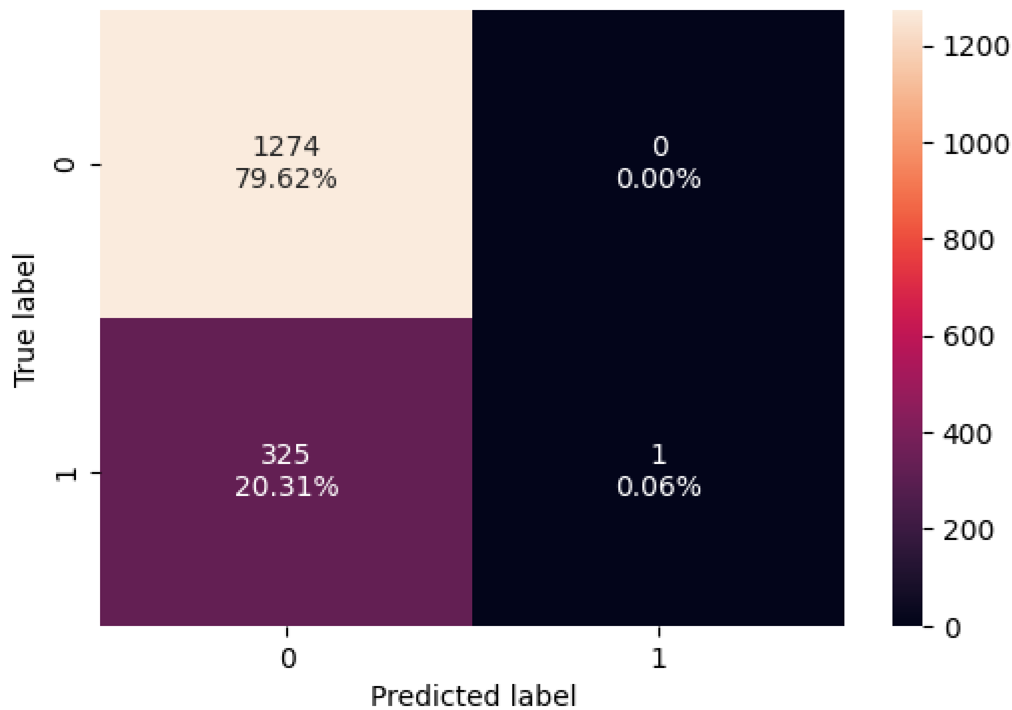
```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1274
           1       1.00      0.00      0.01       326

    accuracy                           0.80      1600
   macro avg       0.90      0.50      0.45      1600
weighted avg       0.84      0.80      0.71      1600
```

Confusion matrix

In [ ]:
```
make_confusion_matrix(y_train, y_train_pred)
```



In [ ]:
```
make_confusion_matrix(y_val, y_val_pred)
```



Model Performance Improvement

Neural Network with Adam Optimizer

In [ ]:
```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the sam
np.random.seed(2)
```

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

In [ ]:
```
#Initializing the neural network
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 [ ]:
```
# Adam Optimizer
optimizer = tf.keras.optimizers.Adam()
metric = keras.metrics.Recall()
```

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

In [ ]:
```
model_1.summary()
```

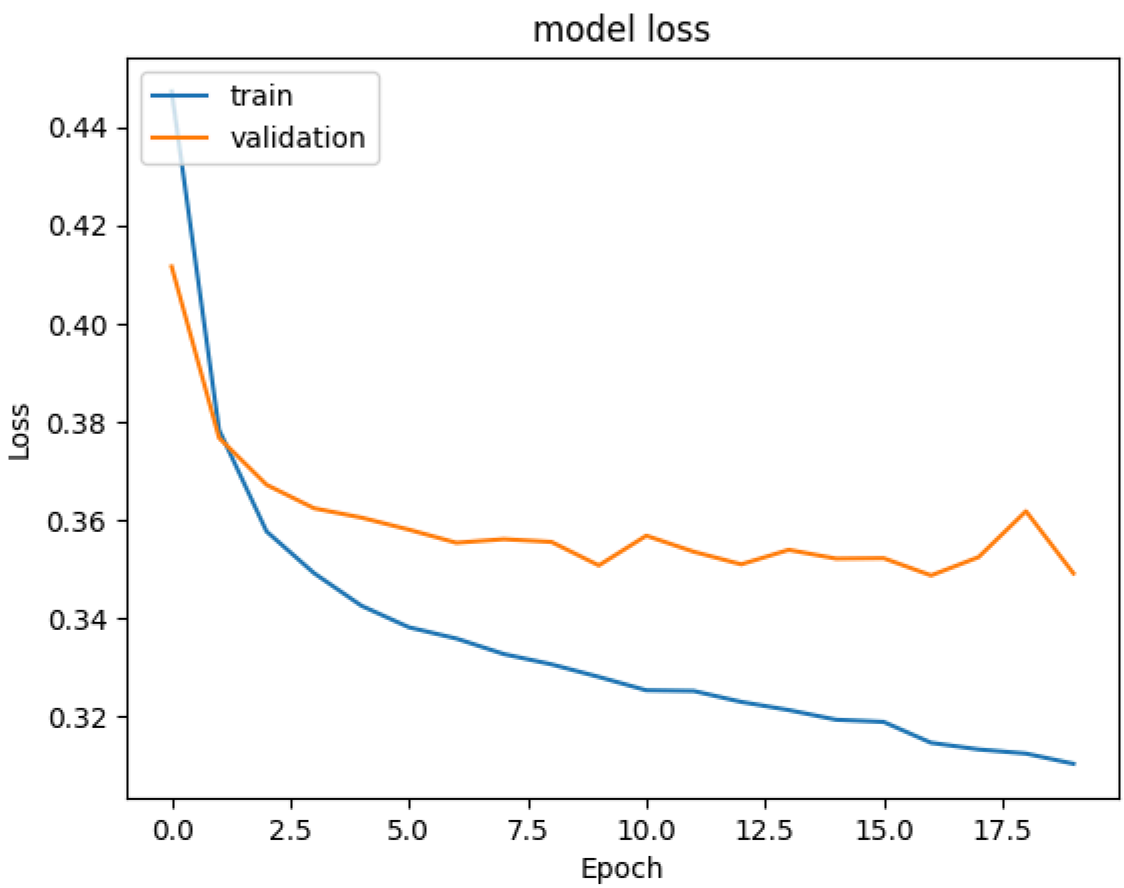
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                768

 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 [ ]:
```
#Fitting the ANN batch_size=32 and epochs=20
history_1 = model_1.fit(
    X_train,y_train,
    batch_size=32,
    validation_data=(X_val,y_val),
    epochs=20,
    verbose=1
)
```

```
Epoch 1/20
200/200 [==============================] - 6s 5ms/step - loss: 0.4473 - recall: 0.125
8 - val_loss: 0.4116 - val_recall: 0.2423
Epoch 2/20
200/200 [==============================] - 1s 5ms/step - loss: 0.3783 - recall: 0.365
8 - val_loss: 0.3767 - val_recall: 0.3957
Epoch 3/20
200/200 [==============================] - 2s 12ms/step - loss: 0.3576 - recall: 0.42
41 - val_loss: 0.3671 - val_recall: 0.4479
Epoch 4/20
200/200 [==============================] - 4s 20ms/step - loss: 0.3491 - recall: 0.43
87 - val_loss: 0.3623 - val_recall: 0.4816
Epoch 5/20
200/200 [==============================] - 4s 21ms/step - loss: 0.3425 - recall: 0.45
78 - val_loss: 0.3604 - val_recall: 0.3620
Epoch 6/20
200/200 [==============================] - 4s 20ms/step - loss: 0.3381 - recall: 0.45
48 - val_loss: 0.3580 - val_recall: 0.4479
Epoch 7/20
200/200 [==============================] - 4s 19ms/step - loss: 0.3358 - recall: 0.47
24 - val_loss: 0.3553 - val_recall: 0.4049
Epoch 8/20
200/200 [==============================] - 5s 25ms/step - loss: 0.3326 - recall: 0.45
94 - val_loss: 0.3560 - val_recall: 0.3988
Epoch 9/20
200/200 [==============================] - 5s 23ms/step - loss: 0.3305 - recall: 0.47
39 - val_loss: 0.3555 - val_recall: 0.3865
Epoch 10/20
200/200 [==============================] - 5s 25ms/step - loss: 0.3280 - recall: 0.46
70 - val_loss: 0.3507 - val_recall: 0.5092
Epoch 11/20
200/200 [==============================] - 3s 16ms/step - loss: 0.3252 - recall: 0.48
24 - val_loss: 0.3567 - val_recall: 0.5429
Epoch 12/20
200/200 [==============================] - 4s 19ms/step - loss: 0.3251 - recall: 0.48
08 - val_loss: 0.3535 - val_recall: 0.4816
Epoch 13/20
200/200 [==============================] - 5s 23ms/step - loss: 0.3229 - recall: 0.49
39 - val_loss: 0.3509 - val_recall: 0.4294
Epoch 14/20
200/200 [==============================] - 4s 22ms/step - loss: 0.3212 - recall: 0.48
47 - val_loss: 0.3539 - val_recall: 0.3896
Epoch 15/20
200/200 [==============================] - 4s 19ms/step - loss: 0.3192 - recall: 0.48
70 - val_loss: 0.3521 - val_recall: 0.3804
Epoch 16/20
200/200 [==============================] - 3s 16ms/step - loss: 0.3188 - recall: 0.49
92 - val_loss: 0.3522 - val_recall: 0.4387
Epoch 17/20
200/200 [==============================] - 4s 18ms/step - loss: 0.3145 - recall: 0.50
38 - val_loss: 0.3486 - val_recall: 0.4540
Epoch 18/20
200/200 [==============================] - 4s 21ms/step - loss: 0.3132 - recall: 0.50
00 - val_loss: 0.3524 - val_recall: 0.3988
Epoch 19/20
200/200 [==============================] - 5s 23ms/step - loss: 0.3124 - recall: 0.50
61 - val_loss: 0.3617 - val_recall: 0.3865
Epoch 20/20
200/200 [==============================] - 5s 23ms/step - loss: 0.3103 - recall: 0.50
15 - val_loss: 0.3491 - val_recall: 0.4693
```

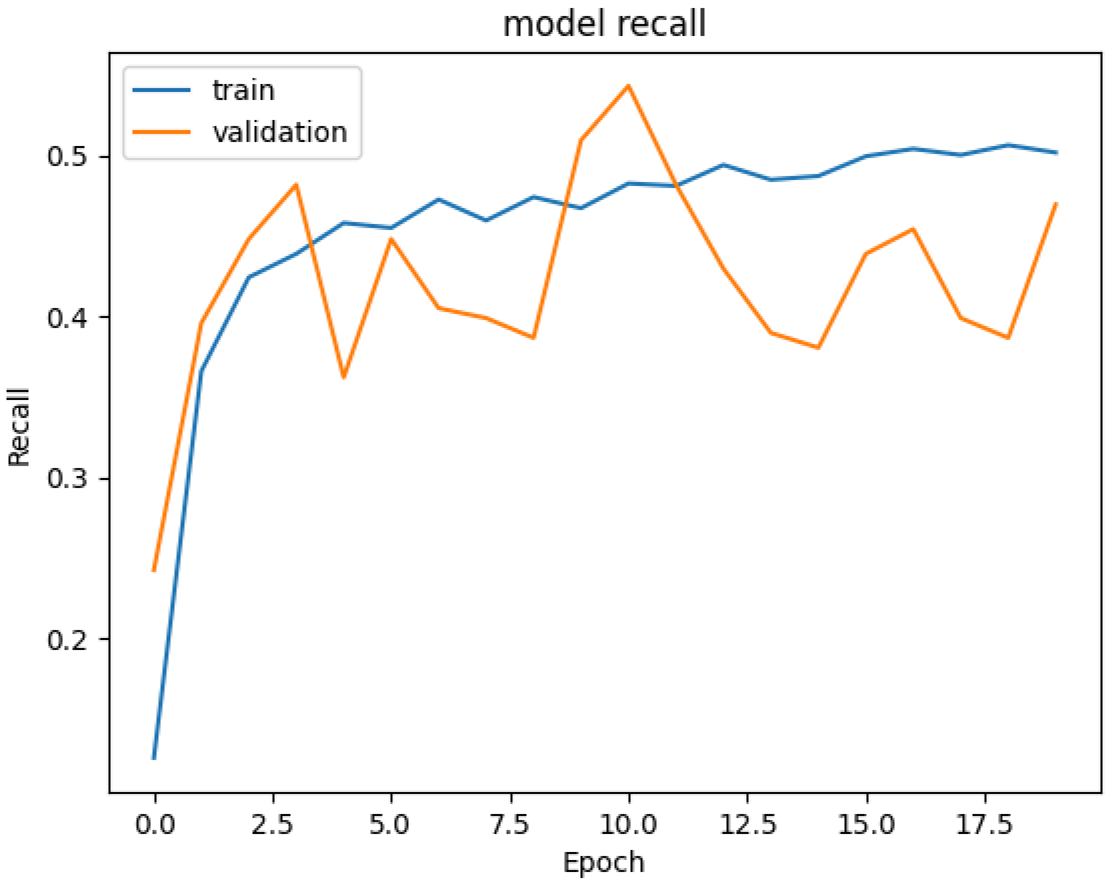
Loss function

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



Recall

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



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

```
200/200 [==============================] - 1s 6ms/step
```

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

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

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

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

In [ ]:
```
model_name = "NN with Adam"
```

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

Classification report

In [ ]:
```
#lassification report
cr=classification_report(y_train,y_train_pred)
print(cr)
```

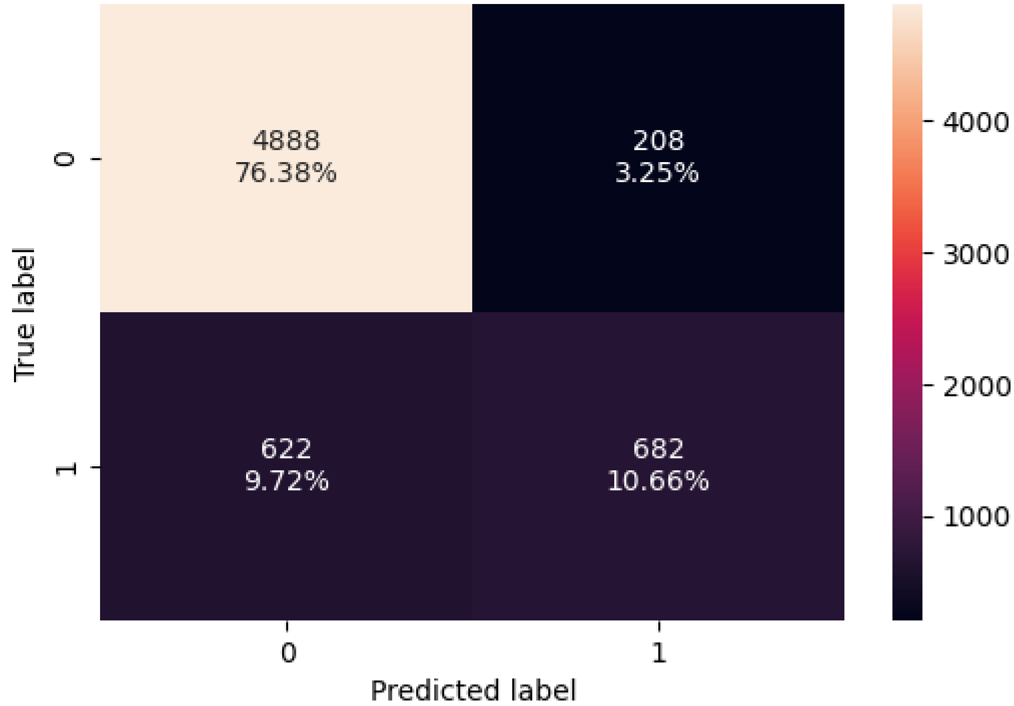
```
              precision    recall  f1-score   support

           0       0.89      0.96      0.92      5096
           1       0.77      0.52      0.62      1304

    accuracy                           0.87      6400
   macro avg       0.83      0.74      0.77      6400
weighted avg       0.86      0.87      0.86      6400
```

In [ ]:
```
#classification report
cr=classification_report(y_val,y_val_pred)
print(cr)
```

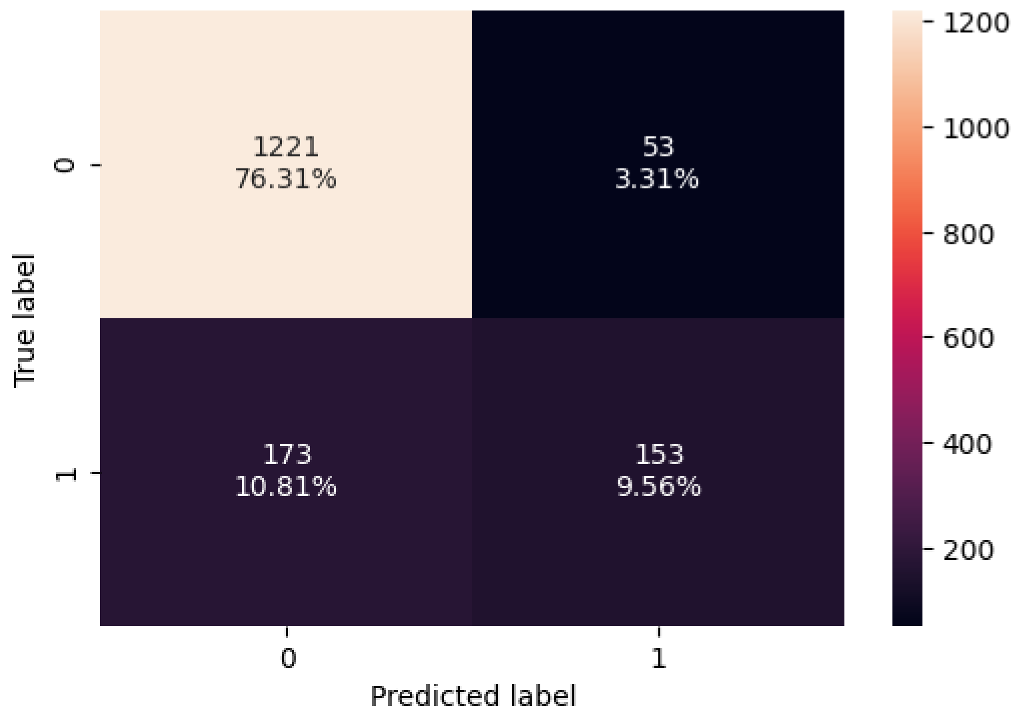
```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1274
           1       0.74      0.47      0.58       326

    accuracy                           0.86      1600
   macro avg       0.81      0.71      0.75      1600
weighted avg       0.85      0.86      0.85      1600
```

Confusion matrix

In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_train, y_train_pred)
```



In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)
```



Neural Network with Adam Optimizer and Dropout

In [ ]:
```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the sam
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [ ]:
```
#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(64,activation='relu'))
model_2.add(Dense(32,activation='relu'))
model_2.add(Dropout(0.2))
model_2.add(Dense(16,activation='relu'))
model_2.add(Dense(1, activation = 'sigmoid'))
```

In [ ]:
```
# Adam
optimizer = tf.keras.optimizers.Adam()
metric = keras.metrics.Recall()
```

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

In [ ]:
```
# Summary of the model
model_2.summary()
```

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

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                384

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 64)                2112

 dense_2 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 16)                528

 dense_4 (Dense)             (None, 1)                 17

=================================================================
Total params: 5121 (20.00 KB)
Trainable params: 5121 (20.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:
```
#Fitting the ANN with batch_size = 32 and 100 epochs
history_2 = model_2.fit(
    X_train,y_train,
    batch_size=32,
    epochs=100,
    verbose=1,
    validation_data=(X_val,y_val)
)
```

```
Epoch 1/100
200/200 [==============================] - 6s 17ms/step - loss: 0.4771 - recall: 0.03
83 - val_loss: 0.4321 - val_recall: 0.0491
Epoch 2/100
200/200 [==============================] - 2s 11ms/step - loss: 0.4246 - recall: 0.18
48 - val_loss: 0.4006 - val_recall: 0.1503
Epoch 3/100
200/200 [==============================] - 3s 13ms/step - loss: 0.3983 - recall: 0.28
45 - val_loss: 0.3756 - val_recall: 0.2577
Epoch 4/100
200/200 [==============================] - 2s 8ms/step - loss: 0.3826 - recall: 0.351
2 - val_loss: 0.3665 - val_recall: 0.5153
Epoch 5/100
200/200 [==============================] - 3s 15ms/step - loss: 0.3766 - recall: 0.38
11 - val_loss: 0.3623 - val_recall: 0.3129
Epoch 6/100
200/200 [==============================] - 5s 23ms/step - loss: 0.3677 - recall: 0.37
73 - val_loss: 0.3603 - val_recall: 0.4233
Epoch 7/100
200/200 [==============================] - 4s 19ms/step - loss: 0.3636 - recall: 0.40
11 - val_loss: 0.3504 - val_recall: 0.3926
Epoch 8/100
200/200 [==============================] - 4s 19ms/step - loss: 0.3587 - recall: 0.42
02 - val_loss: 0.3538 - val_recall: 0.3558
Epoch 9/100
200/200 [==============================] - 5s 25ms/step - loss: 0.3580 - recall: 0.39
65 - val_loss: 0.3494 - val_recall: 0.3681
Epoch 10/100
200/200 [==============================] - 5s 24ms/step - loss: 0.3549 - recall: 0.41
10 - val_loss: 0.3498 - val_recall: 0.4632
Epoch 11/100
200/200 [==============================] - 5s 24ms/step - loss: 0.3486 - recall: 0.42
41 - val_loss: 0.3459 - val_recall: 0.4448
Epoch 12/100
200/200 [==============================] - 4s 19ms/step - loss: 0.3495 - recall: 0.42
48 - val_loss: 0.3558 - val_recall: 0.3834
Epoch 13/100
200/200 [==============================] - 5s 27ms/step - loss: 0.3485 - recall: 0.42
33 - val_loss: 0.3535 - val_recall: 0.3988
Epoch 14/100
200/200 [==============================] - 5s 26ms/step - loss: 0.3495 - recall: 0.42
94 - val_loss: 0.3477 - val_recall: 0.3804
Epoch 15/100
200/200 [==============================] - 5s 25ms/step - loss: 0.3447 - recall: 0.43
48 - val_loss: 0.3446 - val_recall: 0.4233
Epoch 16/100
200/200 [==============================] - 4s 21ms/step - loss: 0.3449 - recall: 0.44
02 - val_loss: 0.3466 - val_recall: 0.4325
Epoch 17/100
200/200 [==============================] - 6s 31ms/step - loss: 0.3492 - recall: 0.44
10 - val_loss: 0.3455 - val_recall: 0.3988
Epoch 18/100
200/200 [==============================] - 5s 25ms/step - loss: 0.3439 - recall: 0.45
86 - val_loss: 0.3435 - val_recall: 0.4264
Epoch 19/100
200/200 [==============================] - 6s 28ms/step - loss: 0.3422 - recall: 0.44
33 - val_loss: 0.3522 - val_recall: 0.4018
Epoch 20/100
200/200 [==============================] - 4s 21ms/step - loss: 0.3419 - recall: 0.44
40 - val_loss: 0.3428 - val_recall: 0.4601
```

```
Epoch 21/100
200/200 [==============================] - 6s 28ms/step - loss: 0.3416 - recall: 0.45
09 - val_loss: 0.3471 - val_recall: 0.4325
Epoch 22/100
200/200 [==============================] - 5s 26ms/step - loss: 0.3400 - recall: 0.45
86 - val_loss: 0.3417 - val_recall: 0.4632
Epoch 23/100
200/200 [==============================] - 5s 26ms/step - loss: 0.3384 - recall: 0.44
63 - val_loss: 0.3460 - val_recall: 0.4387
Epoch 24/100
200/200 [==============================] - 5s 26ms/step - loss: 0.3364 - recall: 0.46
24 - val_loss: 0.3452 - val_recall: 0.4601
Epoch 25/100
200/200 [==============================] - 6s 31ms/step - loss: 0.3365 - recall: 0.46
09 - val_loss: 0.3466 - val_recall: 0.4264
Epoch 26/100
200/200 [==============================] - 7s 34ms/step - loss: 0.3367 - recall: 0.46
17 - val_loss: 0.3445 - val_recall: 0.4172
Epoch 27/100
200/200 [==============================] - 3s 15ms/step - loss: 0.3324 - recall: 0.47
32 - val_loss: 0.3479 - val_recall: 0.4325
Epoch 28/100
200/200 [==============================] - 2s 12ms/step - loss: 0.3304 - recall: 0.47
55 - val_loss: 0.3458 - val_recall: 0.4110
Epoch 29/100
200/200 [==============================] - 1s 7ms/step - loss: 0.3364 - recall: 0.444
8 - val_loss: 0.3450 - val_recall: 0.4663
Epoch 30/100
200/200 [==============================] - 1s 7ms/step - loss: 0.3353 - recall: 0.465
5 - val_loss: 0.3469 - val_recall: 0.4202
Epoch 31/100
200/200 [==============================] - 2s 9ms/step - loss: 0.3286 - recall: 0.471
6 - val_loss: 0.3467 - val_recall: 0.4141
Epoch 32/100
200/200 [==============================] - 2s 8ms/step - loss: 0.3342 - recall: 0.462
4 - val_loss: 0.3437 - val_recall: 0.4049
Epoch 33/100
200/200 [==============================] - 2s 8ms/step - loss: 0.3306 - recall: 0.463
2 - val_loss: 0.3428 - val_recall: 0.4601
Epoch 34/100
200/200 [==============================] - 2s 8ms/step - loss: 0.3296 - recall: 0.466
3 - val_loss: 0.3455 - val_recall: 0.4479
Epoch 35/100
200/200 [==============================] - 1s 7ms/step - loss: 0.3307 - recall: 0.475
5 - val_loss: 0.3486 - val_recall: 0.4908
Epoch 36/100
200/200 [==============================] - 1s 4ms/step - loss: 0.3269 - recall: 0.475
5 - val_loss: 0.3432 - val_recall: 0.4571
Epoch 37/100
200/200 [==============================] - 2s 10ms/step - loss: 0.3250 - recall: 0.49
77 - val_loss: 0.3474 - val_recall: 0.4018
Epoch 38/100
200/200 [==============================] - 2s 9ms/step - loss: 0.3244 - recall: 0.474
7 - val_loss: 0.3458 - val_recall: 0.4601
Epoch 39/100
200/200 [==============================] - 2s 11ms/step - loss: 0.3266 - recall: 0.48
31 - val_loss: 0.3484 - val_recall: 0.4663
Epoch 40/100
200/200 [==============================] - 4s 21ms/step - loss: 0.3238 - recall: 0.48
16 - val_loss: 0.3524 - val_recall: 0.4816
```

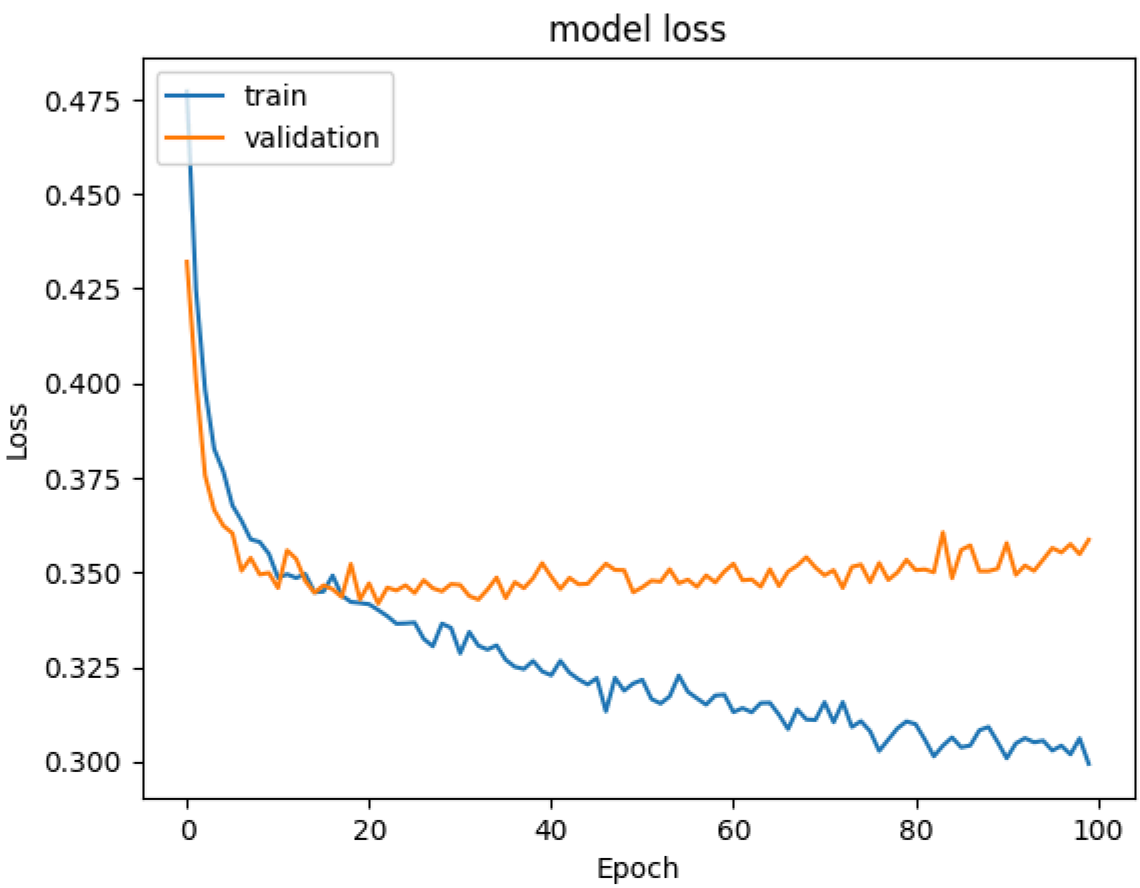
```
Epoch 41/100
200/200 [==============================] - 3s 16ms/step - loss: 0.3228 - recall: 0.48
16 - val_loss: 0.3488 - val_recall: 0.4325
Epoch 42/100
200/200 [==============================] - 1s 7ms/step - loss: 0.3266 - recall: 0.475
5 - val_loss: 0.3456 - val_recall: 0.4202
Epoch 43/100
200/200 [==============================] - 1s 4ms/step - loss: 0.3235 - recall: 0.482
4 - val_loss: 0.3485 - val_recall: 0.4202
Epoch 44/100
200/200 [==============================] - 1s 5ms/step - loss: 0.3217 - recall: 0.484
7 - val_loss: 0.3468 - val_recall: 0.4264
Epoch 45/100
200/200 [==============================] - 2s 9ms/step - loss: 0.3204 - recall: 0.490
8 - val_loss: 0.3470 - val_recall: 0.4417
Epoch 46/100
200/200 [==============================] - 1s 5ms/step - loss: 0.3220 - recall: 0.483
1 - val_loss: 0.3497 - val_recall: 0.4294
Epoch 47/100
200/200 [==============================] - 2s 8ms/step - loss: 0.3133 - recall: 0.499
2 - val_loss: 0.3523 - val_recall: 0.4479
Epoch 48/100
200/200 [==============================] - 1s 4ms/step - loss: 0.3221 - recall: 0.479
3 - val_loss: 0.3506 - val_recall: 0.4110
Epoch 49/100
200/200 [==============================] - 1s 6ms/step - loss: 0.3187 - recall: 0.494
6 - val_loss: 0.3506 - val_recall: 0.4294
Epoch 50/100
200/200 [==============================] - 1s 6ms/step - loss: 0.3206 - recall: 0.480
1 - val_loss: 0.3447 - val_recall: 0.4387
Epoch 51/100
200/200 [==============================] - 1s 4ms/step - loss: 0.3216 - recall: 0.498
5 - val_loss: 0.3461 - val_recall: 0.4110
Epoch 52/100
200/200 [==============================] - 1s 4ms/step - loss: 0.3165 - recall: 0.493
1 - val_loss: 0.3478 - val_recall: 0.4110
Epoch 53/100
200/200 [==============================] - 1s 7ms/step - loss: 0.3153 - recall: 0.506
1 - val_loss: 0.3476 - val_recall: 0.4448
Epoch 54/100
200/200 [==============================] - 3s 15ms/step - loss: 0.3172 - recall: 0.50
77 - val_loss: 0.3508 - val_recall: 0.4049
Epoch 55/100
200/200 [==============================] - 4s 22ms/step - loss: 0.3227 - recall: 0.49
69 - val_loss: 0.3471 - val_recall: 0.4018
Epoch 56/100
200/200 [==============================] - 4s 22ms/step - loss: 0.3184 - recall: 0.50
23 - val_loss: 0.3480 - val_recall: 0.4264
Epoch 57/100
200/200 [==============================] - 4s 21ms/step - loss: 0.3167 - recall: 0.49
39 - val_loss: 0.3462 - val_recall: 0.4325
Epoch 58/100
200/200 [==============================] - 4s 21ms/step - loss: 0.3150 - recall: 0.51
53 - val_loss: 0.3492 - val_recall: 0.4356
Epoch 59/100
200/200 [==============================] - 4s 22ms/step - loss: 0.3174 - recall: 0.49
39 - val_loss: 0.3473 - val_recall: 0.4540
Epoch 60/100
200/200 [==============================] - 4s 21ms/step - loss: 0.3177 - recall: 0.51
23 - val_loss: 0.3502 - val_recall: 0.3804
```

```
Epoch 61/100
200/200 [==============================] - 5s 25ms/step - loss: 0.3131 - recall: 0.50
92 - val_loss: 0.3523 - val_recall: 0.4294
Epoch 62/100
200/200 [==============================] - 4s 22ms/step - loss: 0.3141 - recall: 0.50
46 - val_loss: 0.3479 - val_recall: 0.4571
Epoch 63/100
200/200 [==============================] - 5s 26ms/step - loss: 0.3130 - recall: 0.51
84 - val_loss: 0.3481 - val_recall: 0.4693
Epoch 64/100
200/200 [==============================] - 5s 23ms/step - loss: 0.3154 - recall: 0.51
69 - val_loss: 0.3462 - val_recall: 0.4387
Epoch 65/100
200/200 [==============================] - 5s 26ms/step - loss: 0.3155 - recall: 0.50
46 - val_loss: 0.3508 - val_recall: 0.3834
Epoch 66/100
200/200 [==============================] - 5s 23ms/step - loss: 0.3124 - recall: 0.51
23 - val_loss: 0.3464 - val_recall: 0.4448
Epoch 67/100
200/200 [==============================] - 5s 24ms/step - loss: 0.3085 - recall: 0.52
22 - val_loss: 0.3502 - val_recall: 0.5092
Epoch 68/100
200/200 [==============================] - 5s 25ms/step - loss: 0.3138 - recall: 0.49
62 - val_loss: 0.3518 - val_recall: 0.4356
Epoch 69/100
200/200 [==============================] - 4s 21ms/step - loss: 0.3111 - recall: 0.51
00 - val_loss: 0.3540 - val_recall: 0.4387
Epoch 70/100
200/200 [==============================] - 5s 24ms/step - loss: 0.3110 - recall: 0.51
84 - val_loss: 0.3513 - val_recall: 0.4571
Epoch 71/100
200/200 [==============================] - 5s 24ms/step - loss: 0.3157 - recall: 0.50
00 - val_loss: 0.3492 - val_recall: 0.4233
Epoch 72/100
200/200 [==============================] - 5s 25ms/step - loss: 0.3104 - recall: 0.51
76 - val_loss: 0.3506 - val_recall: 0.4479
Epoch 73/100
200/200 [==============================] - 6s 29ms/step - loss: 0.3157 - recall: 0.50
69 - val_loss: 0.3459 - val_recall: 0.4540
Epoch 74/100
200/200 [==============================] - 4s 20ms/step - loss: 0.3091 - recall: 0.51
76 - val_loss: 0.3514 - val_recall: 0.4571
Epoch 75/100
200/200 [==============================] - 2s 10ms/step - loss: 0.3107 - recall: 0.51
84 - val_loss: 0.3520 - val_recall: 0.3926
Epoch 76/100
200/200 [==============================] - 3s 15ms/step - loss: 0.3080 - recall: 0.49
77 - val_loss: 0.3474 - val_recall: 0.4325
Epoch 77/100
200/200 [==============================] - 5s 25ms/step - loss: 0.3028 - recall: 0.52
15 - val_loss: 0.3525 - val_recall: 0.4264
Epoch 78/100
200/200 [==============================] - 4s 20ms/step - loss: 0.3058 - recall: 0.51
23 - val_loss: 0.3480 - val_recall: 0.4049
Epoch 79/100
200/200 [==============================] - 5s 25ms/step - loss: 0.3087 - recall: 0.50
61 - val_loss: 0.3500 - val_recall: 0.4294
Epoch 80/100
200/200 [==============================] - 4s 20ms/step - loss: 0.3106 - recall: 0.51
07 - val_loss: 0.3533 - val_recall: 0.4141
```

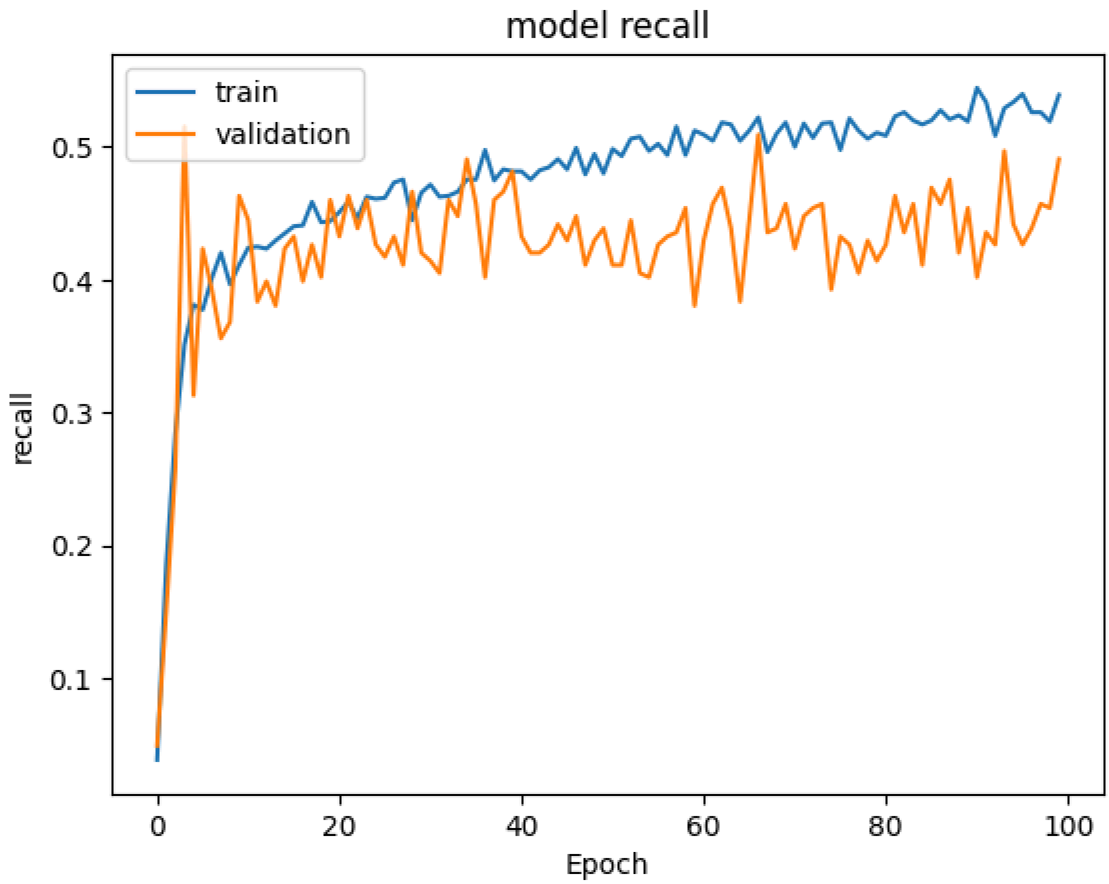
```
Epoch 81/100
200/200 [==============================] - 4s 20ms/step - loss: 0.3099 - recall: 0.50
84 - val_loss: 0.3506 - val_recall: 0.4264
Epoch 82/100
200/200 [==============================] - 4s 22ms/step - loss: 0.3058 - recall: 0.52
30 - val_loss: 0.3508 - val_recall: 0.4632
Epoch 83/100
200/200 [==============================] - 5s 24ms/step - loss: 0.3014 - recall: 0.52
61 - val_loss: 0.3500 - val_recall: 0.4356
Epoch 84/100
200/200 [==============================] - 5s 26ms/step - loss: 0.3042 - recall: 0.51
99 - val_loss: 0.3606 - val_recall: 0.4571
Epoch 85/100
200/200 [==============================] - 5s 23ms/step - loss: 0.3064 - recall: 0.51
69 - val_loss: 0.3484 - val_recall: 0.4110
Epoch 86/100
200/200 [==============================] - 6s 31ms/step - loss: 0.3037 - recall: 0.51
99 - val_loss: 0.3559 - val_recall: 0.4693
Epoch 87/100
200/200 [==============================] - 3s 15ms/step - loss: 0.3042 - recall: 0.52
76 - val_loss: 0.3571 - val_recall: 0.4571
Epoch 88/100
200/200 [==============================] - 4s 20ms/step - loss: 0.3082 - recall: 0.52
07 - val_loss: 0.3503 - val_recall: 0.4755
Epoch 89/100
200/200 [==============================] - 3s 17ms/step - loss: 0.3091 - recall: 0.52
38 - val_loss: 0.3503 - val_recall: 0.4202
Epoch 90/100
200/200 [==============================] - 4s 19ms/step - loss: 0.3051 - recall: 0.51
92 - val_loss: 0.3510 - val_recall: 0.4540
Epoch 91/100
200/200 [==============================] - 5s 23ms/step - loss: 0.3009 - recall: 0.54
45 - val_loss: 0.3577 - val_recall: 0.4018
Epoch 92/100
200/200 [==============================] - 5s 23ms/step - loss: 0.3048 - recall: 0.53
37 - val_loss: 0.3493 - val_recall: 0.4356
Epoch 93/100
200/200 [==============================] - 4s 22ms/step - loss: 0.3062 - recall: 0.50
84 - val_loss: 0.3518 - val_recall: 0.4264
Epoch 94/100
200/200 [==============================] - 3s 17ms/step - loss: 0.3051 - recall: 0.52
91 - val_loss: 0.3504 - val_recall: 0.4969
Epoch 95/100
200/200 [==============================] - 4s 22ms/step - loss: 0.3055 - recall: 0.53
37 - val_loss: 0.3534 - val_recall: 0.4417
Epoch 96/100
200/200 [==============================] - 6s 29ms/step - loss: 0.3029 - recall: 0.53
99 - val_loss: 0.3564 - val_recall: 0.4264
Epoch 97/100
200/200 [==============================] - 5s 22ms/step - loss: 0.3042 - recall: 0.52
61 - val_loss: 0.3553 - val_recall: 0.4387
Epoch 98/100
200/200 [==============================] - 5s 24ms/step - loss: 0.3019 - recall: 0.52
61 - val_loss: 0.3575 - val_recall: 0.4571
Epoch 99/100
200/200 [==============================] - 5s 24ms/step - loss: 0.3061 - recall: 0.51
92 - val_loss: 0.3548 - val_recall: 0.4540
Epoch 100/100
200/200 [==============================] - 5s 27ms/step - loss: 0.2994 - recall: 0.53
91 - val_loss: 0.3586 - val_recall: 0.4908
```

Loss function

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



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



In [ ]:
```
#Predicting the results using best as a threshold
y_train_pred = model_2.predict(X_train)
y_train_pred = (y_train_pred > 0.5)
y_train_pred
```

```
200/200 [==============================] - 4s 10ms/step
```

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

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

```
50/50 [==============================] - 1s 7ms/step
```

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

In [ ]:
```
model_name = "NN with Adam & Dropout"
```

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

Classification report

In [ ]:
```
#classification report
cr=classification_report(y_train,y_train_pred)
print(cr)
```

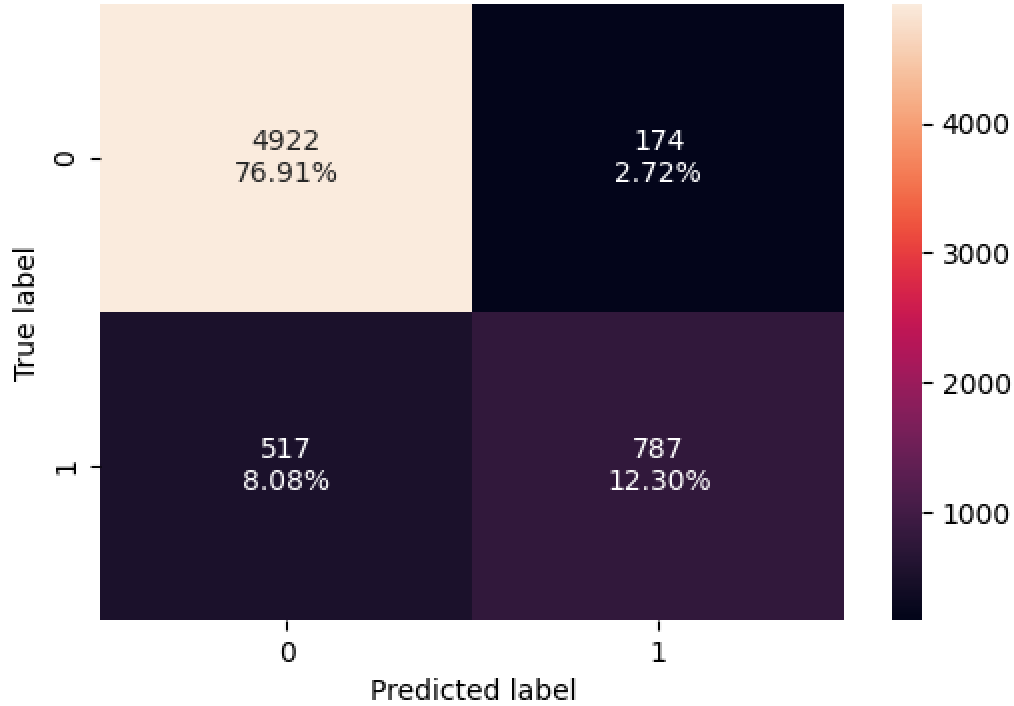
```
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      5096
           1       0.82      0.60      0.69      1304

    accuracy                           0.89      6400
   macro avg       0.86      0.78      0.81      6400
weighted avg       0.89      0.89      0.89      6400
```

In [ ]:
```
#classification report
cr = classification_report(y_val,y_val_pred)
print(cr)
```

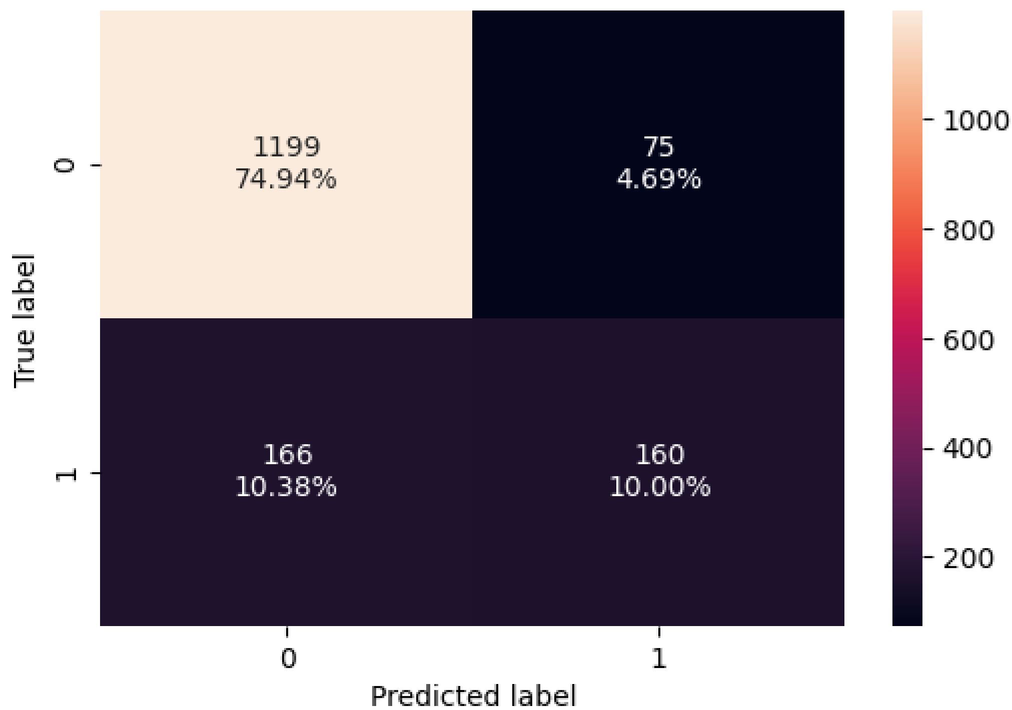
```
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      1274
           1       0.68      0.49      0.57       326

    accuracy                           0.85      1600
   macro avg       0.78      0.72      0.74      1600
weighted avg       0.84      0.85      0.84      1600
```

Confusion matrix

In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_train, y_train_pred)
```



In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)
```



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

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

In [ ]:
```
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,)
```

Let's build a model with the balanced dataset

In [ ]:
```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the sam
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [ ]:
```
#Initializing the model
model_3 = Sequential()
model_3.add(Dense(32,activation='relu',input_dim = X_train_smote.shape[1]))
model_3.add(Dense(16,activation='relu'))
model_3.add(Dense(8,activation='relu'))
model_3.add(Dense(1, activation = 'sigmoid'))
```

In [ ]:
```
# SGD
optimizer = tf.keras.optimizers.SGD(0.001)
metric = keras.metrics.Recall()
```

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

In [ ]:
```
model_3.summary()
```

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

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 8)                 136

 dense_3 (Dense)             (None, 1)                 9

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

In [ ]:
```
#Fitting the ANN
history_3 = model_3.fit(
    X_train_smote, y_train_smote,
```

```
    batch_size=32,
    epochs=50,
    verbose=1,
    validation_data = (X_val,y_val)
)
```

```
Epoch 1/50
319/319 [==============================] - 15s 37ms/step - loss: 0.6995 - recall: 0.7
983 - val_loss: 0.7144 - val_recall: 0.8067
Epoch 2/50
319/319 [==============================] - 5s 16ms/step - loss: 0.6961 - recall: 0.78
55 - val_loss: 0.7066 - val_recall: 0.7822
Epoch 3/50
319/319 [==============================] - 7s 23ms/step - loss: 0.6927 - recall: 0.76
28 - val_loss: 0.6988 - val_recall: 0.7362
Epoch 4/50
319/319 [==============================] - 6s 20ms/step - loss: 0.6892 - recall: 0.72
23 - val_loss: 0.6909 - val_recall: 0.6902
Epoch 5/50
319/319 [==============================] - 5s 17ms/step - loss: 0.6854 - recall: 0.68
11 - val_loss: 0.6830 - val_recall: 0.6626
Epoch 6/50
319/319 [==============================] - 7s 21ms/step - loss: 0.6816 - recall: 0.65
99 - val_loss: 0.6755 - val_recall: 0.6472
Epoch 7/50
319/319 [==============================] - 3s 10ms/step - loss: 0.6779 - recall: 0.63
48 - val_loss: 0.6688 - val_recall: 0.6258
Epoch 8/50
319/319 [==============================] - 4s 11ms/step - loss: 0.6744 - recall: 0.63
07 - val_loss: 0.6627 - val_recall: 0.6227
Epoch 9/50
319/319 [==============================] - 3s 9ms/step - loss: 0.6710 - recall: 0.627
9 - val_loss: 0.6570 - val_recall: 0.6135
Epoch 10/50
319/319 [==============================] - 3s 8ms/step - loss: 0.6677 - recall: 0.630
5 - val_loss: 0.6518 - val_recall: 0.6104
Epoch 11/50
319/319 [==============================] - 4s 11ms/step - loss: 0.6644 - recall: 0.63
15 - val_loss: 0.6469 - val_recall: 0.6043
Epoch 12/50
319/319 [==============================] - 4s 12ms/step - loss: 0.6611 - recall: 0.63
91 - val_loss: 0.6419 - val_recall: 0.5920
Epoch 13/50
319/319 [==============================] - 3s 10ms/step - loss: 0.6578 - recall: 0.63
36 - val_loss: 0.6373 - val_recall: 0.5890
Epoch 14/50
319/319 [==============================] - 4s 14ms/step - loss: 0.6544 - recall: 0.64
15 - val_loss: 0.6327 - val_recall: 0.5951
Epoch 15/50
319/319 [==============================] - 2s 7ms/step - loss: 0.6509 - recall: 0.641
9 - val_loss: 0.6283 - val_recall: 0.5982
Epoch 16/50
319/319 [==============================] - 5s 14ms/step - loss: 0.6474 - recall: 0.64
68 - val_loss: 0.6241 - val_recall: 0.6074
Epoch 17/50
319/319 [==============================] - 5s 14ms/step - loss: 0.6439 - recall: 0.65
31 - val_loss: 0.6200 - val_recall: 0.6074
Epoch 18/50
319/319 [==============================] - 3s 10ms/step - loss: 0.6402 - recall: 0.65
99 - val_loss: 0.6159 - val_recall: 0.6104
Epoch 19/50
319/319 [==============================] - 4s 13ms/step - loss: 0.6365 - recall: 0.66
35 - val_loss: 0.6119 - val_recall: 0.6104
Epoch 20/50
319/319 [==============================] - 4s 13ms/step - loss: 0.6328 - recall: 0.66
44 - val_loss: 0.6083 - val_recall: 0.6104
```

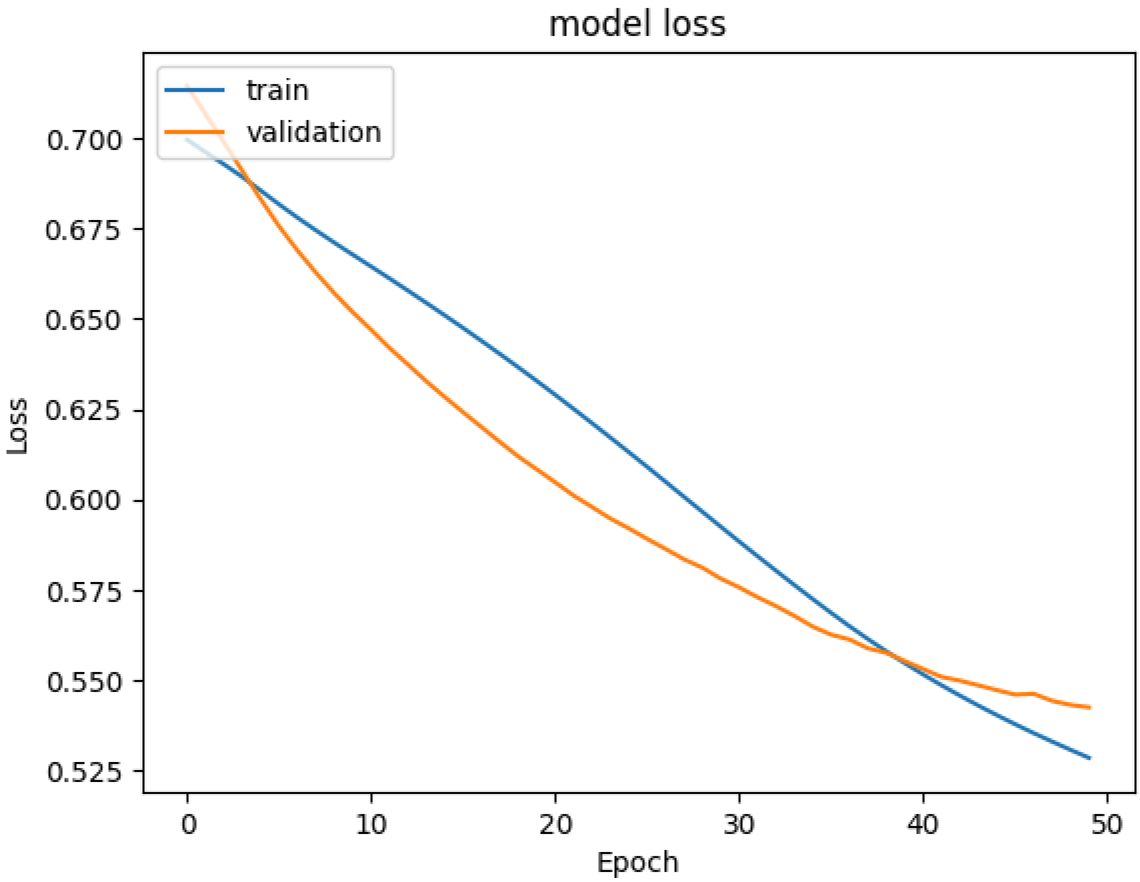
```
Epoch 21/50
319/319 [==============================] - 5s 17ms/step - loss: 0.6290 - recall: 0.67
45 - val_loss: 0.6048 - val_recall: 0.6074
Epoch 22/50
319/319 [==============================] - 3s 9ms/step - loss: 0.6251 - recall: 0.682
9 - val_loss: 0.6011 - val_recall: 0.6074
Epoch 23/50
319/319 [==============================] - 3s 8ms/step - loss: 0.6211 - recall: 0.684
9 - val_loss: 0.5979 - val_recall: 0.6074
Epoch 24/50
319/319 [==============================] - 3s 8ms/step - loss: 0.6171 - recall: 0.691
3 - val_loss: 0.5947 - val_recall: 0.6104
Epoch 25/50
319/319 [==============================] - 3s 11ms/step - loss: 0.6131 - recall: 0.69
31 - val_loss: 0.5920 - val_recall: 0.6135
Epoch 26/50
319/319 [==============================] - 3s 9ms/step - loss: 0.6090 - recall: 0.698
0 - val_loss: 0.5891 - val_recall: 0.6227
Epoch 27/50
319/319 [==============================] - 3s 9ms/step - loss: 0.6049 - recall: 0.701
9 - val_loss: 0.5863 - val_recall: 0.6319
Epoch 28/50
319/319 [==============================] - 2s 6ms/step - loss: 0.6007 - recall: 0.708
2 - val_loss: 0.5835 - val_recall: 0.6350
Epoch 29/50
319/319 [==============================] - 2s 7ms/step - loss: 0.5966 - recall: 0.711
3 - val_loss: 0.5812 - val_recall: 0.6472
Epoch 30/50
319/319 [==============================] - 3s 11ms/step - loss: 0.5925 - recall: 0.72
17 - val_loss: 0.5781 - val_recall: 0.6472
Epoch 31/50
319/319 [==============================] - 4s 12ms/step - loss: 0.5884 - recall: 0.72
37 - val_loss: 0.5757 - val_recall: 0.6595
Epoch 32/50
319/319 [==============================] - 3s 10ms/step - loss: 0.5843 - recall: 0.72
80 - val_loss: 0.5730 - val_recall: 0.6626
Epoch 33/50
319/319 [==============================] - 4s 11ms/step - loss: 0.5803 - recall: 0.73
10 - val_loss: 0.5705 - val_recall: 0.6748
Epoch 34/50
319/319 [==============================] - 4s 12ms/step - loss: 0.5763 - recall: 0.73
51 - val_loss: 0.5678 - val_recall: 0.6748
Epoch 35/50
319/319 [==============================] - 4s 11ms/step - loss: 0.5724 - recall: 0.73
84 - val_loss: 0.5648 - val_recall: 0.6779
Epoch 36/50
319/319 [==============================] - 4s 12ms/step - loss: 0.5686 - recall: 0.73
70 - val_loss: 0.5626 - val_recall: 0.6810
Epoch 37/50
319/319 [==============================] - 3s 10ms/step - loss: 0.5649 - recall: 0.73
98 - val_loss: 0.5613 - val_recall: 0.6840
Epoch 38/50
319/319 [==============================] - 4s 12ms/step - loss: 0.5614 - recall: 0.74
57 - val_loss: 0.5588 - val_recall: 0.6871
Epoch 39/50
319/319 [==============================] - 3s 10ms/step - loss: 0.5580 - recall: 0.74
43 - val_loss: 0.5576 - val_recall: 0.6902
Epoch 40/50
319/319 [==============================] - 3s 10ms/step - loss: 0.5547 - recall: 0.74
90 - val_loss: 0.5553 - val_recall: 0.6902
```

```
Epoch 41/50
319/319 [==============================] - 3s 10ms/step - loss: 0.5516 - recall: 0.75
24 - val_loss: 0.5531 - val_recall: 0.6871
Epoch 42/50
319/319 [==============================] - 3s 10ms/step - loss: 0.5486 - recall: 0.75
29 - val_loss: 0.5509 - val_recall: 0.6902
Epoch 43/50
319/319 [==============================] - 3s 10ms/step - loss: 0.5458 - recall: 0.75
27 - val_loss: 0.5499 - val_recall: 0.6994
Epoch 44/50
319/319 [==============================] - 3s 9ms/step - loss: 0.5430 - recall: 0.754
9 - val_loss: 0.5486 - val_recall: 0.7025
Epoch 45/50
319/319 [==============================] - 2s 7ms/step - loss: 0.5404 - recall: 0.755
7 - val_loss: 0.5473 - val_recall: 0.7086
Epoch 46/50
319/319 [==============================] - 4s 12ms/step - loss: 0.5379 - recall: 0.76
08 - val_loss: 0.5461 - val_recall: 0.7086
Epoch 47/50
319/319 [==============================] - 4s 13ms/step - loss: 0.5354 - recall: 0.75
84 - val_loss: 0.5463 - val_recall: 0.7147
Epoch 48/50
319/319 [==============================] - 3s 9ms/step - loss: 0.5331 - recall: 0.763
1 - val_loss: 0.5443 - val_recall: 0.7147
Epoch 49/50
319/319 [==============================] - 4s 13ms/step - loss: 0.5308 - recall: 0.76
28 - val_loss: 0.5432 - val_recall: 0.7147
Epoch 50/50
319/319 [==============================] - 4s 12ms/step - loss: 0.5286 - recall: 0.76
45 - val_loss: 0.5425 - val_recall: 0.7147
```

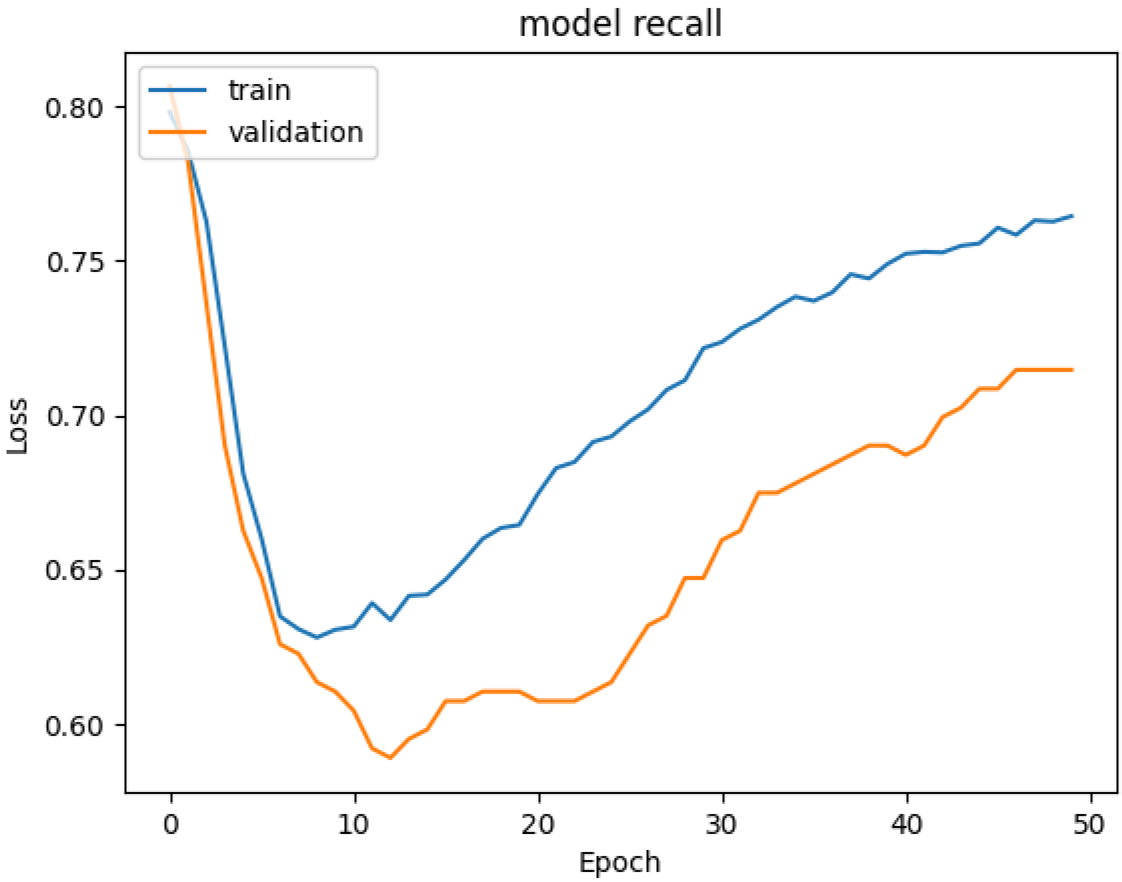
Loss function

In [ ]:

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



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



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

Out[ ]:
```
319/319 [==============================] - 4s 11ms/step
array([[ True],
       [False],
       [False],
       ...,
       [ True],
       [ True],
       [ True]])
```

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

Out[ ]:
```
50/50 [==============================] - 1s 14ms/step
array([[ True],
       [False],
       [False],
       ...,
       [False],
       [ True],
       [ True]])
```

In [ ]:
```
model_name = "NN with SMOTE & SGD"
```

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

Classification report

In [ ]:
```
cr=classification_report(y_train_smote,y_train_pred)
print(cr)
```

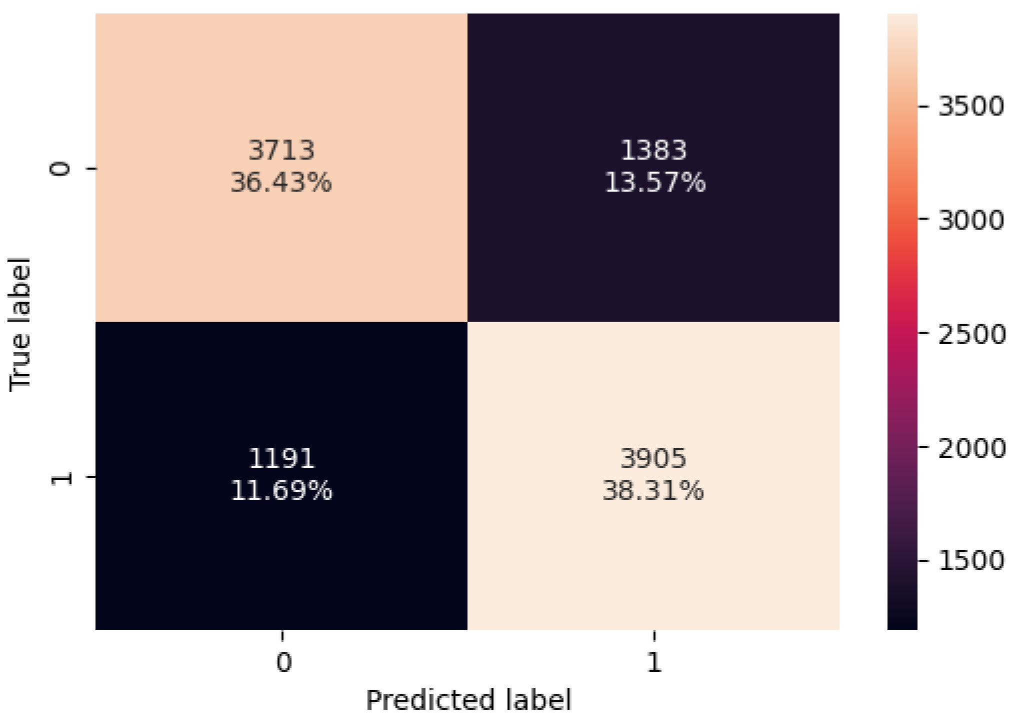
```
              precision    recall  f1-score   support

           0       0.76      0.73      0.74      5096
           1       0.74      0.77      0.75      5096

    accuracy                           0.75     10192
   macro avg       0.75      0.75      0.75     10192
weighted avg       0.75      0.75      0.75     10192
```

In [ ]:
```
cr=classification_report(y_val,y_val_pred)
print(cr)
```

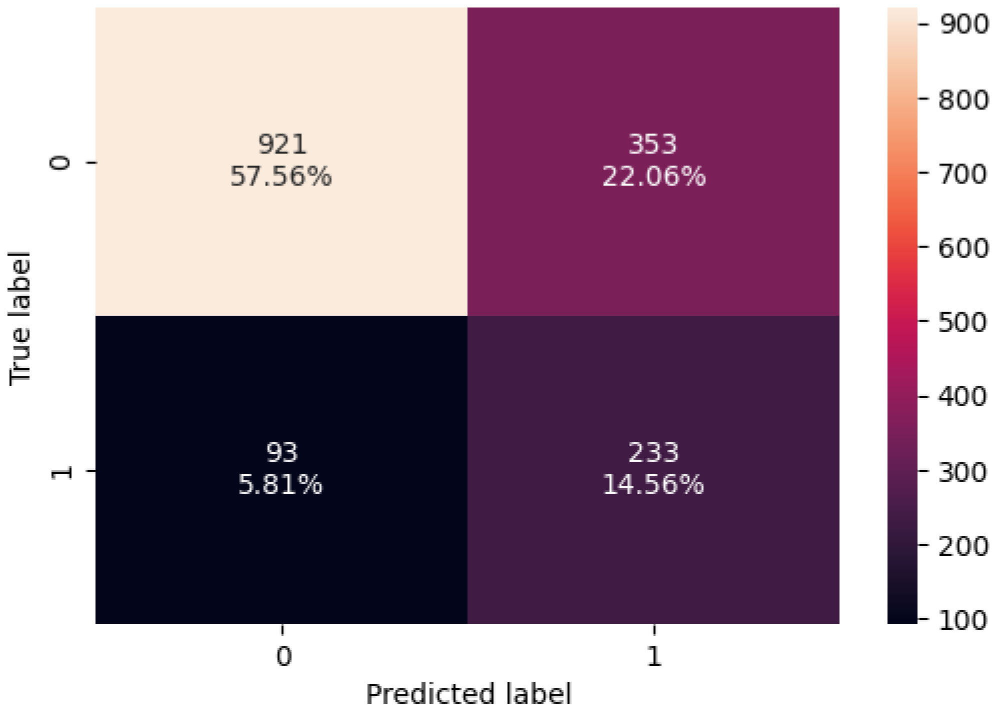
```
              precision    recall  f1-score   support

           0       0.91      0.72      0.81      1274
           1       0.40      0.71      0.51       326

    accuracy                           0.72      1600
   macro avg       0.65      0.72      0.66      1600
weighted avg       0.80      0.72      0.75      1600
```

Confusion matrix

In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_train_smote, y_train_pred)
```



In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)
```



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

Let's build a model with the balanced dataset

In [ ]:
```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the sam
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [ ]:
```
#Initializing the model
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 [ ]:
```
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

 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 [ ]:
```
# Adam
optimizer = tf.keras.optimizers.Adam()
metric = keras.metrics.Recall()
```

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

In [ ]:
```
model_4.summary()
```

```
Model: "sequential"

 Layer (type)                Output Shape              Param #
=================================================================

 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 [ ]:
```
#Fitting the ANN
history_4 = model_4.fit(
    X_train_smote,y_train_smote,
    batch_size=32,
```

```
    epochs=50,
    verbose=1,
    validation_data = (X_val,y_val)
)
```

```
Epoch 1/50
319/319 [==============================] - 3s 4ms/step - loss: 0.5139 - recall: 0.754
9 - val_loss: 0.5364 - val_recall: 0.8006
Epoch 2/50
319/319 [==============================] - 1s 3ms/step - loss: 0.4405 - recall: 0.794
2 - val_loss: 0.4504 - val_recall: 0.7117
Epoch 3/50
319/319 [==============================] - 1s 3ms/step - loss: 0.4239 - recall: 0.795
7 - val_loss: 0.4842 - val_recall: 0.7515
Epoch 4/50
319/319 [==============================] - 1s 4ms/step - loss: 0.4131 - recall: 0.799
5 - val_loss: 0.4363 - val_recall: 0.6810
Epoch 5/50
319/319 [==============================] - 1s 3ms/step - loss: 0.4048 - recall: 0.802
0 - val_loss: 0.5027 - val_recall: 0.7638
Epoch 6/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3975 - recall: 0.813
8 - val_loss: 0.4654 - val_recall: 0.7025
Epoch 7/50
319/319 [==============================] - 1s 5ms/step - loss: 0.3911 - recall: 0.817
5 - val_loss: 0.4358 - val_recall: 0.6687
Epoch 8/50
319/319 [==============================] - 1s 4ms/step - loss: 0.3833 - recall: 0.818
7 - val_loss: 0.4289 - val_recall: 0.6718
Epoch 9/50
319/319 [==============================] - 1s 4ms/step - loss: 0.3766 - recall: 0.830
8 - val_loss: 0.4211 - val_recall: 0.6595
Epoch 10/50
319/319 [==============================] - 1s 4ms/step - loss: 0.3736 - recall: 0.828
7 - val_loss: 0.4278 - val_recall: 0.6595
Epoch 11/50
319/319 [==============================] - 1s 4ms/step - loss: 0.3686 - recall: 0.827
9 - val_loss: 0.4478 - val_recall: 0.6963
Epoch 12/50
319/319 [==============================] - 1s 4ms/step - loss: 0.3618 - recall: 0.833
8 - val_loss: 0.4205 - val_recall: 0.6166
Epoch 13/50
319/319 [==============================] - 2s 5ms/step - loss: 0.3554 - recall: 0.848
3 - val_loss: 0.4727 - val_recall: 0.7331
Epoch 14/50
319/319 [==============================] - 1s 4ms/step - loss: 0.3505 - recall: 0.849
1 - val_loss: 0.4727 - val_recall: 0.6933
Epoch 15/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3448 - recall: 0.851
8 - val_loss: 0.4510 - val_recall: 0.6810
Epoch 16/50
319/319 [==============================] - 1s 4ms/step - loss: 0.3381 - recall: 0.852
6 - val_loss: 0.4627 - val_recall: 0.7147
Epoch 17/50
319/319 [==============================] - 1s 4ms/step - loss: 0.3364 - recall: 0.856
4 - val_loss: 0.5060 - val_recall: 0.7362
Epoch 18/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3301 - recall: 0.862
0 - val_loss: 0.4618 - val_recall: 0.6779
Epoch 19/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3258 - recall: 0.862
0 - val_loss: 0.4359 - val_recall: 0.6043
Epoch 20/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3200 - recall: 0.868
3 - val_loss: 0.4780 - val_recall: 0.6994
```

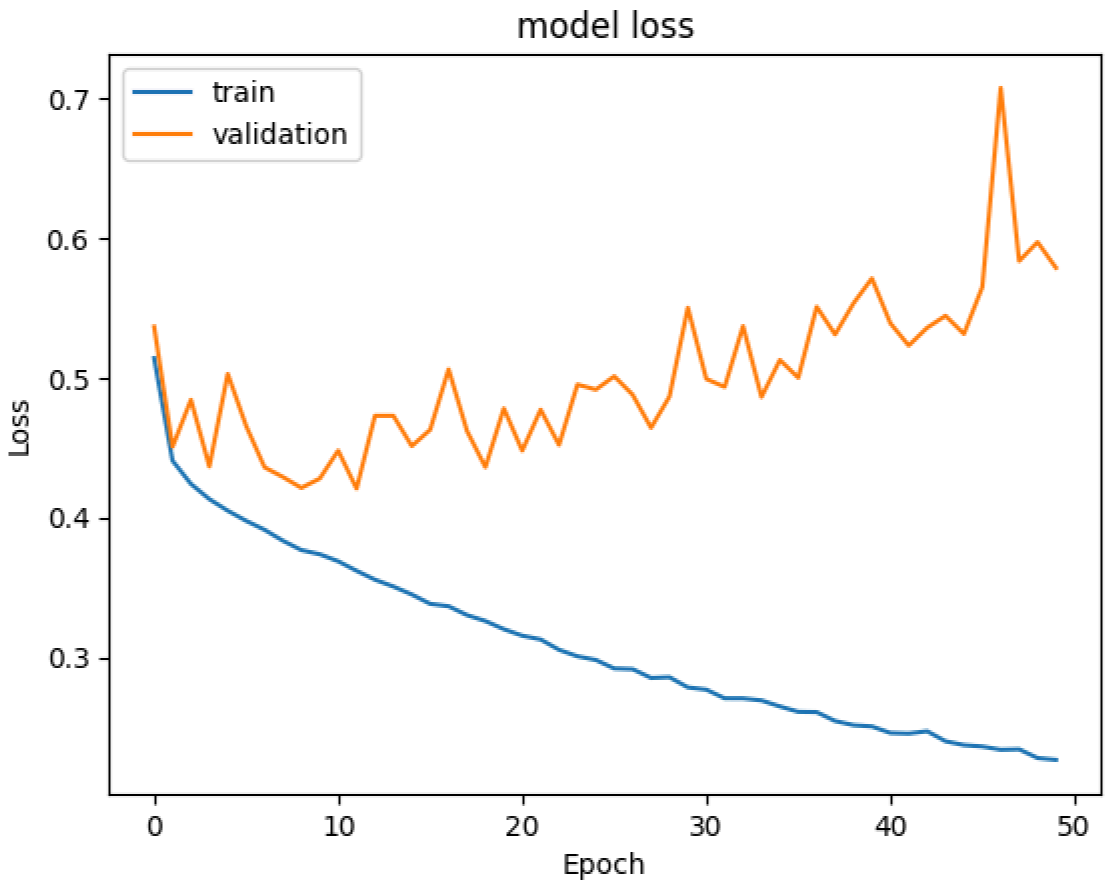
```
Epoch 21/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3154 - recall: 0.869
9 - val_loss: 0.4477 - val_recall: 0.5798
Epoch 22/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3127 - recall: 0.872
1 - val_loss: 0.4771 - val_recall: 0.6902
Epoch 23/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3053 - recall: 0.879
7 - val_loss: 0.4518 - val_recall: 0.6196
Epoch 24/50
319/319 [==============================] - 1s 3ms/step - loss: 0.3006 - recall: 0.880
1 - val_loss: 0.4950 - val_recall: 0.6718
Epoch 25/50
319/319 [==============================] - 1s 4ms/step - loss: 0.2980 - recall: 0.885
4 - val_loss: 0.4914 - val_recall: 0.6442
Epoch 26/50
319/319 [==============================] - 1s 4ms/step - loss: 0.2919 - recall: 0.888
1 - val_loss: 0.5009 - val_recall: 0.6595
Epoch 27/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2914 - recall: 0.885
0 - val_loss: 0.4877 - val_recall: 0.6718
Epoch 28/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2851 - recall: 0.889
3 - val_loss: 0.4641 - val_recall: 0.5767
Epoch 29/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2857 - recall: 0.889
7 - val_loss: 0.4864 - val_recall: 0.6380
Epoch 30/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2784 - recall: 0.897
4 - val_loss: 0.5500 - val_recall: 0.6933
Epoch 31/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2768 - recall: 0.896
0 - val_loss: 0.4990 - val_recall: 0.6442
Epoch 32/50
319/319 [==============================] - 1s 4ms/step - loss: 0.2705 - recall: 0.902
1 - val_loss: 0.4933 - val_recall: 0.5644
Epoch 33/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2705 - recall: 0.899
5 - val_loss: 0.5369 - val_recall: 0.6380
Epoch 34/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2690 - recall: 0.901
7 - val_loss: 0.4861 - val_recall: 0.5828
Epoch 35/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2647 - recall: 0.902
7 - val_loss: 0.5126 - val_recall: 0.6196
Epoch 36/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2609 - recall: 0.900
5 - val_loss: 0.4999 - val_recall: 0.6166
Epoch 37/50
319/319 [==============================] - 2s 5ms/step - loss: 0.2607 - recall: 0.902
5 - val_loss: 0.5507 - val_recall: 0.6871
Epoch 38/50
319/319 [==============================] - 1s 4ms/step - loss: 0.2543 - recall: 0.910
3 - val_loss: 0.5308 - val_recall: 0.6196
Epoch 39/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2514 - recall: 0.908
4 - val_loss: 0.5532 - val_recall: 0.6656
Epoch 40/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2504 - recall: 0.911
7 - val_loss: 0.5710 - val_recall: 0.6595
```

```
Epoch 41/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2458 - recall: 0.912
7 - val_loss: 0.5389 - val_recall: 0.6319
Epoch 42/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2454 - recall: 0.907
4 - val_loss: 0.5229 - val_recall: 0.5368
Epoch 43/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2470 - recall: 0.912
1 - val_loss: 0.5356 - val_recall: 0.6135
Epoch 44/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2398 - recall: 0.916
0 - val_loss: 0.5442 - val_recall: 0.5828
Epoch 45/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2371 - recall: 0.916
8 - val_loss: 0.5312 - val_recall: 0.5890
Epoch 46/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2361 - recall: 0.916
8 - val_loss: 0.5648 - val_recall: 0.6472
Epoch 47/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2338 - recall: 0.919
3 - val_loss: 0.7073 - val_recall: 0.7117
Epoch 48/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2341 - recall: 0.917
8 - val_loss: 0.5834 - val_recall: 0.6411
Epoch 49/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2278 - recall: 0.921
1 - val_loss: 0.5970 - val_recall: 0.6319
Epoch 50/50
319/319 [==============================] - 1s 3ms/step - loss: 0.2265 - recall: 0.922
3 - val_loss: 0.5785 - val_recall: 0.6043
```

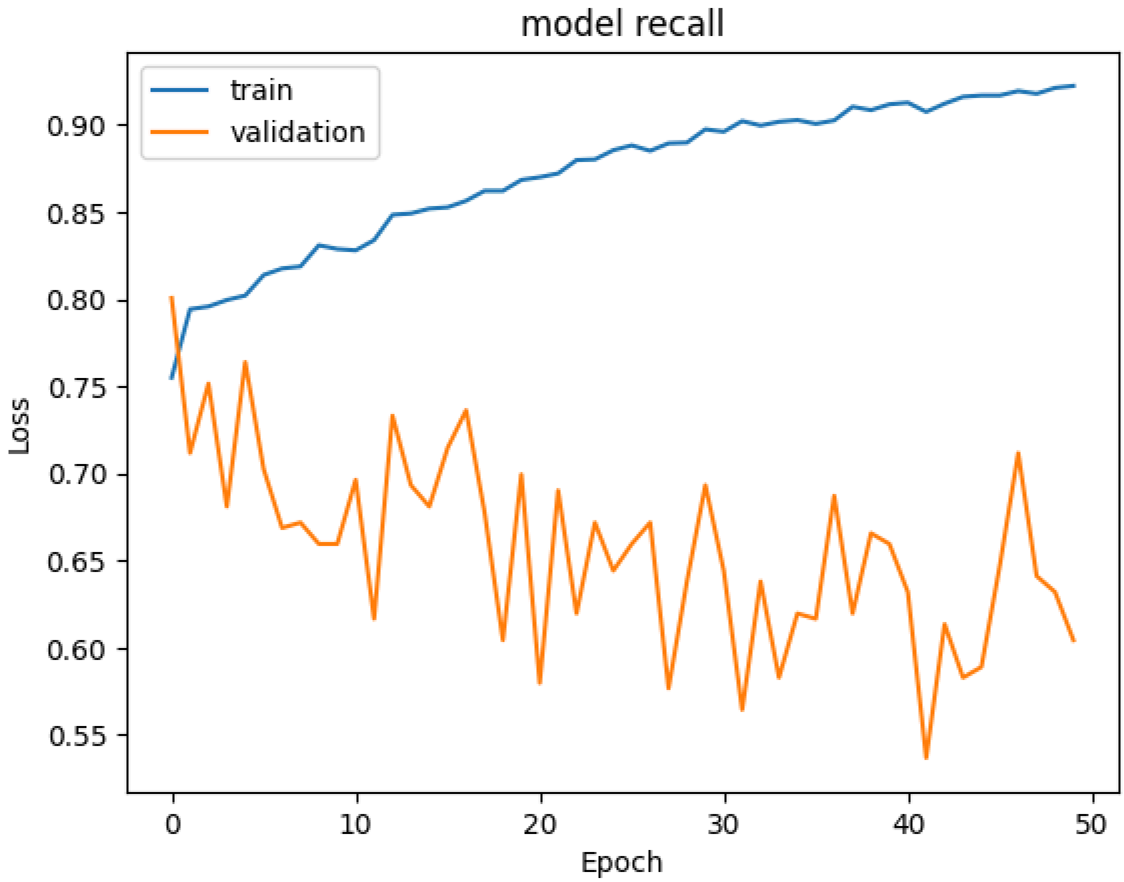
Loss function

In [ ]:

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



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



In [ ]:
```python
y_train_pred = model_4.predict(X_train_smote)
#Predicting the results using 0.5 as the threshold
y_train_pred = (y_train_pred > 0.5)
y_train_pred
```

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

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

In [ ]:
```python
y_val_pred = model_4.predict(X_val)
#Predicting the results using 0.5 as the threshold
y_val_pred = (y_val_pred > 0.5)
y_val_pred
```

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

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

In [ ]:
```python
model_name = "NN with SMOTE & Adam"
```

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

Classification report

In [ ]:
```
cr=classification_report(y_train_smote,y_train_pred)
print(cr)
```

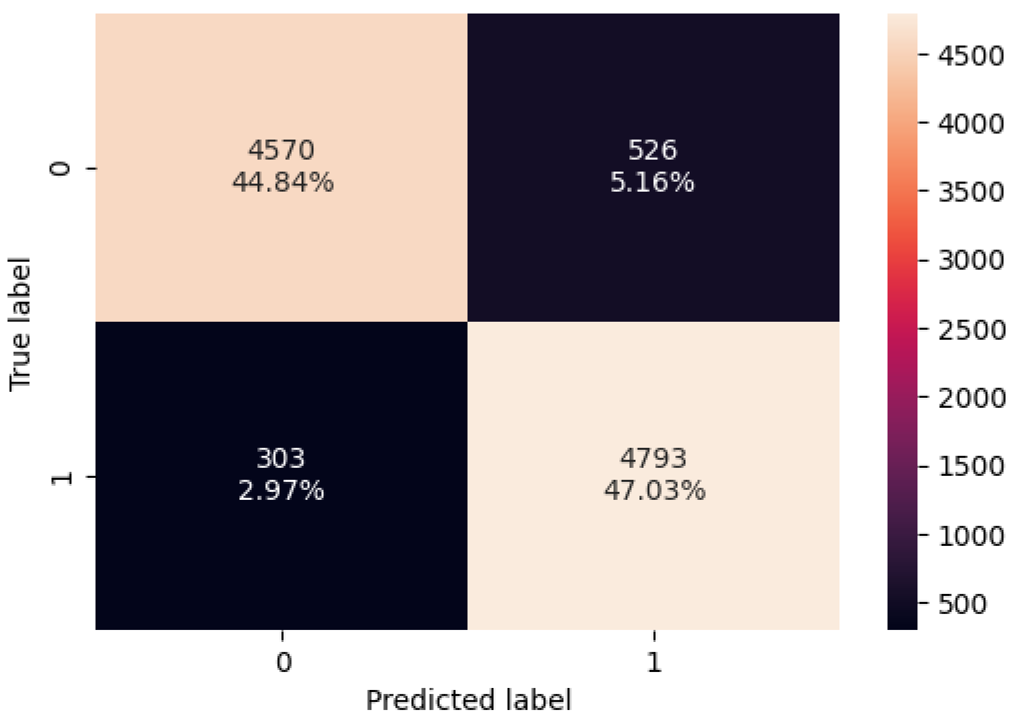
```
              precision    recall  f1-score   support

           0       0.94      0.90      0.92      5096
           1       0.90      0.94      0.92      5096

    accuracy                           0.92     10192
   macro avg       0.92      0.92      0.92     10192
weighted avg       0.92      0.92      0.92     10192
```

In [ ]:
```
cr=classification_report(y_val,y_val_pred)
print(cr)
```

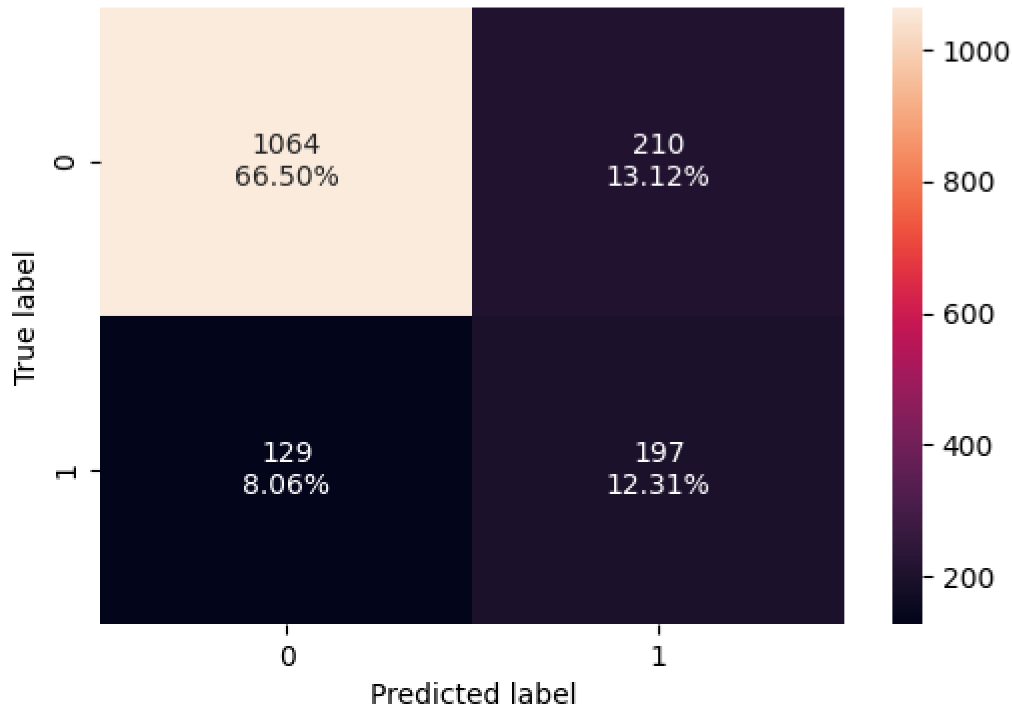
```
              precision    recall  f1-score   support

           0       0.89      0.84      0.86      1274
           1       0.48      0.60      0.54       326

    accuracy                           0.79      1600
   macro avg       0.69      0.72      0.70      1600
weighted avg       0.81      0.79      0.80      1600
```

Confusion matrix

In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_train_smote, y_train_pred)
```



In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)
```



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

In [ ]:
```
backend.clear_session()
#Fixing the seed for random number generators so that we can ensure we receive the sam
np.random.seed(2)
random.seed(2)
tf.random.set_seed(2)
```

In [ ]:
```
# Initializing the model
model_5 = Sequential()
# Adding the input layer
model_5.add(Dense(64, activation='relu', input_dim=X_train_smote.shape[1]))
# Adding dropout regularization
model_5.add(Dropout(0.5))  # Dropout rate of 0.5 (50% dropout)
# Adding a hidden layer
model_5.add(Dense(32, activation='relu'))
# Adding dropout regularization
model_5.add(Dropout(0.3))  # Dropout rate of 0.3 (30% dropout)
# Adding another hidden layer
model_5.add(Dense(8, activation='relu'))
# Adding the output layer
model_5.add(Dense(1, activation='sigmoid'))
```

In [ ]:
```
# Adam
optimizer = tf.keras.optimizers.Adam()
metric = keras.metrics.Recall()
```

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

In [ ]:
```
model_5.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
_________________________________________________________________
 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)
_________________________________________________________________
```

In [ ]:
```
history_5 = model_5.fit(
    X_train_smote,y_train_smote,
    batch_size=32,
    epochs=50,
    verbose=1,
    validation_data = (X_val,y_val))
```

```
Epoch 1/50
319/319 [==============================] - 7s 10ms/step - loss: 0.6190 - recall: 0.67
82 - val_loss: 0.5435 - val_recall: 0.7025
Epoch 2/50
319/319 [==============================] - 1s 3ms/step - loss: 0.5411 - recall: 0.735
3 - val_loss: 0.4899 - val_recall: 0.7117
Epoch 3/50
319/319 [==============================] - 5s 16ms/step - loss: 0.5112 - recall: 0.75
75 - val_loss: 0.4611 - val_recall: 0.6933
Epoch 4/50
319/319 [==============================] - 3s 10ms/step - loss: 0.4919 - recall: 0.75
94 - val_loss: 0.4611 - val_recall: 0.6748
Epoch 5/50
319/319 [==============================] - 3s 9ms/step - loss: 0.4787 - recall: 0.753
1 - val_loss: 0.4586 - val_recall: 0.6779
Epoch 6/50
319/319 [==============================] - 3s 10ms/step - loss: 0.4756 - recall: 0.75
63 - val_loss: 0.4474 - val_recall: 0.6963
Epoch 7/50
319/319 [==============================] - 3s 11ms/step - loss: 0.4676 - recall: 0.76
10 - val_loss: 0.4448 - val_recall: 0.6748
Epoch 8/50
319/319 [==============================] - 4s 11ms/step - loss: 0.4661 - recall: 0.76
49 - val_loss: 0.4284 - val_recall: 0.6810
Epoch 9/50
319/319 [==============================] - 3s 11ms/step - loss: 0.4605 - recall: 0.77
30 - val_loss: 0.4577 - val_recall: 0.7147
Epoch 10/50
319/319 [==============================] - 3s 10ms/step - loss: 0.4560 - recall: 0.77
73 - val_loss: 0.4325 - val_recall: 0.6840
Epoch 11/50
319/319 [==============================] - 3s 10ms/step - loss: 0.4552 - recall: 0.77
71 - val_loss: 0.4415 - val_recall: 0.7025
Epoch 12/50
319/319 [==============================] - 3s 11ms/step - loss: 0.4483 - recall: 0.78
67 - val_loss: 0.4410 - val_recall: 0.6871
Epoch 13/50
319/319 [==============================] - 3s 11ms/step - loss: 0.4448 - recall: 0.77
86 - val_loss: 0.4358 - val_recall: 0.6810
Epoch 14/50
319/319 [==============================] - 4s 12ms/step - loss: 0.4447 - recall: 0.78
24 - val_loss: 0.4468 - val_recall: 0.7025
Epoch 15/50
319/319 [==============================] - 3s 11ms/step - loss: 0.4397 - recall: 0.77
41 - val_loss: 0.4504 - val_recall: 0.7025
Epoch 16/50
319/319 [==============================] - 4s 13ms/step - loss: 0.4415 - recall: 0.79
26 - val_loss: 0.4386 - val_recall: 0.6871
Epoch 17/50
319/319 [==============================] - 3s 10ms/step - loss: 0.4402 - recall: 0.79
10 - val_loss: 0.4432 - val_recall: 0.6994
Epoch 18/50
319/319 [==============================] - 6s 20ms/step - loss: 0.4388 - recall: 0.79
85 - val_loss: 0.4200 - val_recall: 0.6656
Epoch 19/50
319/319 [==============================] - 8s 24ms/step - loss: 0.4336 - recall: 0.79
22 - val_loss: 0.4345 - val_recall: 0.6994
Epoch 20/50
319/319 [==============================] - 8s 24ms/step - loss: 0.4347 - recall: 0.79
38 - val_loss: 0.4370 - val_recall: 0.6902
```

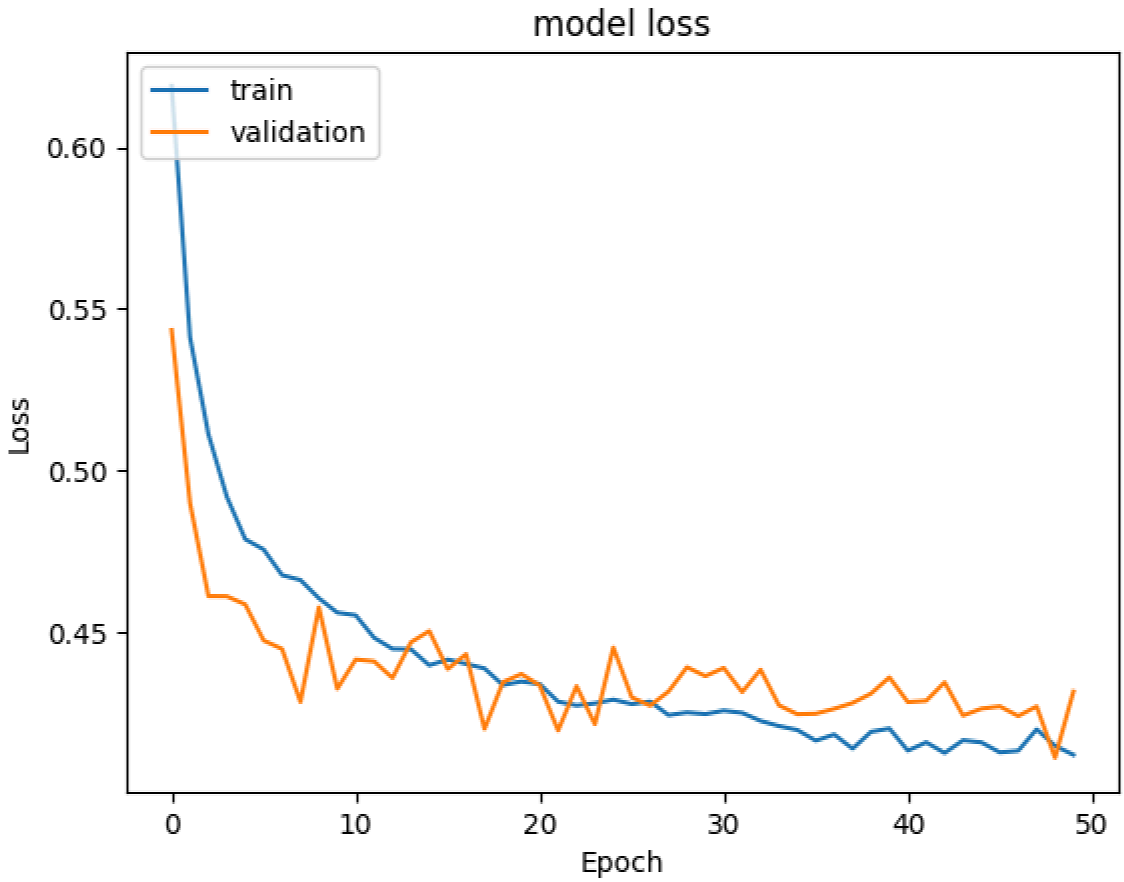
```
Epoch 21/50
319/319 [==============================] - 8s 25ms/step - loss: 0.4339 - recall: 0.79
67 - val_loss: 0.4335 - val_recall: 0.6963
Epoch 22/50
319/319 [==============================] - 8s 26ms/step - loss: 0.4285 - recall: 0.79
42 - val_loss: 0.4196 - val_recall: 0.6871
Epoch 23/50
319/319 [==============================] - 8s 25ms/step - loss: 0.4273 - recall: 0.80
00 - val_loss: 0.4333 - val_recall: 0.7025
Epoch 24/50
319/319 [==============================] - 8s 25ms/step - loss: 0.4280 - recall: 0.80
53 - val_loss: 0.4215 - val_recall: 0.6902
Epoch 25/50
319/319 [==============================] - 5s 16ms/step - loss: 0.4291 - recall: 0.80
47 - val_loss: 0.4453 - val_recall: 0.7147
Epoch 26/50
319/319 [==============================] - 8s 25ms/step - loss: 0.4278 - recall: 0.81
16 - val_loss: 0.4298 - val_recall: 0.7025
Epoch 27/50
319/319 [==============================] - 9s 27ms/step - loss: 0.4285 - recall: 0.80
57 - val_loss: 0.4273 - val_recall: 0.6963
Epoch 28/50
319/319 [==============================] - 6s 20ms/step - loss: 0.4243 - recall: 0.80
93 - val_loss: 0.4317 - val_recall: 0.7117
Epoch 29/50
319/319 [==============================] - 7s 23ms/step - loss: 0.4252 - recall: 0.80
87 - val_loss: 0.4392 - val_recall: 0.7025
Epoch 30/50
319/319 [==============================] - 7s 23ms/step - loss: 0.4246 - recall: 0.80
24 - val_loss: 0.4364 - val_recall: 0.7178
Epoch 31/50
319/319 [==============================] - 7s 22ms/step - loss: 0.4257 - recall: 0.80
71 - val_loss: 0.4389 - val_recall: 0.7117
Epoch 32/50
319/319 [==============================] - 3s 10ms/step - loss: 0.4250 - recall: 0.81
30 - val_loss: 0.4315 - val_recall: 0.7025
Epoch 33/50
319/319 [==============================] - 5s 15ms/step - loss: 0.4225 - recall: 0.79
85 - val_loss: 0.4384 - val_recall: 0.7086
Epoch 34/50
319/319 [==============================] - 6s 19ms/step - loss: 0.4209 - recall: 0.80
65 - val_loss: 0.4274 - val_recall: 0.6779
Epoch 35/50
319/319 [==============================] - 8s 25ms/step - loss: 0.4197 - recall: 0.80
59 - val_loss: 0.4246 - val_recall: 0.6902
Epoch 36/50
319/319 [==============================] - 8s 25ms/step - loss: 0.4164 - recall: 0.81
81 - val_loss: 0.4248 - val_recall: 0.6810
Epoch 37/50
319/319 [==============================] - 7s 22ms/step - loss: 0.4183 - recall: 0.80
73 - val_loss: 0.4263 - val_recall: 0.6871
Epoch 38/50
319/319 [==============================] - 9s 27ms/step - loss: 0.4139 - recall: 0.81
14 - val_loss: 0.4281 - val_recall: 0.6810
Epoch 39/50
319/319 [==============================] - 7s 23ms/step - loss: 0.4192 - recall: 0.81
28 - val_loss: 0.4310 - val_recall: 0.7117
Epoch 40/50
319/319 [==============================] - 7s 21ms/step - loss: 0.4202 - recall: 0.81
51 - val_loss: 0.4360 - val_recall: 0.6871
```

```
Epoch 41/50
319/319 [==============================] - 8s 25ms/step - loss: 0.4133 - recall: 0.81
28 - val_loss: 0.4284 - val_recall: 0.6902
Epoch 42/50
319/319 [==============================] - 6s 19ms/step - loss: 0.4159 - recall: 0.81
42 - val_loss: 0.4288 - val_recall: 0.6748
Epoch 43/50
319/319 [==============================] - 7s 23ms/step - loss: 0.4126 - recall: 0.80
79 - val_loss: 0.4345 - val_recall: 0.6933
Epoch 44/50
319/319 [==============================] - 8s 24ms/step - loss: 0.4166 - recall: 0.81
50 - val_loss: 0.4242 - val_recall: 0.6748
Epoch 45/50
319/319 [==============================] - 5s 17ms/step - loss: 0.4159 - recall: 0.81
57 - val_loss: 0.4263 - val_recall: 0.6810
Epoch 46/50
319/319 [==============================] - 8s 24ms/step - loss: 0.4128 - recall: 0.81
14 - val_loss: 0.4270 - val_recall: 0.6779
Epoch 47/50
319/319 [==============================] - 7s 22ms/step - loss: 0.4133 - recall: 0.81
55 - val_loss: 0.4240 - val_recall: 0.6994
Epoch 48/50
319/319 [==============================] - 7s 23ms/step - loss: 0.4199 - recall: 0.81
16 - val_loss: 0.4270 - val_recall: 0.7055
Epoch 49/50
319/319 [==============================] - 8s 24ms/step - loss: 0.4148 - recall: 0.81
42 - val_loss: 0.4111 - val_recall: 0.6748
Epoch 50/50
319/319 [==============================] - 8s 26ms/step - loss: 0.4120 - recall: 0.81
71 - val_loss: 0.4316 - val_recall: 0.7025
```

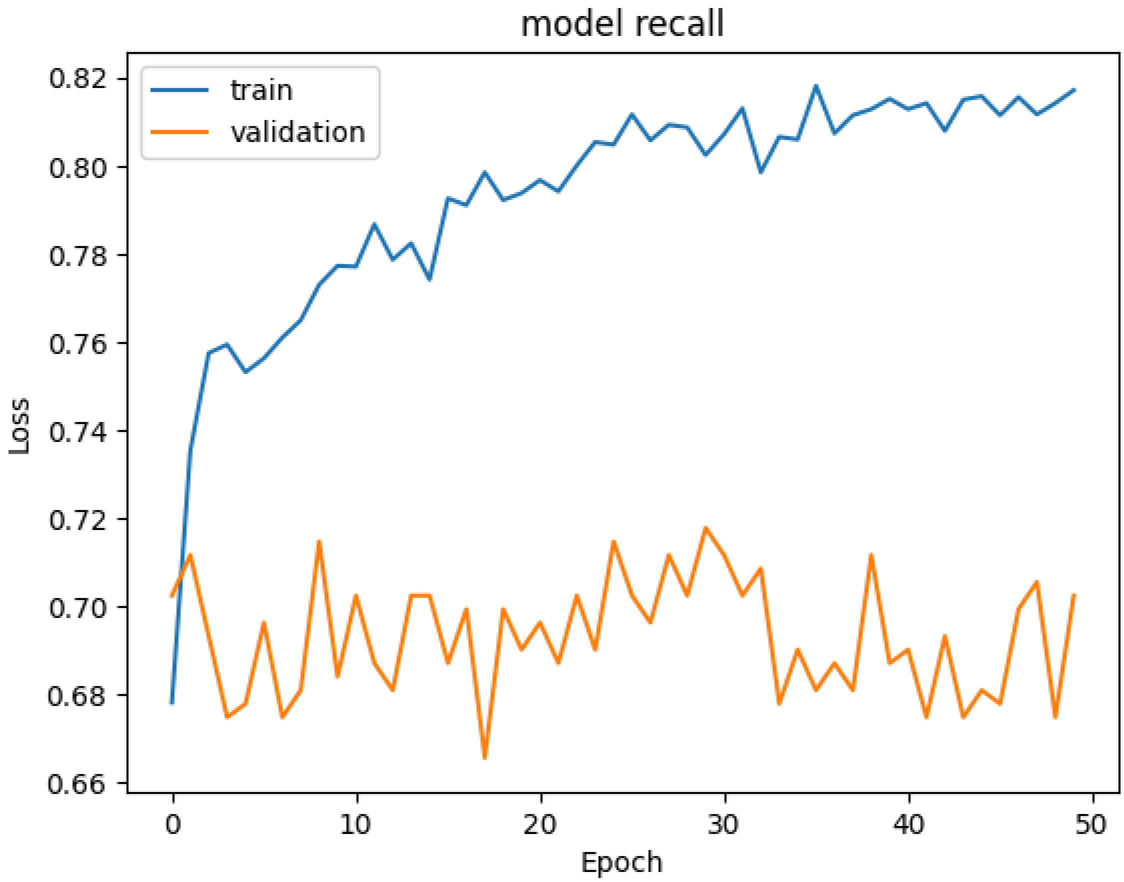
Loss function

In [ ]:

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



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



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

```
319/319 [==============================] - 6s 14ms/step
```

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

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

```
50/50 [==============================] - 2s 16ms/step
```

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

In [ ]:
```
model_name = "NN with SMOTE,Adam & Dropout"
```

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

Classification report

In [ ]:
```
cr=classification_report(y_train_smote,y_train_pred)
print(cr)
```

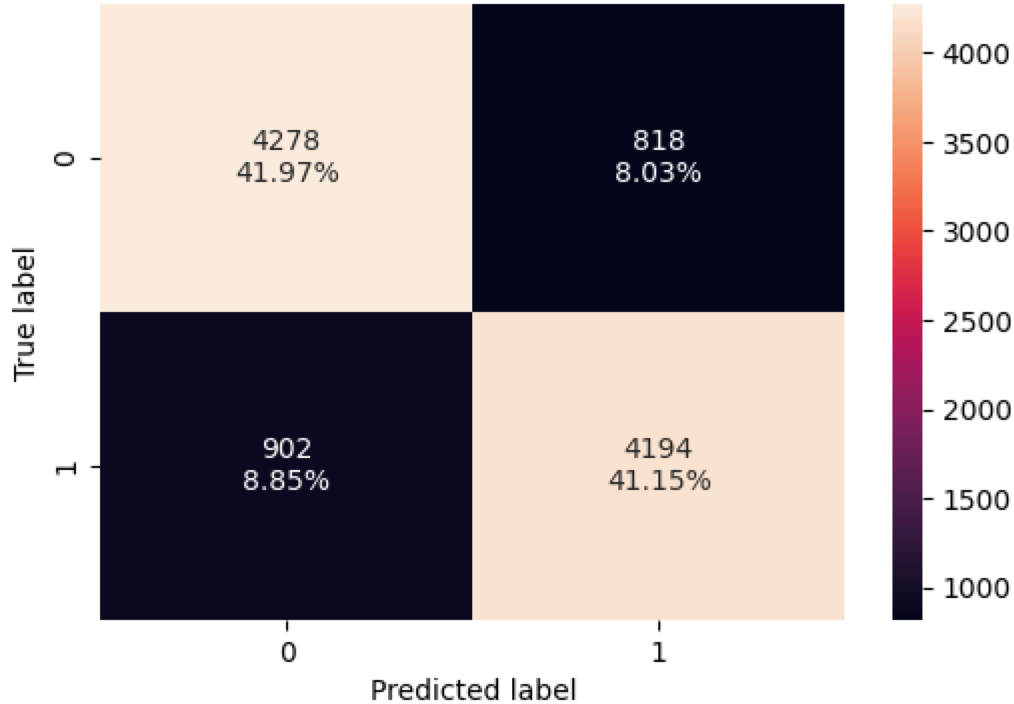
```
              precision    recall  f1-score   support

           0       0.83      0.84      0.83      5096
           1       0.84      0.82      0.83      5096

    accuracy                           0.83     10192
   macro avg       0.83      0.83      0.83     10192
weighted avg       0.83      0.83      0.83     10192
```

In [ ]:
```
#classification report
cr=classification_report(y_val,y_val_pred)
print(cr)
```

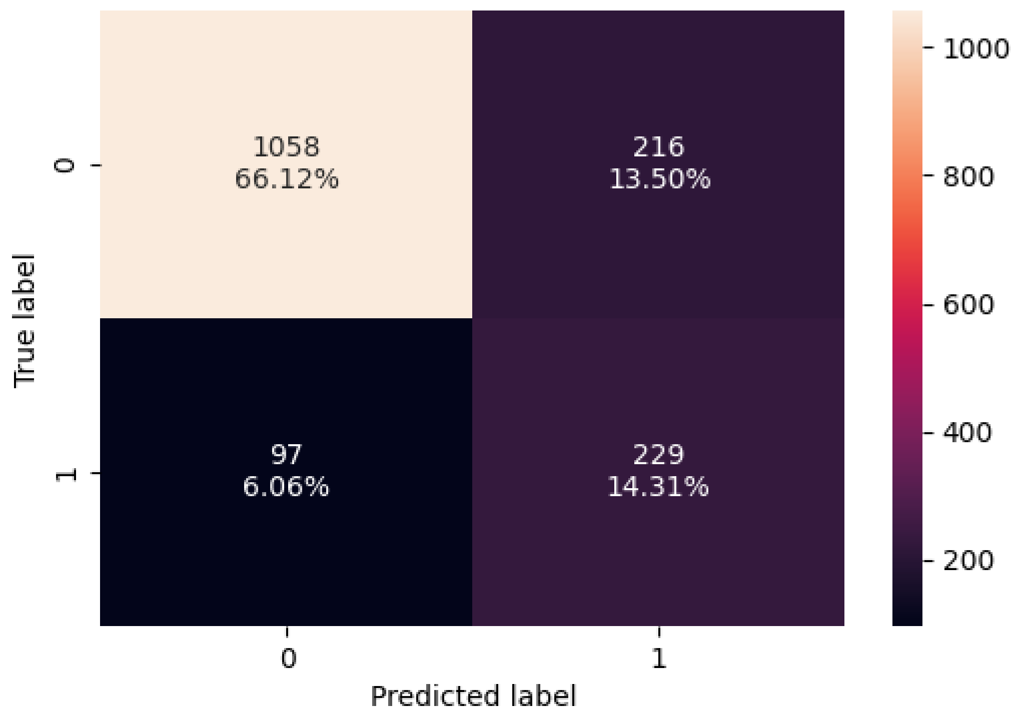
```
              precision    recall  f1-score   support

           0       0.92      0.83      0.87      1274
           1       0.51      0.70      0.59       326

    accuracy                           0.80      1600
   macro avg       0.72      0.77      0.73      1600
weighted avg       0.83      0.80      0.81      1600
```

Confusion matrix

In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_train_smote, y_train_pred)
```



In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_val,y_val_pred)
```



Model Performance Comparison and Final Model Selection

In [ ]:
```
print("Training performance comparison")
train_metric_df
```

```
Training performance comparison
```

Out[ ]:

	recall
NN with SGD	0.000767
NN with Adam	0.523006
NN with Adam & Dropout	0.603528
NN with SMOTE & SGD	0.766287
NN with SMOTE & Adam	0.940542
NN with SMOTE,Adam & Dropout	0.822998

In [ ]:
```
print("Validation set performance comparison")
valid_metric_df
```

```
Validation set performance comparison
```

Out[ ]:

	recall
NN with SGD	0.003067
NN with Adam	0.469325
NN with Adam & Dropout	0.490798
NN with SMOTE & SGD	0.714724
NN with SMOTE & Adam	0.604294
NN with SMOTE,Adam & Dropout	0.702454

In [ ]:
```
train_metric_df - valid_metric_df
```

Out[ ]:

	recall
NN with SGD	-0.002301
NN with Adam	0.053681
NN with Adam & Dropout	0.112730
NN with SMOTE & SGD	0.051563
NN with SMOTE & Adam	0.336247
NN with SMOTE,Adam & Dropout	0.120544

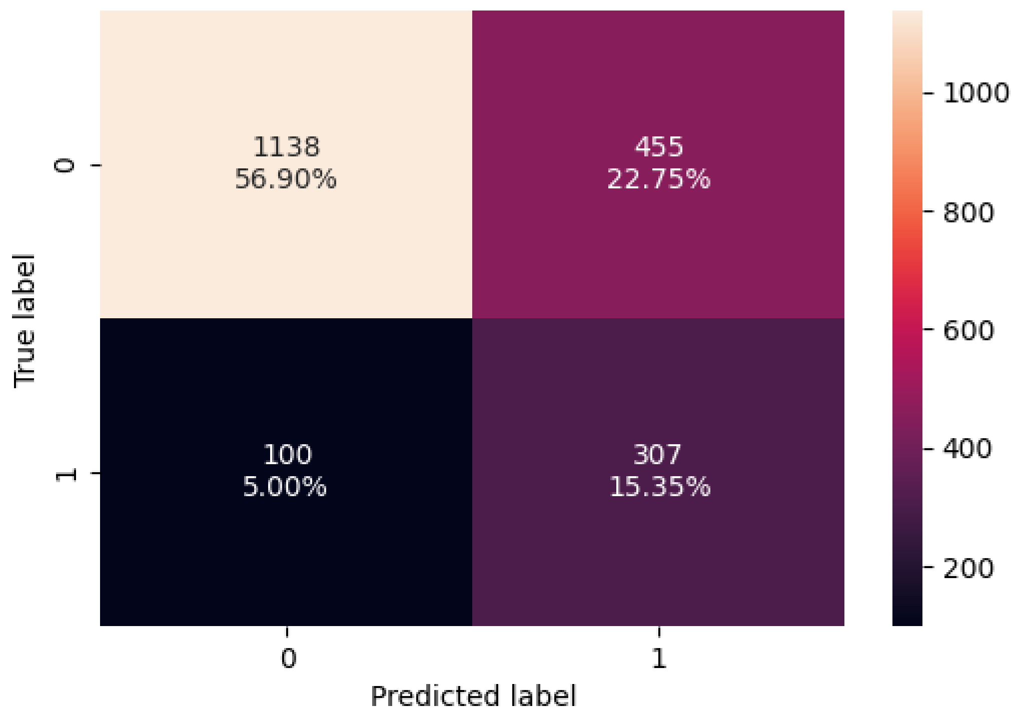
In [ ]:
```
y_test_pred = model_3.predict(X_test)
y_test_pred = (y_test_pred > 0.5)
print(y_test_pred)
```

```
63/63 [==============================] - 0s 4ms/step
[[False]
 [ True]
 [False]
 ...
 [ True]
 [False]
 [False]]
```

In [ ]:
```
#Lets print classification report
cr=classification_report(y_test,y_test_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.92      0.71      0.80      1593
           1       0.40      0.75      0.53       407

    accuracy                           0.72      2000
   macro avg       0.66      0.73      0.66      2000
weighted avg       0.81      0.72      0.75      2000
```

In [ ]:
```
#Calculating the confusion matrix
make_confusion_matrix(y_test,y_test_pred)
```



Actionable Insights and Business Recommendations

Recommendations and Insights

- The EDA showed that most predictors were not strong and were unbalanced
- The models that used SMOTE performed the best with the highest recall values. SMOTE balanced the data set and took care of underrepresented classes. The imbalanced data led to weaker models for the first 3 tested models. ### Conclusion
- The Neural Network Model with SGD and SMOTE is the model with the highest recall. (Model 3)
- This model was slightly overfit but produced the highest recall at 71.5% in the validation set with slight overfitting.

- This model is the best to predict customers action since we are maximizing recall and minimizing false negatives.

By Raghuram Palaniappan