Bank Churn Prediction

Problem Statement

Context

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

Objective

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

Data Dictionary

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

In [ ]:

```

```

Importing necessary libraries

In [1]:

```python
import pandas as pd  # Library for data manipulation and analysis.
import numpy as np   # Fundamental package for scientific computing.
import matplotlib.pyplot as plt  # Plotting library for creating visualizations.
import seaborn as sns #For advanced visualizations.

from sklearn.model_selection import train_test_split  # Function for splitting datasets for training and testin
from sklearn.preprocessing import StandardScaler

import time  # Module for time-related operations.
from imblearn.over_sampling import SMOTE
import tensorflow as tf #An end-to-end open source machine learning platform
from tensorflow import keras  # High-level neural networks API for deep learning.
from keras import backend   # Abstraction layer for neural network backend engines.
from sklearn.metrics import confusion_matrix,roc_curve,classification_report,recall_score, precision_score, f1_
from keras.models import Sequential  # Model for building NN sequentially.
from keras.layers import Dense,Dropout,BatchNormalization   # for creating fully connected neural network layer
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from scipy.stats import chi2_contingency
```

Loading the dataset

In [2]:
```
# Mounting google drive and initilizing path variable
from google.colab import drive
drive.mount("/content/drive")
path = '/content/drive/MyDrive/PGPAIML/Project-4/'
```

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

In [3]:
```
# Loading the data
df = pd.read_csv(path+'Churn.csv')
# Making a copy to keep the original data intact, may required later.
data = df.copy()
```

Data Overview

In [ ]:
```
data.head(10)
```

Out[ ]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsA
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
5	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	
6	7	15592531	Bartlett	822	France	Male	50	7	0.00	2	1	
7	8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	
8	9	15792365	He	501	France	Male	44	4	142051.07	2	0	
9	10	15592389	H?	684	France	Male	27	2	134603.88	1	1	

In [ ]:
```
data.shape
```

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

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [ ]:
```
data.describe(include='all').T
```

Out[ ]:

	count	unique	top	freq	mean	std	min	25%	50%	75%
RowNumber	10000.0	NaN	NaN	NaN	5000.5	2886.89568	1.0	2500.75	5000.5	7500.25
CustomerId	10000.0	NaN	NaN	NaN	15690940.5694	71936.186123	15565701.0	15628528.25	15690738.0	15753233.75
Surname	10000	2932	Smith	32	NaN	NaN	NaN	NaN	NaN	NaN
CreditScore	10000.0	NaN	NaN	NaN	650.5288	96.653299	350.0	584.0	652.0	718.0
Geography	10000	3	France	5014	NaN	NaN	NaN	NaN	NaN	NaN
Gender	10000	2	Male	5457	NaN	NaN	NaN	NaN	NaN	NaN
Age	10000.0	NaN	NaN	NaN	38.9218	10.487806	18.0	32.0	37.0	44.0
Tenure	10000.0	NaN	NaN	NaN	5.0128	2.892174	0.0	3.0	5.0	7.0
Balance	10000.0	NaN	NaN	NaN	76485.889288	62397.405202	0.0	0.0	97198.54	127644.24
NumOfProducts	10000.0	NaN	NaN	NaN	1.5302	0.581654	1.0	1.0	1.0	2.0
HasCrCard	10000.0	NaN	NaN	NaN	0.7055	0.45584	0.0	0.0	1.0	1.0
IsActiveMember	10000.0	NaN	NaN	NaN	0.5151	0.499797	0.0	0.0	1.0	1.0
EstimatedSalary	10000.0	NaN	NaN	NaN	100090.239881	57510.492818	11.58	51002.11	100193.915	149388.2475
Exited	10000.0	NaN	NaN	NaN	0.2037	0.402769	0.0	0.0	0.0	0.0

Observations

- The dataset has 10,000 rows and 14 columns
- The *Exited* column is the target variable (1 = Churned, 0 = Not Churned)
- Categorical Columns - Geography, Gender, HasCrCard and IsActiveMember.
- Numerical Columns - CreditScore, Age, Balance, and EstimatedSalary - may need normalization.
- No missing values are indicated, as all columns have 10000 non-null entries.
- *RowNumber* & *CustomerId* columns are identifiers and don't provide useful predictive information.
- There are 2932 unique surnames, with "Smith" being the most frequent (32 times). This feature may only have significant predictive power if linked to geography or culture.
- *CreditScore* column Normally distributed with no extreme outliers.
- Gender equally distributed Male (5457 occurrences) and Female.
- Customer's *Age* ranges from 18 to 92 with a mean 38.92.
- Customer's *Tenure* mean is 5 with a range 0 to 10.
- For *Balance*, the mean is 76,485.89, but the 25th percentile is 0, indicating many customers have zero balance.
- For *NumOfProducts*, the mean is 1.53, range: 1 to 4 products.
- *HasCrCard* is a binary variable (0 or 1). It only represents whether the customer owns a credit card or not.
- *IsActiveMember* is also a binary variable (0 or 1) with a mean of 0.5151.
- For *EstimatedSalary* the mean is 100,090.24, range: 11.58 to 199,992.48. Wide range but no extreme outliers. Normalization or scaling may help.

Exploratory Data Analysis

Common Functions used for EDA

In [4]:

```
# 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 the show 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 triangle will indicate the mean value of the column
    sns.histplot(
```

```python
        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 [5]:
```python
# 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
```

In [6]:
```python
# 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))
```

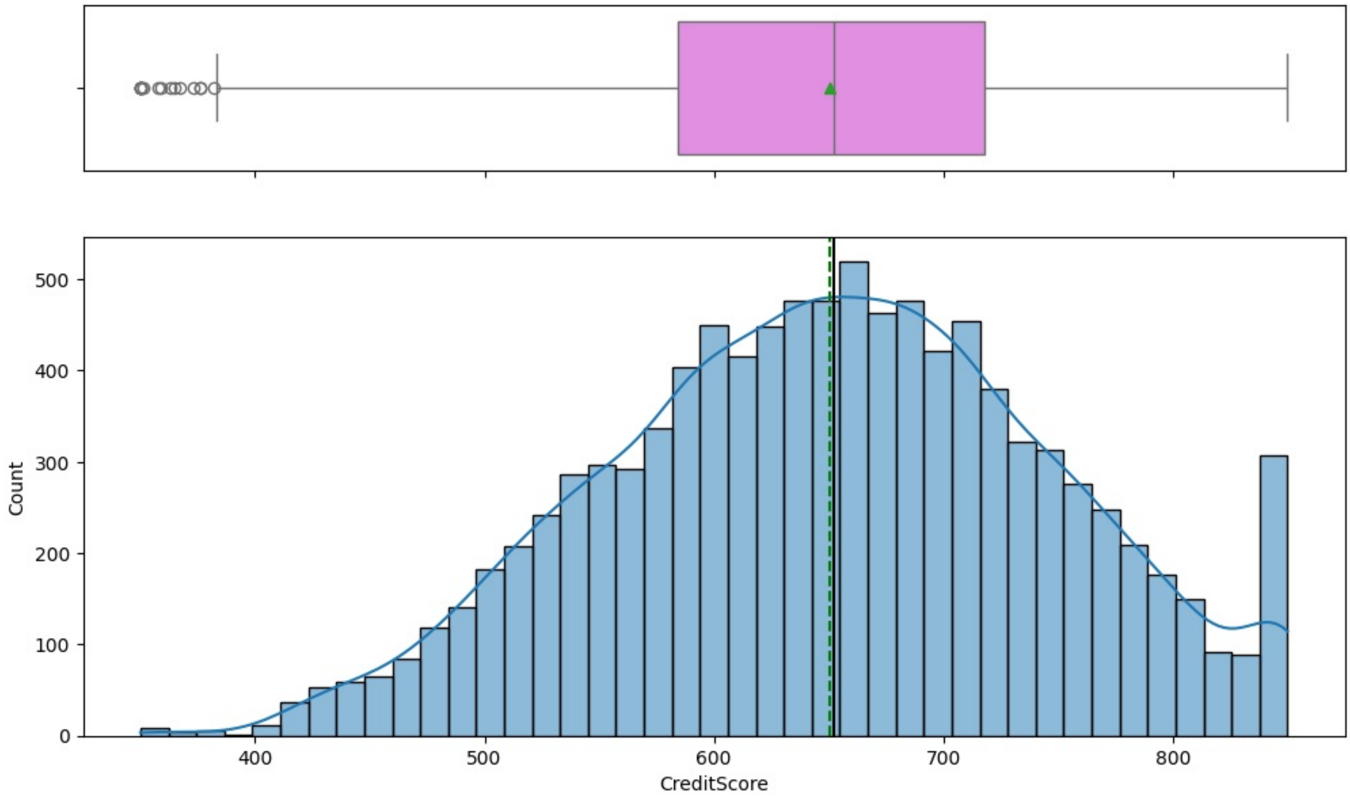
```python
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [7]:
```python
### Function to plot distributions

def distribution_plot_wrt_target(data, predictor, target):

    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    target_uniq = data[target].unique()

    axs[0, 0].set_title("Distribution of target for target=" + str(target_uniq[0]))
    sns.histplot(
        data=data[data[target] == target_uniq[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color="teal",
    )

    axs[0, 1].set_title("Distribution of target for target=" + str(target_uniq[1]))
    sns.histplot(
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color="orange",
    )

    axs[1, 0].set_title("Boxplot w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="gist_rainbow")

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

    plt.tight_layout()
    plt.show()
```

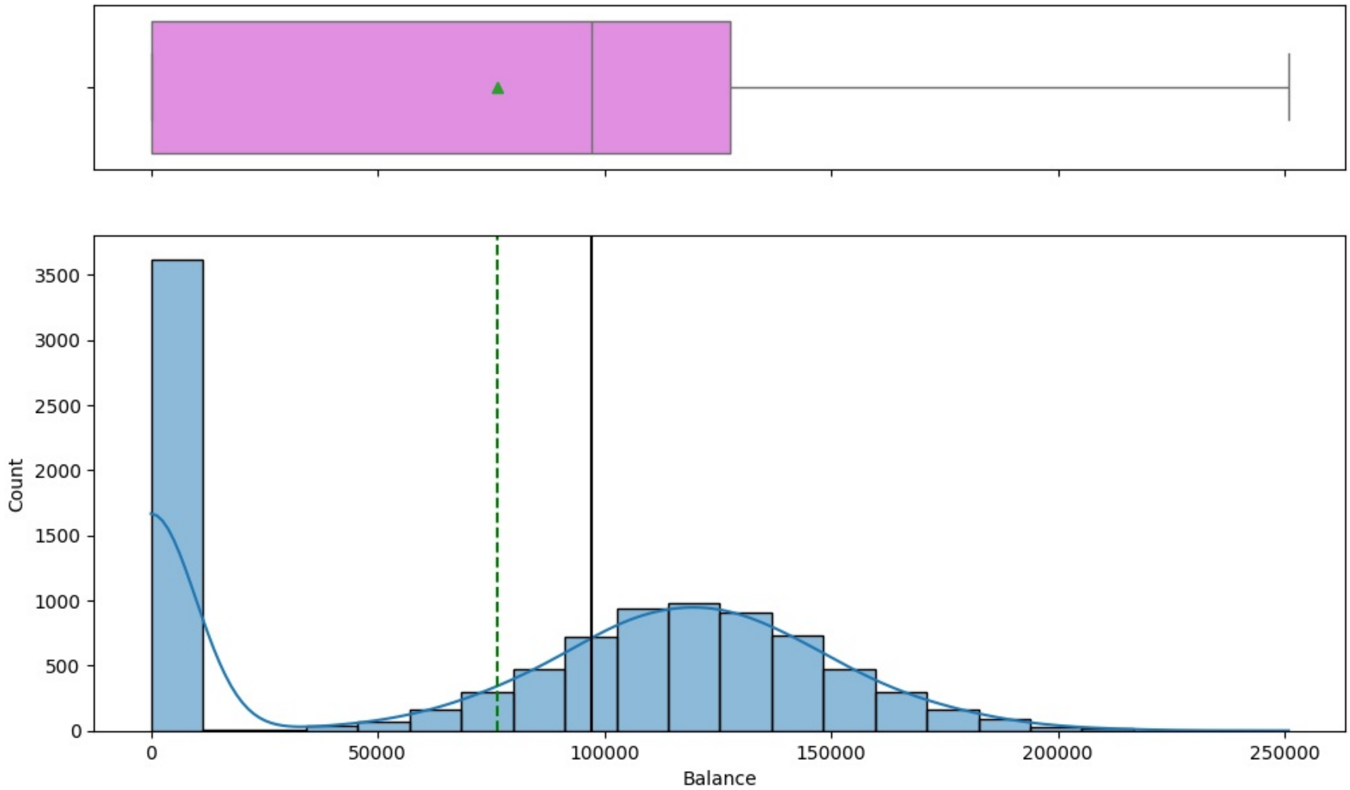
Univariate Analysis

Analysis of 4 Continious Variables - CreditScore, Balance, EstimatedSalary & Age.

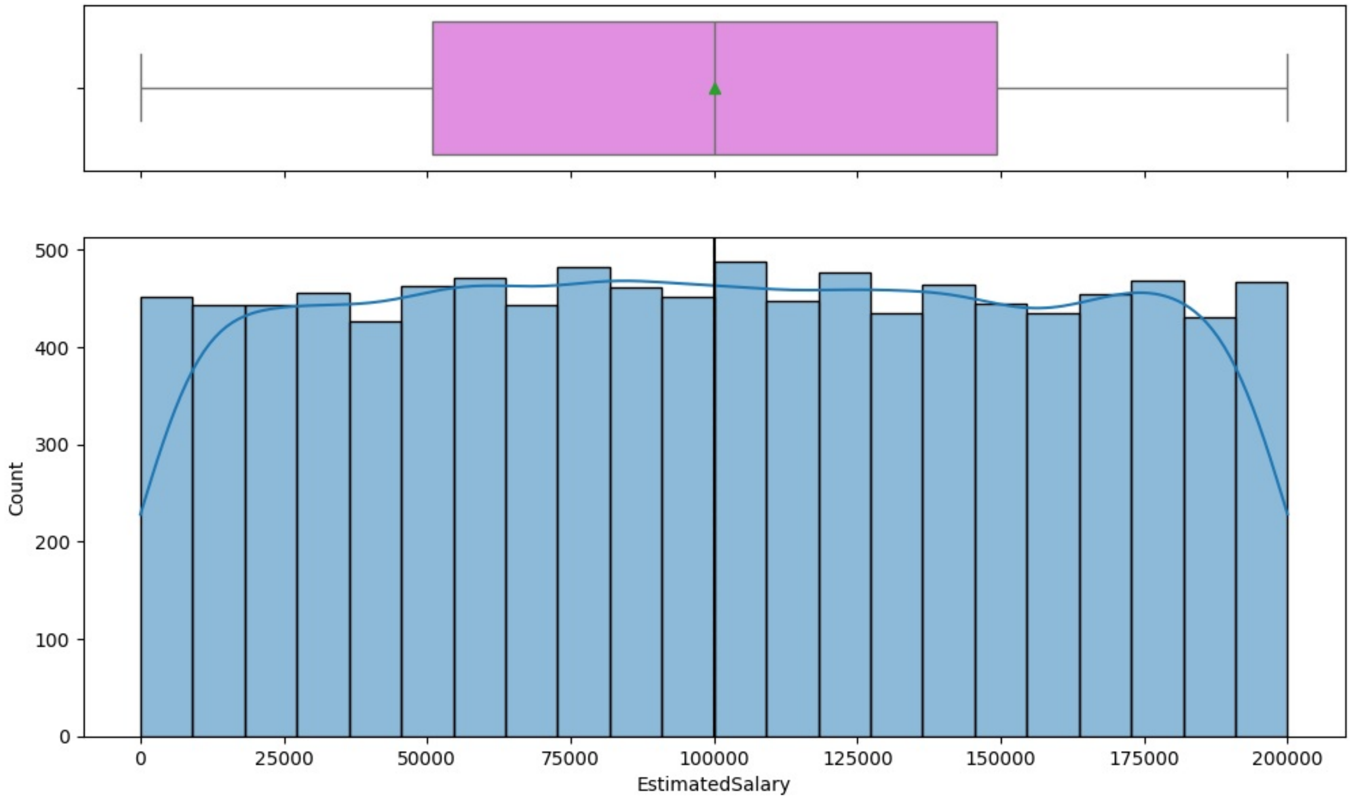
In [ ]:
```python
histogram_boxplot(data=data,feature='CreditScore', kde=True)
```



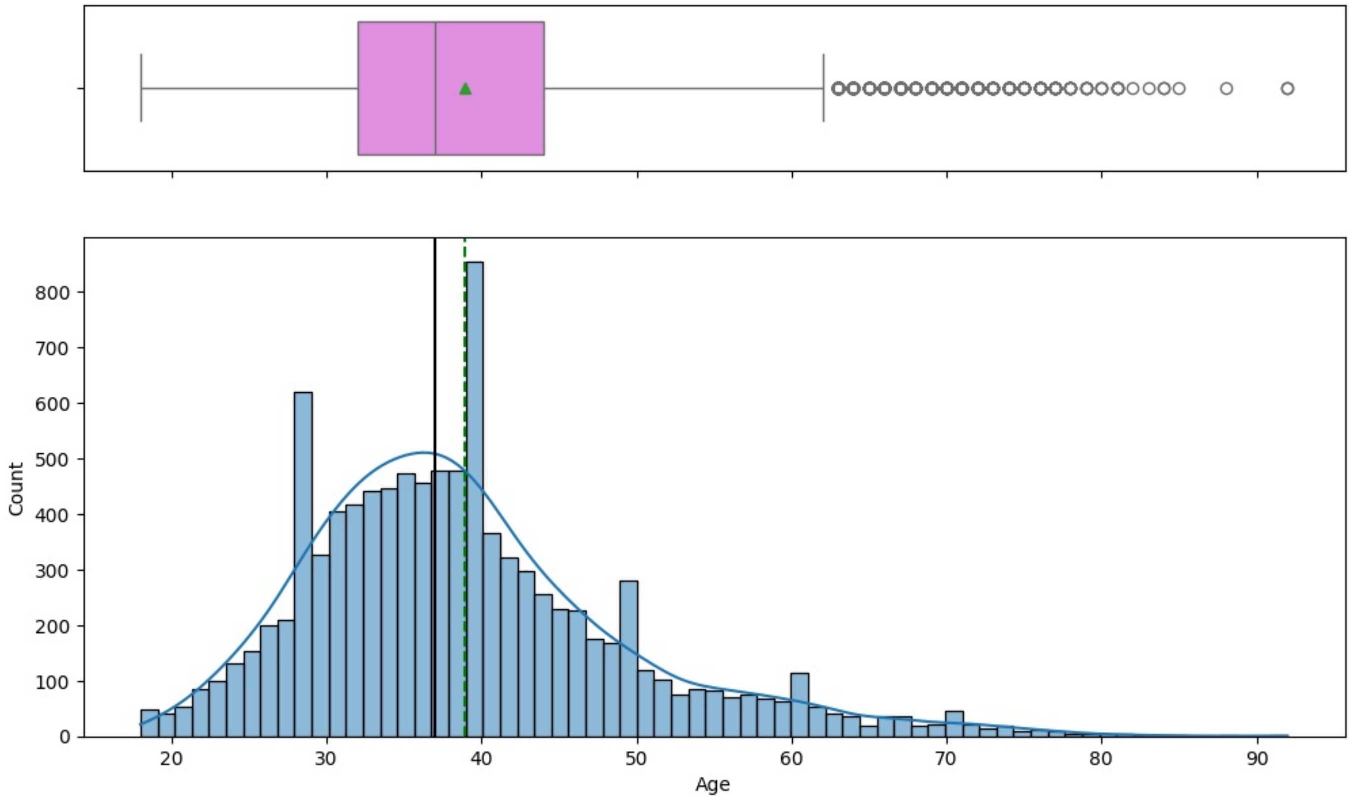

```
In [ ]: histogram_boxplot(data=data,feature='Balance', kde=True)
```




```
In [ ]: histogram_boxplot(data=data,feature='EstimatedSalary', kde=True)
```




```
In [ ]: histogram_boxplot(data=data,feature='Age', kde=True)
```



Observations

1. Credit Score - The credit scores appear normally distributed, with most values concentrated around 650–700. There are a few outliers on the lower end (less than 400), but they are not extreme. CreditScore seems clean, and the distribution suggests it's a significant feature to retain. Outliers may not need removal as they are minor.

2. Balance - The distribution shows a high concentration at 0, followed by a wide range of balances up to ~250,000. No significant outliers, but the skewness caused by a high number of zero-balance accounts. The presence of many customers with zero balance could indicate inactive accounts, a potential indicator of churn.

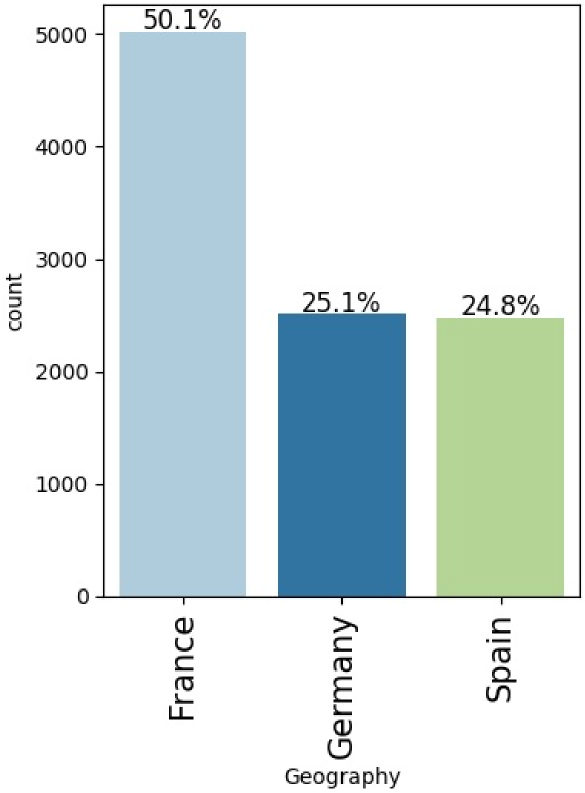
3. Estimated Salary - Salaries are uniformly distributed across the range (0 to ~200,000). No noticeable outliers or skewness. Salary appears evenly distributed, with no transformations necessary. It might serve as a neutral or minor predictor of churn.

4. Age - The age distribution is right-skewed, with a peak around 35–40 years and very few older customers (>70). Outliers exist on the higher end (>70), but these are expected for older customer demographics. Age is a critical feature and shows variability. Older

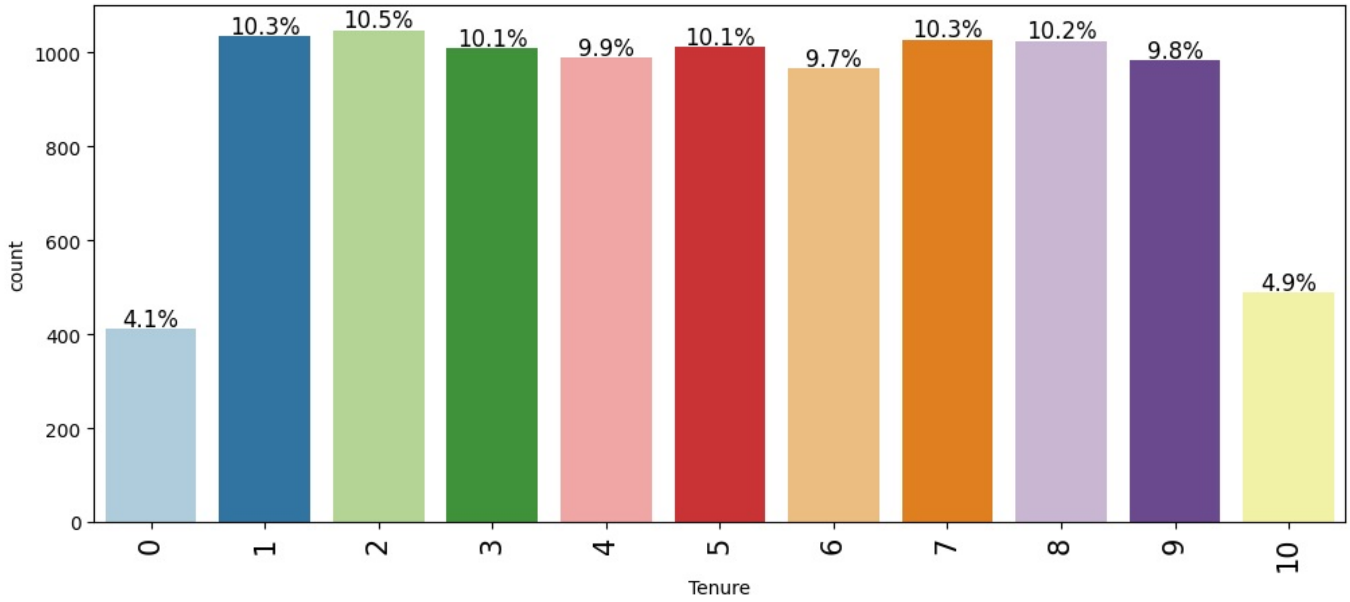
customers may have unique churn behaviors.

Analysis of 4 Categorical & 2 Binary variables - Geography, Tenure, NumOfProducts, Gender, HasCrCard & IsActiveMember.

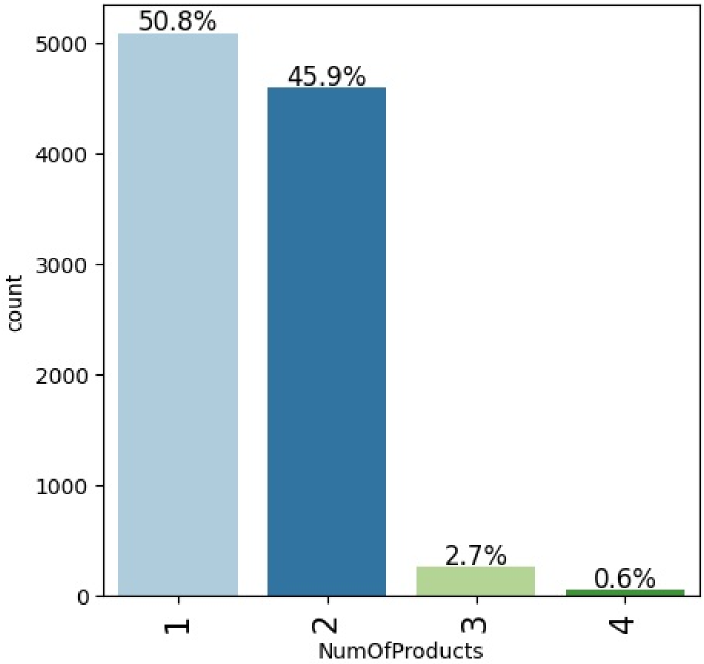
In [8]:
```
labeled_barplot(data=data, feature='Geography', perc=True)
```



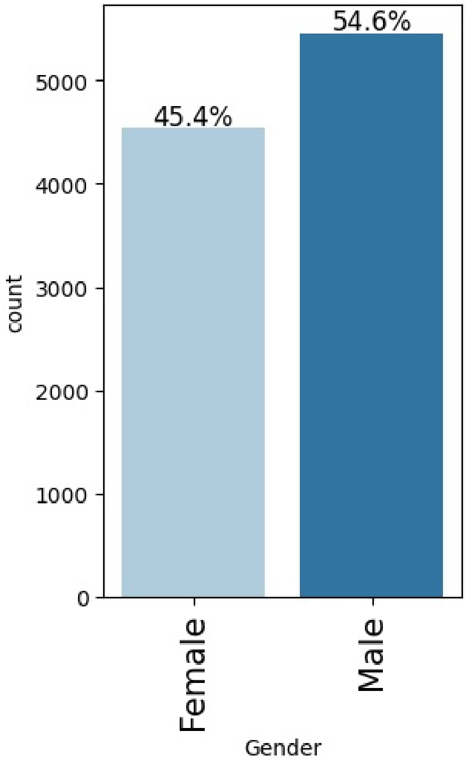
In [9]:
```
labeled_barplot(data=data, feature='Tenure', perc=True)
```



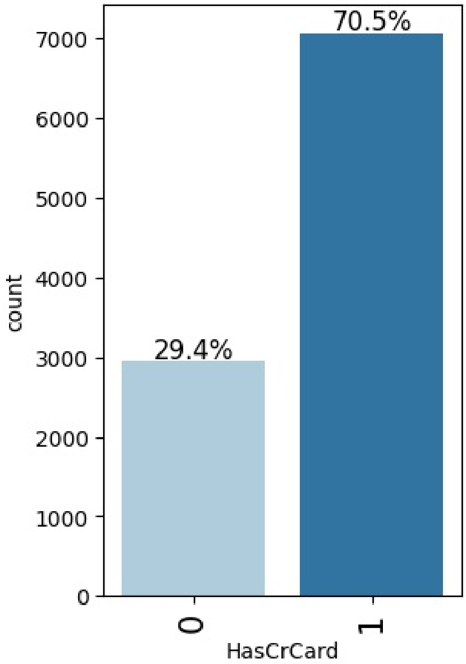
In [10]:
```
labeled_barplot(data=data, feature='NumOfProducts', perc=True)
```



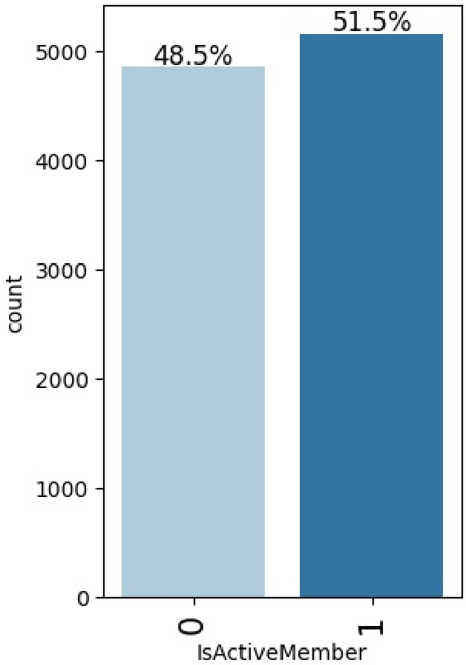
In [11]:
```
labeled_barplot(data=data, feature='Gender', perc=True)
```



In [12]:
```
labeled_barplot(data=data, feature='HasCrCard', perc=True)
```



In [13]:
```
labeled_barplot(data=data, feature='IsActiveMember', perc=True)
```



Observations

1. Geography: France has the highest proportion of customers (50.1%), followed by Germany (25.1%) and Spain (24.8%). The data is slightly imbalanced geographically. This may reflect the bank's operations or customer base Geography could influence churn due to regional differences in services or competition.

2. Tenure: It is relatively evenly distributed from 1 to 9 years (~10% each), except for:

   - 0 years (4.1%): Likely new or inactive customers.
   - 10 years (4.9%): Loyal customers.

Customers with extremely low or high tenure may have different churn behaviors, requiring further analysis.

3. Number of Products (NumOfProducts): Most customers hold **1 product (50.8%)** or **2 products (45.9%)**. Very few have 3 (2.7%) or 4 products (0.6%). The limited adoption of more than 2 products may indicate an opportunity to cross-sell or upsell, potentially influencing churn.

4. Gender: Males slightly outnumber females in the dataset (54.6% vs. 45.4%).

5. Has Credit Card (HasCrCard): Most customers (70.5%) have a credit card, while 29.4% do not. This could impact churn rates; customers without credit cards may use fewer services and be more likely to leave.

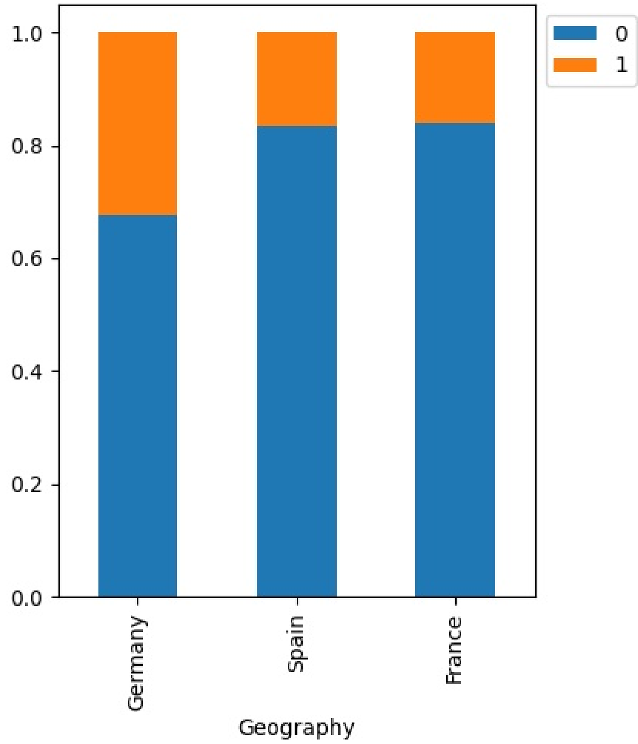
6. Is Active Member (IsActiveMember): The active member ratio is almost evenly split (51.5% active vs. 48.5% inactive). This may be a critical feature; inactive members are likely at higher risk of churn, making this a key predictor.

Bivariate Analysis

Bivariate Analysis of Categorical variables with respect to Target Variable (Exited)

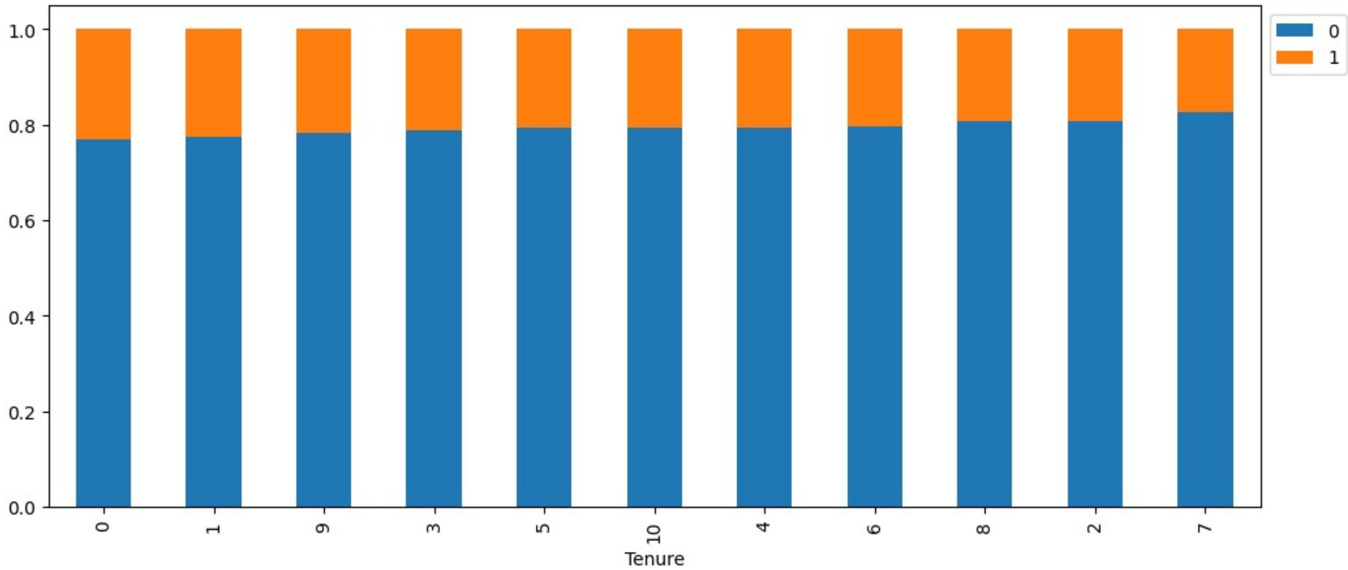
In [ ]:
```
stacked_barplot(data=data, predictor='Geography', target='Exited')
```

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



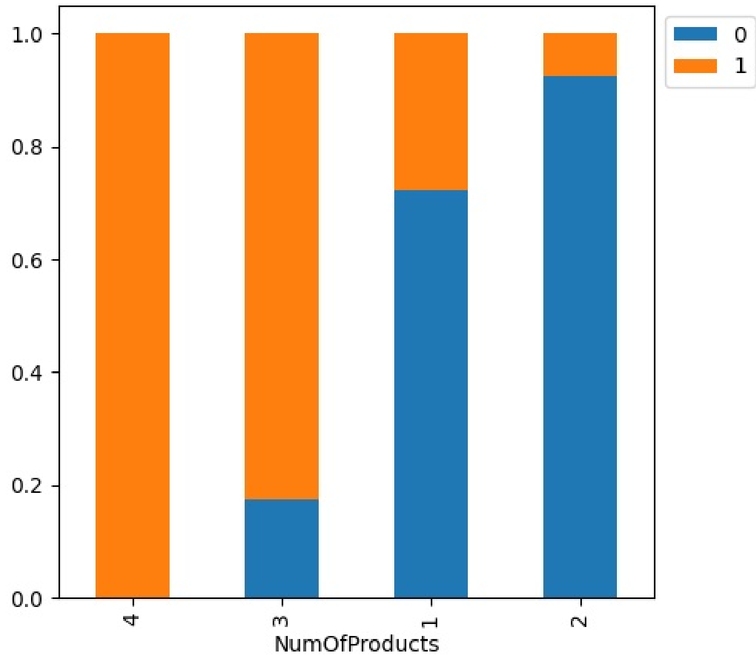
In [ ]:
```
stacked_barplot(data=data, predictor='Tenure', target='Exited')
```

```
Exited     0     1    All
Tenure
All     7963  2037  10000
1        803   232   1035
3        796   213   1009
9        771   213    984
5        803   209   1012
4        786   203    989
2        847   201   1048
8        828   197   1025
6        771   196    967
7        851   177   1028
10       389   101    490
0        318    95    413
------------------------------------------------------------------------------------------------------------------------
```



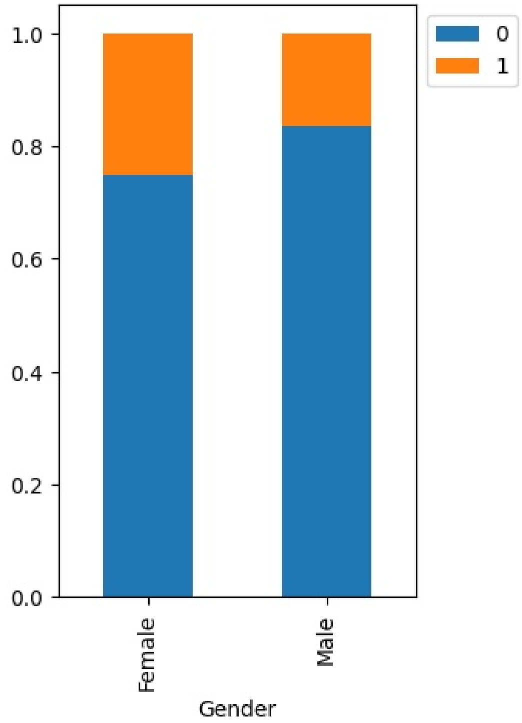

```
In [ ]: stacked_barplot(data=data, predictor='NumOfProducts', target='Exited')
```

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



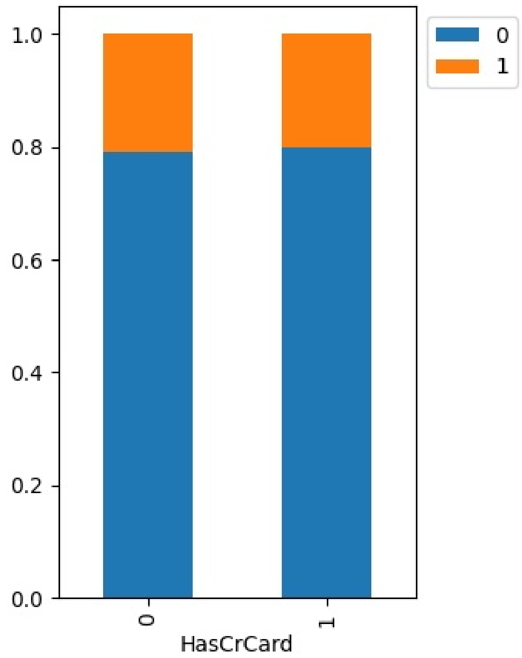

```
In [ ]: stacked_barplot(data=data, predictor='Gender', target='Exited')
```

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



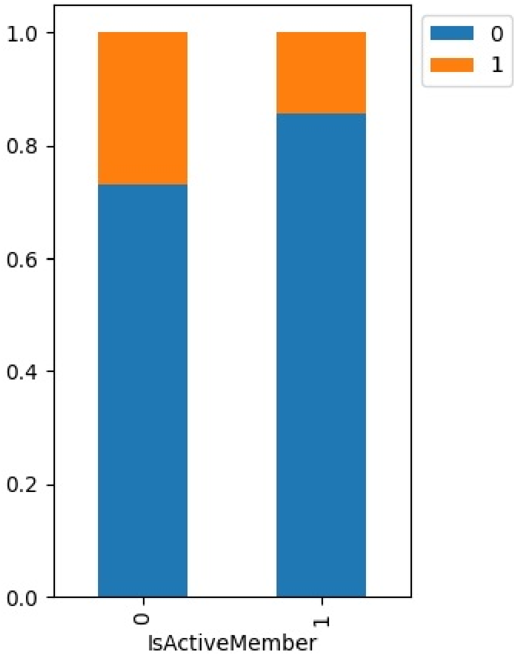
In [ ]:
```
stacked_barplot(data=data, predictor='HasCrCard', target='Exited')
```

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



In [ ]:
```
stacked_barplot(data=data, predictor='IsActiveMember', target='Exited')
```

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

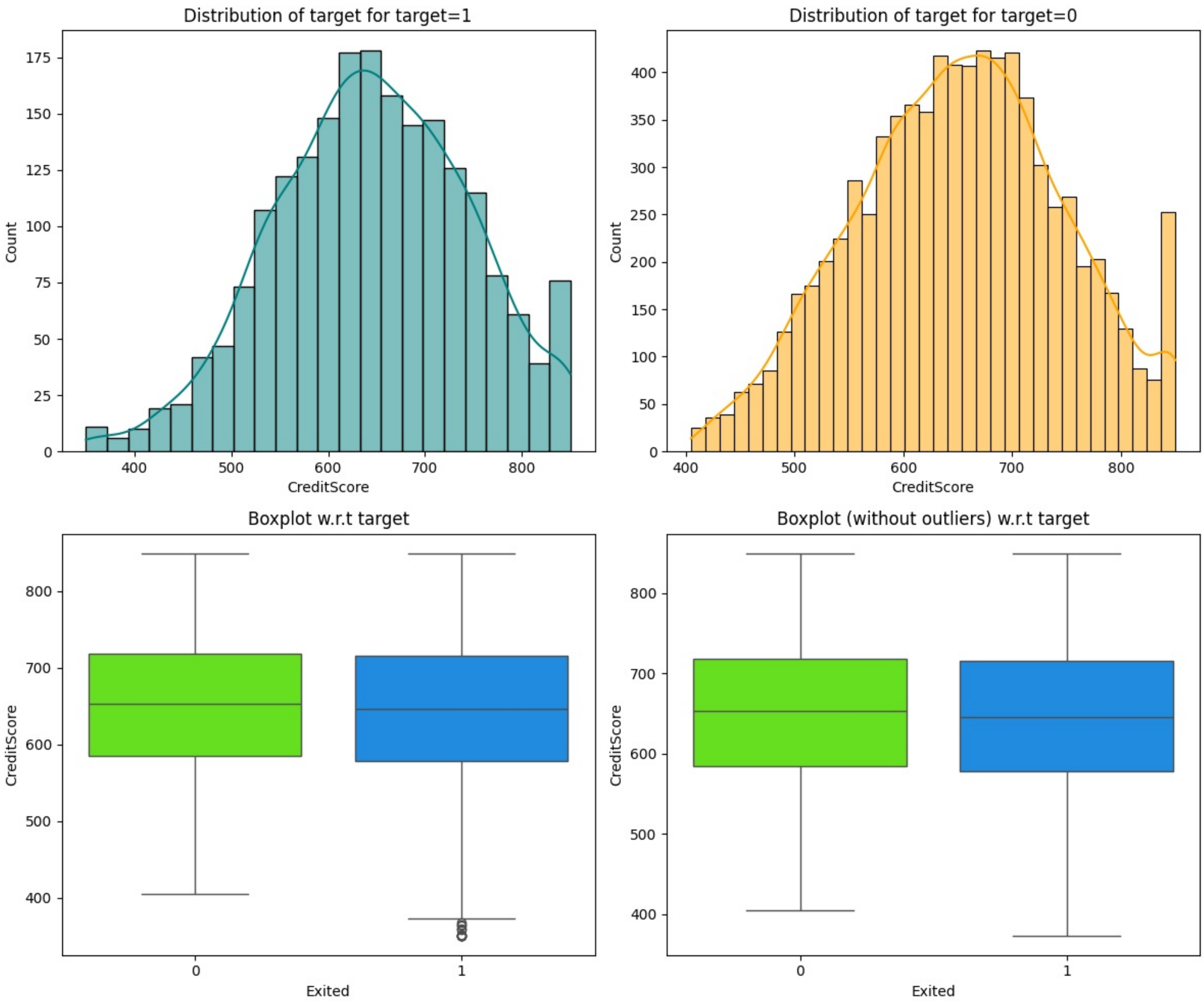


Observations

1. Geography
   - Germany has the highest proportion of customers who churned (32.4%), followed by Spain (16.7%) and France (16.1%).
   - Geography significantly influences churn, with German customers being at higher risk.
2. Tenure
   - Customers with **0 years of tenure** have the highest churn rate (23%). Churn rates decrease as tenure increases, but customers with long tenure (10 years) also show slightly elevated churn (~20.6%).
   - Customers with very short or very long tenure may be more likely to leave.
3. Number of Products
   - Customers with **1 product** have the highest churn (~27.7%), while those with 2 products have the lowest (~7.6%).
   - Customers with 3 or 4 products show extremely high churn rates (~82.7% and 100%, respectively).
4. Gender
   - Female customers have a higher churn rate (~25.1%) compared to males (~16.4%).
5. Has Credit Card
   - Customers without credit cards have a slightly higher churn rate (~20.8%) compared to those with credit cards (~16.5%).
   - Credit card ownership may be associated with higher engagement, reducing churn.
6. Is Active Member
   - Inactive members have a significantly higher churn rate (~26.9%) compared to active members (~14.3%).
   - Customer engagement is a strong predictor of retention, emphasizing the importance of involvement.

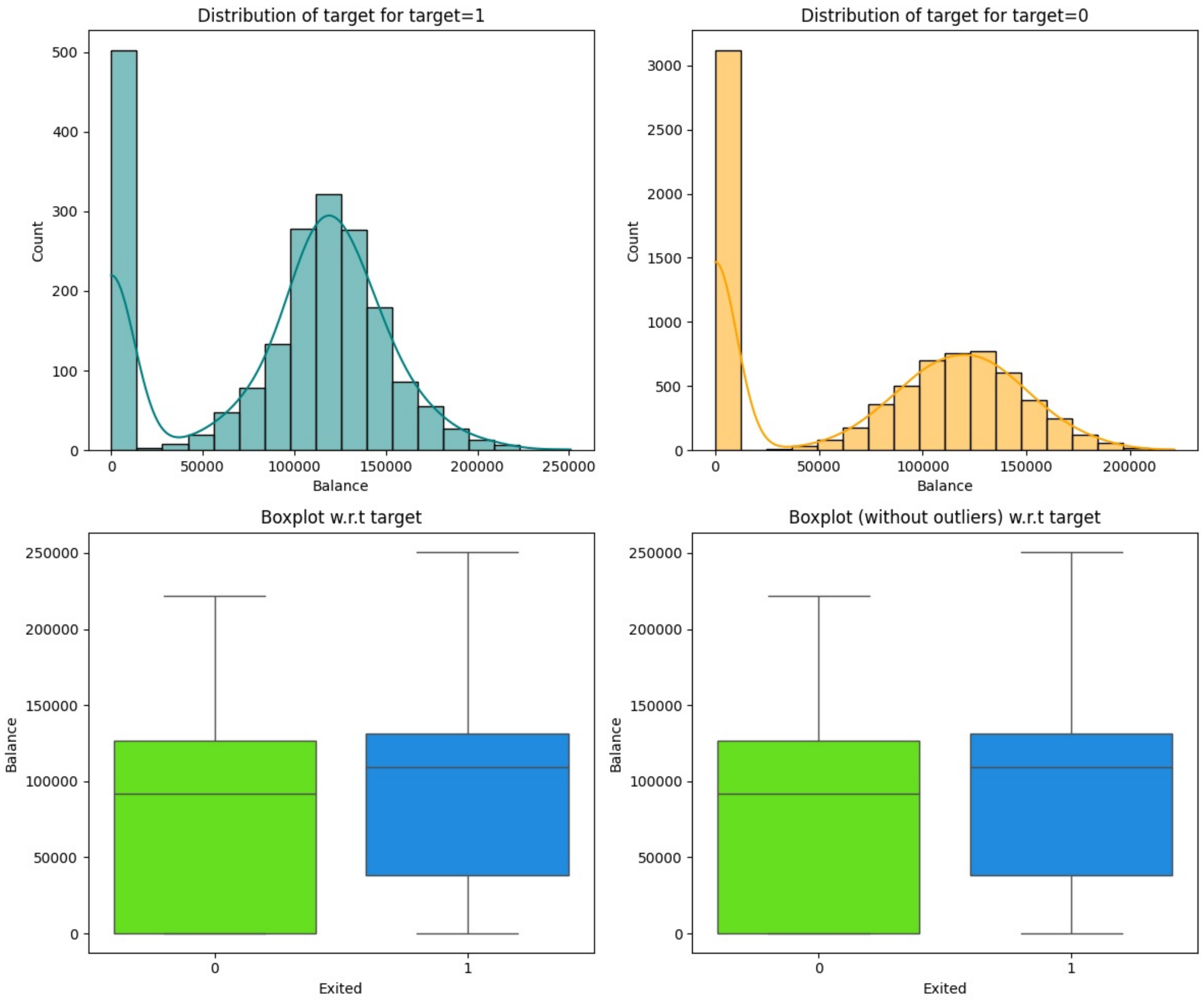
Bivariate Analysis of Continius variables with respect to Target Variable (Exited)

In [ ]:
```
distribution_plot_wrt_target(data=data, predictor='CreditScore', target='Exited')
```

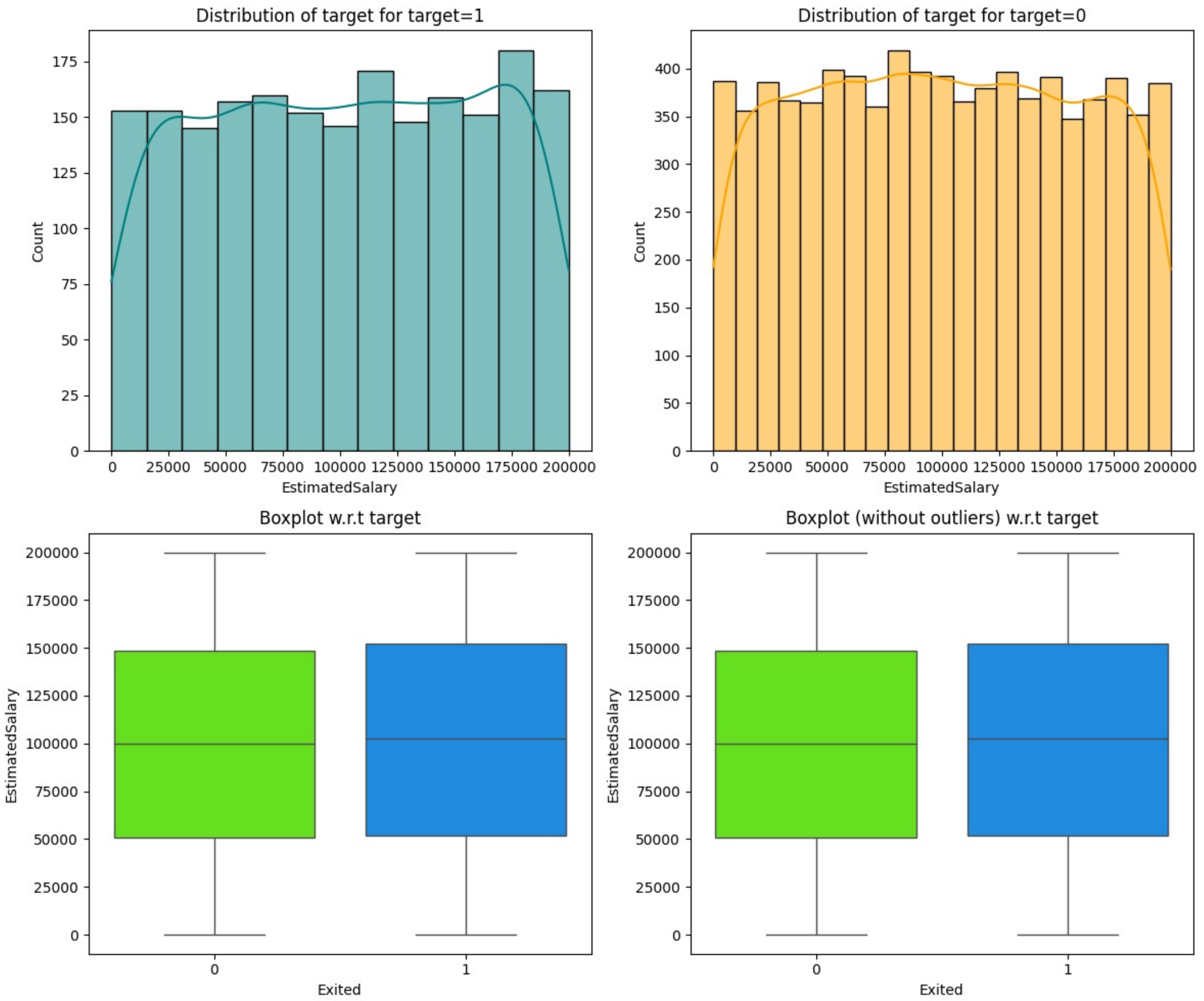


In [ ]:

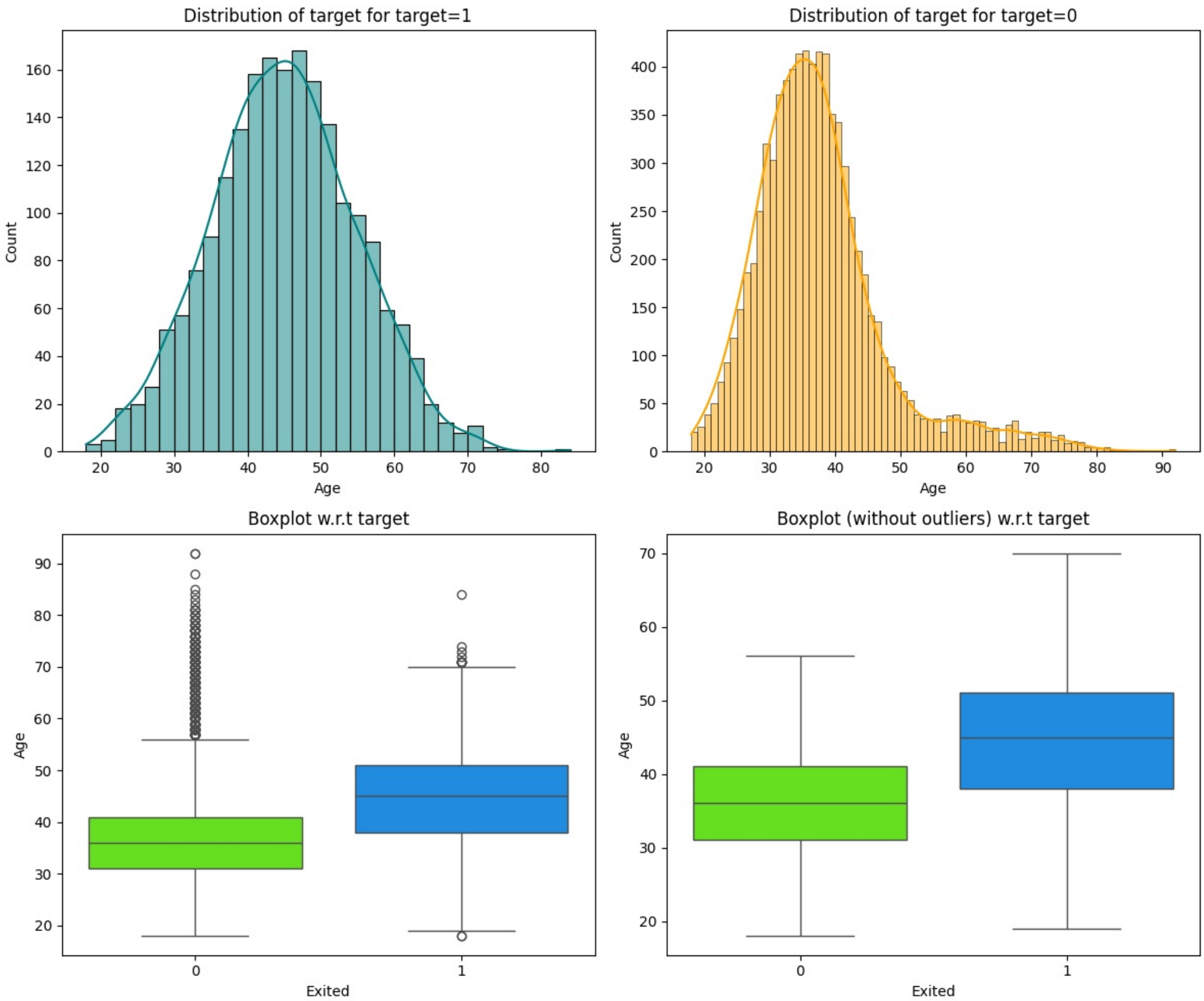
```
distribution_plot_wrt_target(data=data, predictor='Balance', target='Exited')
```



In [ ]:
```
distribution_plot_wrt_target(data=data, predictor='EstimatedSalary', target='Exited')
```




```
In [ ]: distribution_plot_wrt_target(data=data, predictor='Age', target='Exited')
```



Observations

1. Age

   - Target=1 (Churned): The distribution is concentrated around 40-60 years, with the median age being higher than for non-churned customers.
   - Target=0 (Non-Churned): The age distribution is broader, including younger customers, but with a notable peak in the 30-50 range.
   - Churned customers tend to be older than non-churned customers. Age seems to be a strong predictor.

2. Estimated Salary

   - The distribution appears relatively uniform across the salary range for both churned and non-churned customers.
   - There is no significant difference in the salary distribution between churned and non-churned customers. Estimated Salary may not be a strong predictor.

3. Balance

   - Target=1 (Churned): Customers with higher balances show a higher tendency to churn. Also notable number of churned customers have zero balance.
   - Target=0 (Non-Churned): Many non-churned customers also have a balance of zero, but the distribution is broader.
   - Higher account balances might slightly correlate with churn, but a large proportion of churned customers still have zero balance.

4. Credit Score

   - The credit score distributions for churned and non-churned customers are similar, with no distinct peaks.
   - The median credit score is slightly higher for churned customers. Credit score alone might not be a strong predictor for churn.

Analysis of the impact of Surname as a predictor variable against the target variable.

In [ ]:
```
# Counting how often each surname appears for churned (Exited = 1) and non-churned (Exited = 0) customers.
surname_analysis = data.groupby(['Surname', 'Exited']).size().unstack(fill_value=0)
surname_analysis['Total_Occurance'] = surname_analysis[0] + surname_analysis[1]
# Calculating 'Churn_Rate' as exited / total
surname_analysis['Churn_Rate'] = surname_analysis[1] / (surname_analysis[0] + surname_analysis[1])
surname_analysis.sort_values('Total_Occurance', ascending=False).head(10)  # Top surnames by churn rate
```

Out[ ]:

Exited	0	1	Total_Occurance	Churn_Rate
Surname				
Smith	23	9	32	0.281250
Martin	20	9	29	0.310345
Scott	26	3	29	0.103448
Walker	24	4	28	0.142857
Brown	21	5	26	0.192308
Shih	18	7	25	0.280000
Genovese	21	4	25	0.160000
Yeh	22	3	25	0.120000
Wright	18	6	24	0.250000
Maclean	19	5	24	0.208333

Observation:

- The top 10 surnames sorted by total occurrences range from 24 to 32 total counts.
- Churn rates for these surnames vary from 10.3% (Scott) to 31.0% (Martin).
- Common surnames like **Smith** (32 occurrences) and **Martin** (29 occurrences) show churn rates of 28.1% and 31%, respectively, which align with the dataset's overall churn rate (~20%).
- While there is some variability in churn rates across surnames, the differences are not dramatic. Found that A **Chi-Square test** can confirm whether these variations are statistically significant, here

In [ ]:

```
contingency_table = pd.crosstab(data['Surname'], data['Exited'])
chi2, p, dof, expected = chi2_contingency(contingency_table)
print("Chi-Square Test p-value:", p)
```

```
Chi-Square Test p-value: 0.9720408097645417
```

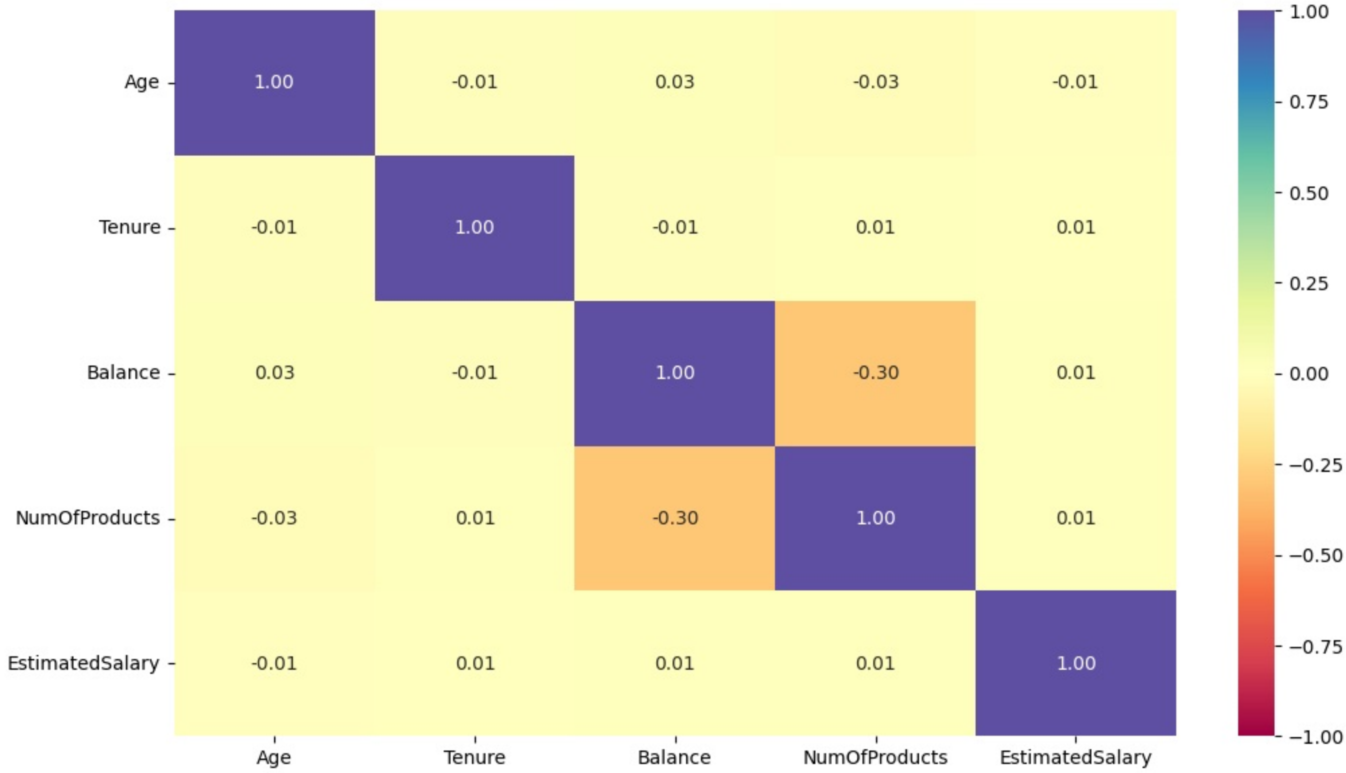
Observations

- The p-value of 0.972 from the Chi-Square test indicates that there is no significant association between the Surname column and the target variable Exited. This high p-value means that any observed differences in churn rates across surnames are likely due to random chance rather than an underlying relationship.
- Decided that **dropping the Surname column** is the most logical choice to avoid adding **noise** or **unnecessary complexity** to the model.

Corelations and Pairplot

In [ ]:

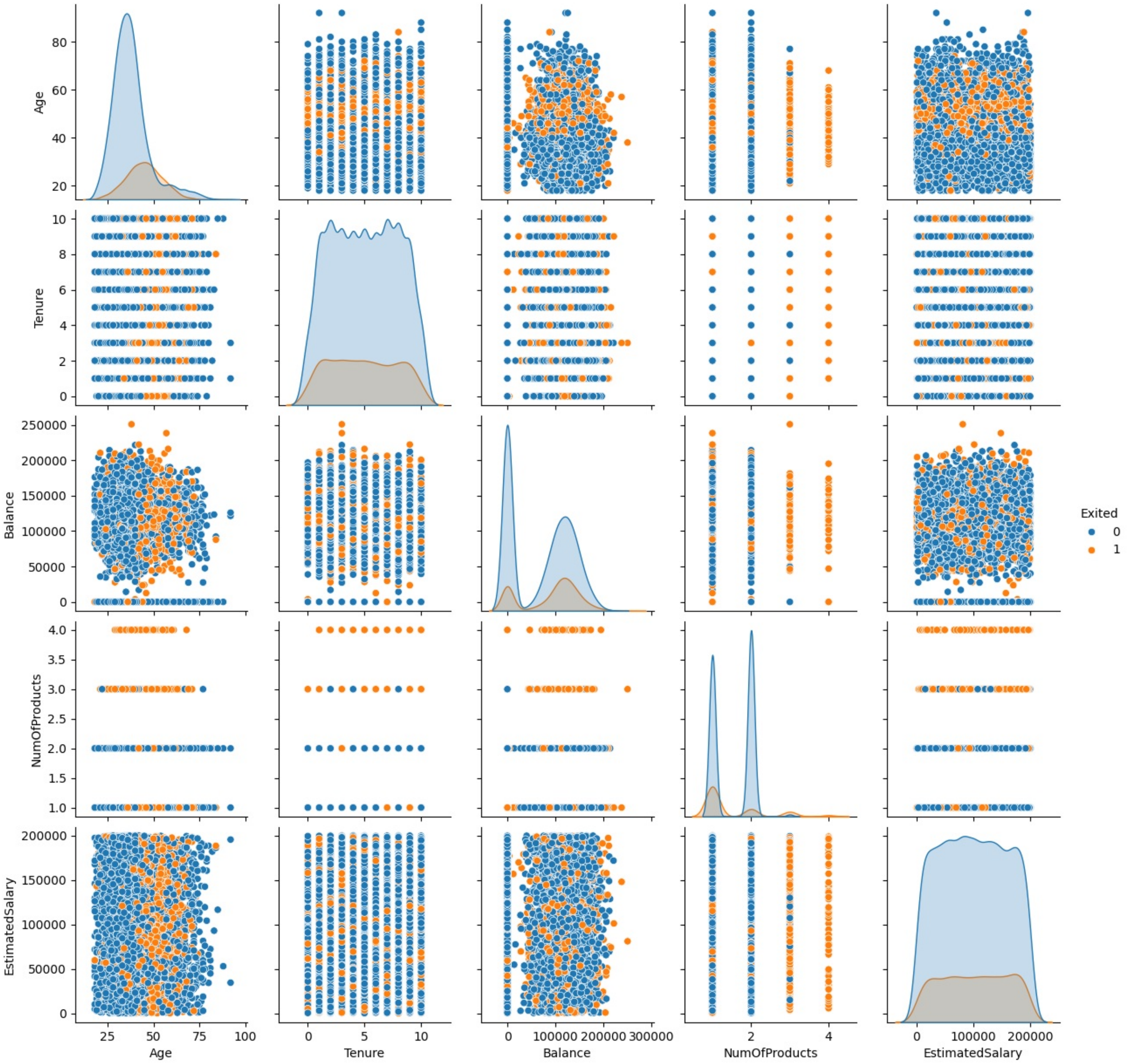
```
# Generating heatmap for all numerical veriables
# defining the size of the plot
plt.figure(figsize=(12, 7))

numerical_columns = ['Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']

# plotting the heatmap for correlation
sns.heatmap(data[numerical_columns].corr(),annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
);
```



In [ ]:

```
# Generating pairplot plot for continius veriables with hue='Exited' as target variable.
plt.figure(figsize=(12, 7))
sns.pairplot(data, vars=numerical_columns , hue='Exited', diag_kind='kde');
plt.show()
```

```
<Figure size 1200x700 with 0 Axes>
```



Observations

- **Balance** and **NumOfProducts** have a moderate negative correlation (-0.30). Customers with higher balances tend to have fewer products, which is an interesting pattern.
- Other variables, such as **Age**, **Tenure**, and **EstimatedSalary**, have near-zero correlations with one another.

Pairplot:

- For **Age**, customers who exited seem to be concentrated more around the age group of 40-60, while customers who stayed are distributed over a broader range.
- **Balance** has a noticeable difference where a significant proportion of exited customers have higher balances compared to customers who stayed.
- **NumOfProducts** shows that exited customers are more concentrated around fewer products, primarily 1 or 2.
- **EstimatedSalary** appears uniformly distributed for both exited and non-exited customers, indicating it might not be a significant differentiator.

Data Preprocessing

Checking missing or duplicate values.

In [ ]:
```
data.isnull().sum()
```

Out[ ]:

	0
RowNumber	0
CustomerId	0
Surname	0
CreditScore	0
Geography	0
Gender	0
Age	0
Tenure	0
Balance	0
NumOfProducts	0
HasCrCard	0
IsActiveMember	0
EstimatedSalary	0
Exited	0

dtype: int64

In [ ]:
```
data.duplicated().sum()
```

Out[ ]:
```
0
```

Observations

- There are no missing values in the data-set.
- There is no duplicated values in the data-set.

Dropping unnecessary columns / variables

In [ ]:
```
# Dropping the RowNumber & the CustomerId column as they are simply ID columns and have no statistical signific
data.drop(columns=['RowNumber','CustomerId'],inplace=True)
# Over the EDA analysis, we decided that the Surname column can be dropped as any observed differences in churn
data.drop(columns=['Surname'],inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CreditScore      10000 non-null  int64
 1   Geography        10000 non-null  object
 2   Gender           10000 non-null  object
 3   Age              10000 non-null  int64
 4   Tenure           10000 non-null  int64
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64
 7   HasCrCard        10000 non-null  int64
 8   IsActiveMember   10000 non-null  int64
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

Dummy Variable Creation

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CreditScore      10000 non-null  int64  
 1   Geography        10000 non-null  object 
 2   Gender           10000 non-null  object 
 3   Age              10000 non-null  int64  
 4   Tenure           10000 non-null  int64  
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64  
 7   HasCrCard        10000 non-null  int64  
 8   IsActiveMember   10000 non-null  int64  
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64  
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

In [ ]:
```
# Creating Dummy variables using get_dummies and dropping the first (One-Hot-encoding)
data = pd.get_dummies(data=data,columns=data.select_dtypes(include=["object"]).columns.tolist(),drop_first=True
data = data.astype(float)
data.info()
```

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

Train-validation-test Split

In [ ]:
```
# Splitting the data between indipendent variables and target variable and making copies - i.e. X & y
# Leaving data as intact
y = data["Exited"].copy()
X = data.drop("Exited" , axis=1).copy()
```

In [ ]:
```
# Splitting the data into train, validation and test category with ratio 70:15:15 with stratify=y
# Initial split: train (70%) and temp (30%, for validation + test)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)

# Split temp into validation (15%) and test (15%)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=1)

# Making sure that the split data sustains a balance distribution on Attrition_Flag 0(Existing Customer) and 1(
print("Shape of training set:", X_train.shape)
print("Shape of validation set:", X_val.shape)
print("Shape of test set:", X_test.shape, '\n')
print("Percentage of classes in training set:")
print(100*y_train.value_counts(normalize=True), '\n')
print("Percentage of classes in validation set:")
print(100*y_val.value_counts(normalize=True), '\n')
print("Percentage of classes in test set:")
print(100*y_test.value_counts(normalize=True))
```

```
Shape of training set: (7000, 11)
Shape of validation set: (1500, 11)
Shape of test set: (1500, 11)

Percentage of classes in training set:
Exited
0.0    79.628571
1.0    20.371429
Name: proportion, dtype: float64

Percentage of classes in validation set:
Exited
0.0    79.8
1.0    20.2
Name: proportion, dtype: float64

Percentage of classes in test set:
Exited
0.0    79.466667
1.0    20.533333
Name: proportion, dtype: float64
```

Data Normalization

In [ ]:
```
col_names_to_normalize = ['CreditScore','Age','Balance','EstimatedSalary'];
sc = StandardScaler()

X_train[col_names_to_normalize] = sc.fit_transform(X_train[col_names_to_normalize])
X_val[col_names_to_normalize] = sc.transform(X_val[col_names_to_normalize])
X_test[col_names_to_normalize] = sc.transform(X_test[col_names_to_normalize])
print(X_train[col_names_to_normalize].describe())
```

```
        CreditScore           Age       Balance  EstimatedSalary
count  7.000000e+03  7.000000e+03  7.000000e+03     7.000000e+03
mean  -2.984279e-16  2.773654e-16  3.755726e-17     1.893089e-16
std    1.000071e+00  1.000071e+00  1.000071e+00     1.000071e+00
min   -3.099859e+00 -1.979491e+00 -1.215278e+00    -1.741260e+00
25%   -6.832038e-01 -6.551276e-01 -1.215278e+00    -8.427693e-01
50%    8.744516e-03 -1.821406e-01  3.306276e-01    -9.364191e-03
75%    6.903653e-01  4.800413e-01  8.217812e-01     8.475445e-01
max    2.063934e+00  5.020717e+00  2.788969e+00     1.757709e+00
```

Data Serialization

Store Data

In [ ]:
```
# Saving all the data sets so that we can read them back if required, later.
X_train.to_csv(path+'X_train')
X_val.to_csv(path+'X_val')
X_test.to_csv(path+'X_test')
y_train.to_csv(path+'y_train')
y_val.to_csv(path+'y_val')
y_test.to_csv(path+'y_test')
```

Reload Data

In [ ]:
```
# Read splitted datasets not needed when session is current.
X_train=pd.read_csv(path+'X_train', index_col=0)
X_val=pd.read_csv(path+'X_val', index_col=0)
X_test=pd.read_csv(path+'X_test', index_col=0)
y_train=pd.read_csv(path+'y_train', index_col=0)
y_val=pd.read_csv(path+'y_val', index_col=0)
y_test=pd.read_csv(path+'y_test', index_col=0)
```

Model Building

Model Evaluation Criterion

For the customer churn prediction project, the primary **Model Evaluation Criteria** should align with the business objective of minimizing customer churn by accurately identifying customers at risk of leaving.

1. Primary Metric: Recall for the Positive Class

- Recall measures how well the model identifies customers who are likely to churn (true positives).
- Missing a customer who is likely to churn (false negative) is more costly for the business than incorrectly identifying a loyal customer as a churn risk (false positive).

2. Secondary Metric: Precision

- While recall is prioritized, precision ensures that the bank does not spend too many resources on false positives.

3. F1-Score

- F1-Score is the harmonic mean of precision and recall.

Priorities for evaluation

- **Recall** > **Precision** > **F1-Score** > **ROC-AUC** > **Accuracy**

This ensures the model focuses on identifying churners (maximizing recall) while keeping the false positive rate under control (precision).

Model evaluation utility functions & data-structure to store evaluation metrics

In [ ]:
```
#Defining the columns of the dataframe which are nothing but the hyper parameters and the metrics.
columns = ["# hidden layers","# neurons - hidden layer","activation function - hidden layer ","# epochs","batch
           "weight initializer","regularization","train loss","validation loss","train recall","validation reca

#Creating a pandas dataframe.
results = pd.DataFrame(columns=columns)

eval_metric = pd.DataFrame(columns=["train-recall", "validation-recall", 'train-f1-score', 'validation-f1-score
```

In [ ]:
```
def plot(history, name):
    """
    Function to plot loss/accuracy

    history: an object which stores the metrics and losses.
    name: can be one of Loss or Accuracy
    """
    fig, ax = plt.subplots() #Creating a subplot with figure and axes.
    plt.plot(history.history[name]) #Plotting the train accuracy or train loss
    plt.plot(history.history['val_'+name]) #Plotting the validation accuracy or validation loss

    plt.title('Model ' + name.capitalize()) #Defining the title of the plot.
    plt.ylabel(name.capitalize()) #Capitalizing the first letter.
    plt.xlabel('Epoch') #Defining the label for the x-axis.
    fig.legend(['Train', 'Validation'], loc="outside right upper") #Defining the legend, loc controls the posit
```

In [ ]:
```
def plot_confusion_matrix(model, X, y, d_type, model_name):
    """
    To plot the confusion_matrix with percentages

    actual_targets: actual target (dependent) variable values
    predicted_targets: predicted target (dependent) variable values
    """
    print(X.shape)
    print(y.shape)

    print('Confusion Matrix - ' + model_name + ' - ' + d_type)
    y_pred = model.predict(X)
    y_pred = (y_pred > 0.5)

    if d_type == 'train' or d_type == 'TRAIN' or d_type == 'training' or d_type == 'TRAINING':
      eval_metric.loc[model_name, 'train-recall'] = recall_score(y, y_pred)
    else:
      eval_metric.loc[model_name, 'validation-recall'] = recall_score(y, y_pred)

    if d_type == 'train' or d_type == 'TRAIN' or d_type == 'training' or d_type == 'TRAINING':
      eval_metric.loc[model_name, 'train-f1-score'] = f1_score(y, y_pred)
    else:
      eval_metric.loc[model_name, 'validation-f1-score'] = f1_score(y, y_pred)

    if d_type == 'train' or d_type == 'TRAIN' or d_type == 'training' or d_type == 'TRAINING':
      eval_metric.loc[model_name, 'train-precision'] = precision_score(y, y_pred)
    else:
      eval_metric.loc[model_name, 'validation-precision'] = precision_score(y, y_pred)

    if d_type == 'train' or d_type == 'TRAIN' or d_type == 'training' or d_type == 'TRAINING':
      eval_metric.loc[model_name, 'train-roc-auc'] = roc_auc_score(y, y_pred)
    else:
      eval_metric.loc[model_name, 'validation-roc-auc'] = roc_auc_score(y, y_pred)

    if d_type == 'train' or d_type == 'TRAIN' or d_type == 'training' or d_type == 'TRAINING':
      eval_metric.loc[model_name, 'train-accurecy'] = accuracy_score(y, y_pred)
    else:
      eval_metric.loc[model_name, 'validation-accurecy'] = accuracy_score(y, y_pred)
```

```
    cm = confusion_matrix(y, y_pred)
    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")
    plt.show()
    cr = classification_report(y, y_pred)
    print(cr)
```

In [ ]:
```
#Fixing the seed for random number generators so that we can ensure we receive the same output everytime
# Set the seed using keras.utils.set_random_seed. This will set:
# 1) `numpy` seed
# 2) backend random seed
# 3) `python` random seed
seed_value = 1
tf.random.set_seed(seed_value)
keras.utils.set_random_seed(seed_value)
# If using TensorFlow, this will make GPU ops as deterministic as possible,
# but it might affect the overall performance
tf.config.experimental.enable_op_determinism()
```

Neural Network with SGD Optimizer

In [ ]:
```
tf.keras.backend.clear_session() #Clearing the session.
#Initializing the neural network
model_name = 'NN with SGD';
batch_size=50
epochs=100
model_0 = Sequential()
input_dimention = X_train.shape[1]
print("Input Dimention:", input_dimention)
# Adding hidden layers
model_0.add(Dense(64, activation='relu', input_dim = input_dimention))
model_0.add(Dense(32, activation='tanh'))
# Adding output layers
model_0.add(Dense(1, activation = 'sigmoid'))
optimizer = keras.optimizers.SGD()  # defining SGD as the optimizer to be used
model_0.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["recall"])
model_0.summary()
```

```
Input Dimention: 11
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 1)	33

Total params: 2,881 (11.25 KB)

Trainable params: 2,881 (11.25 KB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
start = time.time()
# Training the model
history = model_0.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs, ve
end=time.time()
```

```
Epoch 1/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.5428 - recall: 0.0018 - val_loss: 0.4887 - val_recall: 0.0033
Epoch 2/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4907 - recall: 0.0012 - val_loss: 0.4572 - val_recall: 0.0033
Epoch 3/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4635 - recall: 0.0096 - val_loss: 0.4399 - val_recall: 0.0297
Epoch 4/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4491 - recall: 0.0455 - val_loss: 0.4305 - val_recall: 0.0660
Epoch 5/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4413 - recall: 0.0949 - val_loss: 0.4249 - val_recall: 0.1155
Epoch 6/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4364 - recall: 0.1328 - val_loss: 0.4213 - val_recall: 0.1419
Epoch 7/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4330 - recall: 0.1630 - val_loss: 0.4187 - val_recall: 0.1650
```

```
Epoch 8/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4304 - recall: 0.1840 - val_loss: 0.4167 - val_recall: 0.1716
Epoch 9/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4283 - recall: 0.1967 - val_loss: 0.4151 - val_recall: 0.1848
Epoch 10/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4265 - recall: 0.2064 - val_loss: 0.4137 - val_recall: 0.1914
Epoch 11/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4249 - recall: 0.2145 - val_loss: 0.4126 - val_recall: 0.2013
Epoch 12/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4235 - recall: 0.2204 - val_loss: 0.4116 - val_recall: 0.2079
Epoch 13/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4223 - recall: 0.2247 - val_loss: 0.4107 - val_recall: 0.2178
Epoch 14/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4212 - recall: 0.2281 - val_loss: 0.4099 - val_recall: 0.2244
Epoch 15/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4201 - recall: 0.2307 - val_loss: 0.4092 - val_recall: 0.2343
Epoch 16/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4192 - recall: 0.2348 - val_loss: 0.4085 - val_recall: 0.2442
Epoch 17/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4182 - recall: 0.2360 - val_loss: 0.4078 - val_recall: 0.2508
Epoch 18/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4174 - recall: 0.2464 - val_loss: 0.4071 - val_recall: 0.2574
Epoch 19/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4165 - recall: 0.2541 - val_loss: 0.4065 - val_recall: 0.2607
Epoch 20/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4157 - recall: 0.2545 - val_loss: 0.4059 - val_recall: 0.2640
Epoch 21/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4149 - recall: 0.2563 - val_loss: 0.4053 - val_recall: 0.2805
Epoch 22/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4142 - recall: 0.2604 - val_loss: 0.4047 - val_recall: 0.2805
Epoch 23/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4135 - recall: 0.2639 - val_loss: 0.4041 - val_recall: 0.2871
Epoch 24/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4127 - recall: 0.2641 - val_loss: 0.4035 - val_recall: 0.2937
Epoch 25/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4120 - recall: 0.2681 - val_loss: 0.4030 - val_recall: 0.2937
Epoch 26/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4113 - recall: 0.2688 - val_loss: 0.4024 - val_recall: 0.2970
Epoch 27/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4107 - recall: 0.2743 - val_loss: 0.4018 - val_recall: 0.3003
Epoch 28/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4100 - recall: 0.2781 - val_loss: 0.4012 - val_recall: 0.3003
Epoch 29/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4093 - recall: 0.2791 - val_loss: 0.4007 - val_recall: 0.3003
Epoch 30/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4086 - recall: 0.2794 - val_loss: 0.4001 - val_recall: 0.3003
Epoch 31/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4080 - recall: 0.2831 - val_loss: 0.3995 - val_recall: 0.3036
Epoch 32/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4073 - recall: 0.2858 - val_loss: 0.3990 - val_recall: 0.3036
Epoch 33/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4066 - recall: 0.2900 - val_loss: 0.3984 - val_recall: 0.3069
Epoch 34/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4059 - recall: 0.2924 - val_loss: 0.3978 - val_recall: 0.3102
Epoch 35/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4052 - recall: 0.2929 - val_loss: 0.3972 - val_recall: 0.3102
Epoch 36/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4045 - recall: 0.2976 - val_loss: 0.3966 - val_recall: 0.3135
Epoch 37/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4039 - recall: 0.3013 - val_loss: 0.3960 - val_recall: 0.3135
Epoch 38/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4032 - recall: 0.3071 - val_loss: 0.3954 - val_recall: 0.3201
Epoch 39/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4025 - recall: 0.3052 - val_loss: 0.3947 - val_recall: 0.3201
Epoch 40/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4018 - recall: 0.3064 - val_loss: 0.3941 - val_recall: 0.3234
Epoch 41/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4011 - recall: 0.3067 - val_loss: 0.3935 - val_recall: 0.3300
Epoch 42/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4004 - recall: 0.3096 - val_loss: 0.3929 - val_recall: 0.3399
Epoch 43/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3997 - recall: 0.3110 - val_loss: 0.3923 - val_recall: 0.3432
Epoch 44/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3990 - recall: 0.3147 - val_loss: 0.3916 - val_recall: 0.3432
Epoch 45/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3982 - recall: 0.3214 - val_loss: 0.3910 - val_recall: 0.3465
Epoch 46/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3975 - recall: 0.3260 - val_loss: 0.3903 - val_recall: 0.3465
Epoch 47/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3968 - recall: 0.3285 - val_loss: 0.3897 - val_recall: 0.3498
Epoch 48/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3961 - recall: 0.3352 - val_loss: 0.3890 - val_recall: 0.3564
Epoch 49/100
```

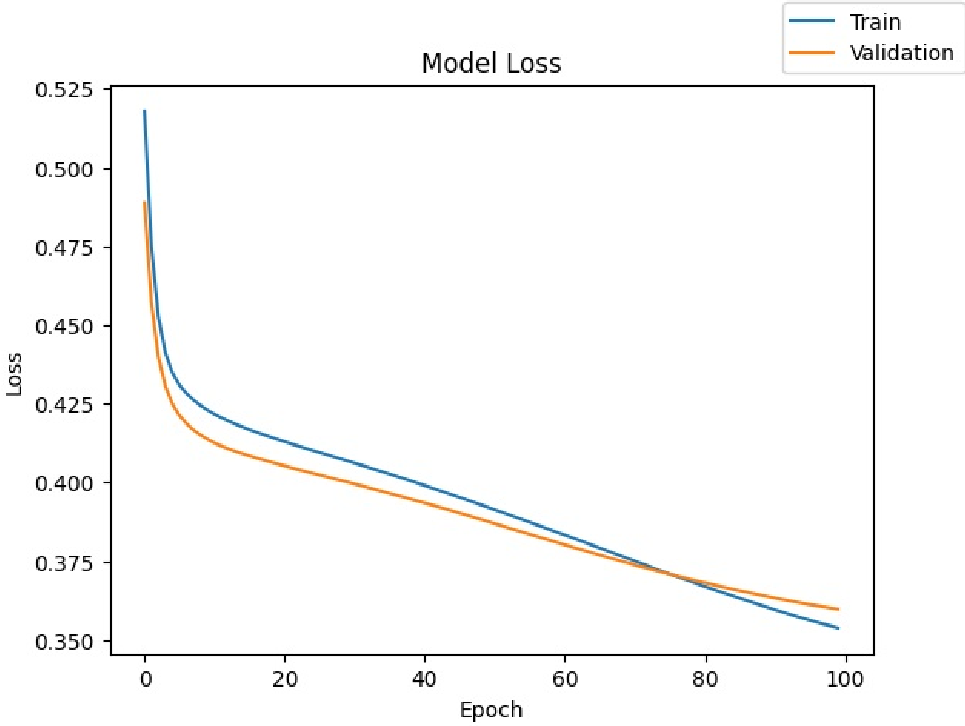
```
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3953 - recall: 0.3359 - val_loss: 0.3883 - val_recall: 0.3597
Epoch 50/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3946 - recall: 0.3383 - val_loss: 0.3876 - val_recall: 0.3630
Epoch 51/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3938 - recall: 0.3354 - val_loss: 0.3869 - val_recall: 0.3696
Epoch 52/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3931 - recall: 0.3382 - val_loss: 0.3863 - val_recall: 0.3696
Epoch 53/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3923 - recall: 0.3395 - val_loss: 0.3856 - val_recall: 0.3729
Epoch 54/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3916 - recall: 0.3430 - val_loss: 0.3849 - val_recall: 0.3729
Epoch 55/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3908 - recall: 0.3450 - val_loss: 0.3842 - val_recall: 0.3729
Epoch 56/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3900 - recall: 0.3469 - val_loss: 0.3836 - val_recall: 0.3729
Epoch 57/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3893 - recall: 0.3495 - val_loss: 0.3829 - val_recall: 0.3762
Epoch 58/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3885 - recall: 0.3479 - val_loss: 0.3822 - val_recall: 0.3762
Epoch 59/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3877 - recall: 0.3493 - val_loss: 0.3816 - val_recall: 0.3795
Epoch 60/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3869 - recall: 0.3507 - val_loss: 0.3809 - val_recall: 0.3795
Epoch 61/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3862 - recall: 0.3512 - val_loss: 0.3802 - val_recall: 0.3828
Epoch 62/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3854 - recall: 0.3500 - val_loss: 0.3796 - val_recall: 0.3795
Epoch 63/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3846 - recall: 0.3537 - val_loss: 0.3789 - val_recall: 0.3795
Epoch 64/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3838 - recall: 0.3540 - val_loss: 0.3783 - val_recall: 0.3828
Epoch 65/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3830 - recall: 0.3532 - val_loss: 0.3776 - val_recall: 0.3861
Epoch 66/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3822 - recall: 0.3551 - val_loss: 0.3770 - val_recall: 0.3861
Epoch 67/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3813 - recall: 0.3567 - val_loss: 0.3763 - val_recall: 0.3861
Epoch 68/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3805 - recall: 0.3628 - val_loss: 0.3757 - val_recall: 0.3861
Epoch 69/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3797 - recall: 0.3650 - val_loss: 0.3751 - val_recall: 0.3861
Epoch 70/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3789 - recall: 0.3713 - val_loss: 0.3745 - val_recall: 0.3861
Epoch 71/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3780 - recall: 0.3708 - val_loss: 0.3739 - val_recall: 0.3927
Epoch 72/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3772 - recall: 0.3782 - val_loss: 0.3733 - val_recall: 0.3927
Epoch 73/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3763 - recall: 0.3785 - val_loss: 0.3727 - val_recall: 0.3927
Epoch 74/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3755 - recall: 0.3818 - val_loss: 0.3721 - val_recall: 0.3927
Epoch 75/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3746 - recall: 0.3892 - val_loss: 0.3715 - val_recall: 0.3927
Epoch 76/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3738 - recall: 0.3894 - val_loss: 0.3709 - val_recall: 0.3927
Epoch 77/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3730 - recall: 0.3915 - val_loss: 0.3704 - val_recall: 0.3894
Epoch 78/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3722 - recall: 0.3950 - val_loss: 0.3698 - val_recall: 0.3894
Epoch 79/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3714 - recall: 0.3987 - val_loss: 0.3692 - val_recall: 0.3894
Epoch 80/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3706 - recall: 0.4006 - val_loss: 0.3687 - val_recall: 0.3894
Epoch 81/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3698 - recall: 0.4016 - val_loss: 0.3682 - val_recall: 0.3927
Epoch 82/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3690 - recall: 0.4021 - val_loss: 0.3676 - val_recall: 0.3993
Epoch 83/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3682 - recall: 0.4024 - val_loss: 0.3671 - val_recall: 0.3993
Epoch 84/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3674 - recall: 0.4049 - val_loss: 0.3666 - val_recall: 0.3960
Epoch 85/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3666 - recall: 0.4083 - val_loss: 0.3661 - val_recall: 0.3993
Epoch 86/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3658 - recall: 0.4086 - val_loss: 0.3657 - val_recall: 0.3993
Epoch 87/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3651 - recall: 0.4114 - val_loss: 0.3652 - val_recall: 0.4059
Epoch 88/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3643 - recall: 0.4126 - val_loss: 0.3647 - val_recall: 0.4059
Epoch 89/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3635 - recall: 0.4174 - val_loss: 0.3643 - val_recall: 0.4092
Epoch 90/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3628 - recall: 0.4157 - val_loss: 0.3638 - val_recall: 0.4059
```

```
Epoch 91/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3620 - recall: 0.4213 - val_loss: 0.3634 - val_recall: 0.4092
Epoch 92/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3613 - recall: 0.4210 - val_loss: 0.3629 - val_recall: 0.4125
Epoch 93/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3605 - recall: 0.4224 - val_loss: 0.3625 - val_recall: 0.4125
Epoch 94/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3598 - recall: 0.4235 - val_loss: 0.3621 - val_recall: 0.4158
Epoch 95/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3591 - recall: 0.4256 - val_loss: 0.3617 - val_recall: 0.4158
Epoch 96/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3584 - recall: 0.4256 - val_loss: 0.3613 - val_recall: 0.4224
Epoch 97/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3577 - recall: 0.4234 - val_loss: 0.3609 - val_recall: 0.4257
Epoch 98/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3570 - recall: 0.4223 - val_loss: 0.3606 - val_recall: 0.4290
Epoch 99/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3564 - recall: 0.4268 - val_loss: 0.3602 - val_recall: 0.4290
Epoch 100/100
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3557 - recall: 0.4277 - val_loss: 0.3598 - val_recall: 0.4323
```

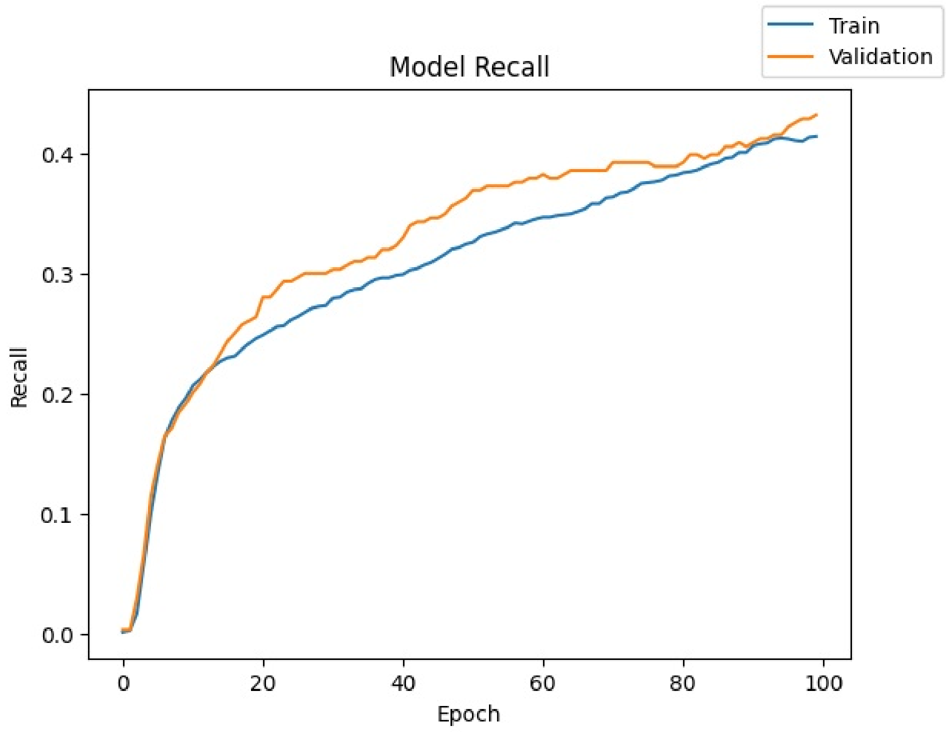
In [ ]:
```python
print("Time taken in seconds: ",end-start)
```

```
Time taken in seconds:  26.363971233336792
```

In [ ]:
```python
plot(history,'loss')
```

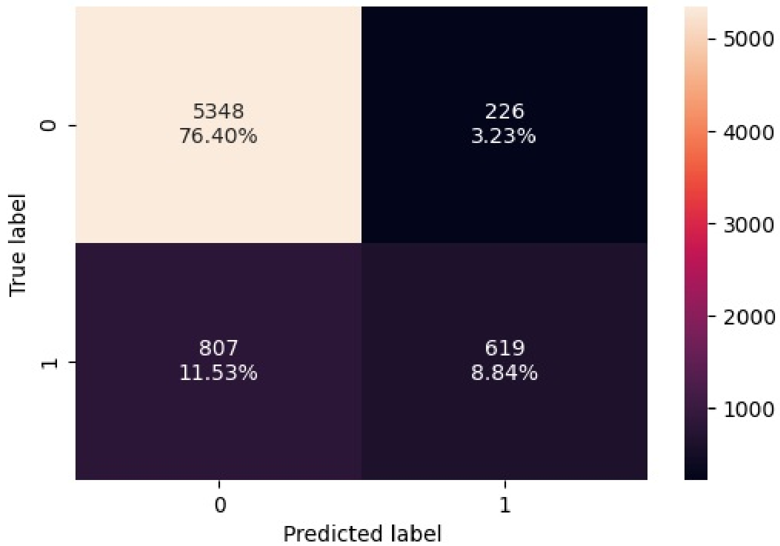


In [ ]:
```python
plot(history,'recall')
```



In [ ]:
```
plot_confusion_matrix(model_0, X_train, y_train, 'train', model_name)
```

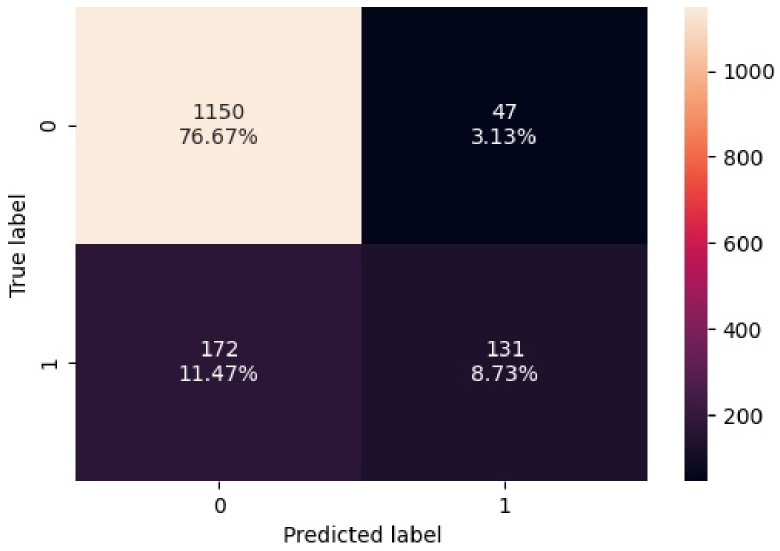
```
(7000, 11)
(7000, 1)
Confusion Matrix - NN with SGD - train
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.87      0.96      0.91      5574
         1.0       0.73      0.43      0.55      1426

    accuracy                           0.85      7000
   macro avg       0.80      0.70      0.73      7000
weighted avg       0.84      0.85      0.84      7000
```

In [ ]:
```
plot_confusion_matrix(model_0, X_val, y_val, 'validation', model_name)
```

```
(1500, 11)
(1500, 1)
Confusion Matrix - NN with SGD - validation
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.87      0.96      0.91      1197
         1.0       0.74      0.43      0.54       303

    accuracy                           0.85      1500
   macro avg       0.80      0.70      0.73      1500
weighted avg       0.84      0.85      0.84      1500
```

In [ ]:
```
results.drop([0], inplace=True, errors='ignore')
results.loc[0] = [2,[64,32],["relu","tanh"],100,50,"sgd",[0.001, "-"],"xavier","-",history.history["loss"][-1],l
results
```

Out[ ]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	learning rate, momentum	weight initializer	regularization	train loss	validation loss	train recall	vali
0	2	[64, 32]	[relu, tanh]	100	50	sgd	[0.001, -]	xavier	-	0.353841	0.359846	0.414446	0.4

In [ ]:
```
eval_metric
```

Out[ ]:

	train-recall	validation-recall	train-f1-score	validation-f1-score	train-precision	validation-precision	train-roc-auc	validation-roc-auc	train-accurecy	validation-accurecy
NN with SGD	0.434081	0.432343	0.545134	0.544699	0.732544	0.735955	0.696768	0.696539	0.852429	0.854

Observations - (Analysis of the Training History and Metrics)

- The loss for both training and validation decreases steadily over epochs, indicating that the model is learning without overfitting. The training and validation losses converge closely by the end of the training process.
- The recall values for both training and validation sets improve consistently over the epochs, eventually converging with no signs of overfitting. Validation recall closely follows the training recall.
- The model performs well for the majority class (Exited=0) but struggles with the minority class (Exited=1). Recall for the minority class are lower, this is expected in imbalanced datasets.
- Class imbalance is evident, and the model needs strategies like oversampling, undersampling, or class-weight adjustments to improve minority class performance. This behaviour is consistent for both training and validation dataset.
- The precision is relatively high, but recall for the minority class low, indicating that the model is better at avoiding false positives than false negatives. The ROC-AUC is moderate, suggesting potential for improvement in distinguishing between the classes.

Model Performance Improvement

Neural Network with Adam Optimizer

In [ ]:
```
tf.keras.backend.clear_session() #Clearing the session.
#Initializing the neural network
model_name = 'NN with Adam';
batch_size=25
epochs=50
model_1 = Sequential()
input_dimention = X_train.shape[1]
print("Input Dimention:", input_dimention)
# Adding hidden layers
model_1.add(Dense(64, activation='relu', input_dim = input_dimention))
model_1.add(Dense(32, activation='relu'))
# Adding output layer
model_1.add(Dense(1, activation = 'sigmoid'))
optimizer = keras.optimizers.Adam(learning_rate=0.001)   # defining Adam as the optimizer to be used
model_1.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["recall"])
model_1.summary()
```

```
Input Dimention: 11
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 1)	33

Total params: 2,881 (11.25 KB)

Trainable params: 2,881 (11.25 KB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
start = time.time()
history = model_1.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs, v
end=time.time()
```

```
Epoch 1/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.4815 - recall: 0.1433 - val_loss: 0.4071 - val_recall: 0.2409
Epoch 2/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4164 - recall: 0.2635 - val_loss: 0.3970 - val_recall: 0.2871
Epoch 3/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4061 - recall: 0.2932 - val_loss: 0.3878 - val_recall: 0.3201
Epoch 4/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3964 - recall: 0.3311 - val_loss: 0.3798 - val_recall: 0.3333
Epoch 5/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3858 - recall: 0.3580 - val_loss: 0.3734 - val_recall: 0.3465
Epoch 6/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3742 - recall: 0.3871 - val_loss: 0.3681 - val_recall: 0.3762
Epoch 7/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3627 - recall: 0.4225 - val_loss: 0.3642 - val_recall: 0.4026
Epoch 8/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3528 - recall: 0.4451 - val_loss: 0.3607 - val_recall: 0.4191
Epoch 9/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3450 - recall: 0.4474 - val_loss: 0.3578 - val_recall: 0.4323
Epoch 10/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3385 - recall: 0.4672 - val_loss: 0.3546 - val_recall: 0.4290
Epoch 11/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3333 - recall: 0.4831 - val_loss: 0.3526 - val_recall: 0.4356
Epoch 12/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3291 - recall: 0.4951 - val_loss: 0.3509 - val_recall: 0.4455
Epoch 13/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3257 - recall: 0.5031 - val_loss: 0.3491 - val_recall: 0.4521
Epoch 14/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3232 - recall: 0.5136 - val_loss: 0.3489 - val_recall: 0.4488
Epoch 15/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3210 - recall: 0.5190 - val_loss: 0.3482 - val_recall: 0.4554
Epoch 16/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3189 - recall: 0.5274 - val_loss: 0.3474 - val_recall: 0.4554
Epoch 17/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3173 - recall: 0.5288 - val_loss: 0.3471 - val_recall: 0.4587
Epoch 18/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3160 - recall: 0.5342 - val_loss: 0.3467 - val_recall: 0.4620
Epoch 19/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3145 - recall: 0.5395 - val_loss: 0.3468 - val_recall: 0.4587
```

```
Epoch 20/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3135 - recall: 0.5420 - val_loss: 0.3473 - val_recall: 0.4587
Epoch 21/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3122 - recall: 0.5446 - val_loss: 0.3478 - val_recall: 0.4488
Epoch 22/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3112 - recall: 0.5467 - val_loss: 0.3476 - val_recall: 0.4554
Epoch 23/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3104 - recall: 0.5481 - val_loss: 0.3480 - val_recall: 0.4488
Epoch 24/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3097 - recall: 0.5503 - val_loss: 0.3481 - val_recall: 0.4488
Epoch 25/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3089 - recall: 0.5518 - val_loss: 0.3486 - val_recall: 0.4455
Epoch 26/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3081 - recall: 0.5517 - val_loss: 0.3489 - val_recall: 0.4455
Epoch 27/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3075 - recall: 0.5528 - val_loss: 0.3489 - val_recall: 0.4521
Epoch 28/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3069 - recall: 0.5527 - val_loss: 0.3495 - val_recall: 0.4620
Epoch 29/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3062 - recall: 0.5572 - val_loss: 0.3506 - val_recall: 0.4554
Epoch 30/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3057 - recall: 0.5585 - val_loss: 0.3511 - val_recall: 0.4620
Epoch 31/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3050 - recall: 0.5584 - val_loss: 0.3521 - val_recall: 0.4587
Epoch 32/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3044 - recall: 0.5612 - val_loss: 0.3519 - val_recall: 0.4653
Epoch 33/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3037 - recall: 0.5605 - val_loss: 0.3527 - val_recall: 0.4686
Epoch 34/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3032 - recall: 0.5629 - val_loss: 0.3529 - val_recall: 0.4653
Epoch 35/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3027 - recall: 0.5642 - val_loss: 0.3536 - val_recall: 0.4620
Epoch 36/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3019 - recall: 0.5627 - val_loss: 0.3544 - val_recall: 0.4620
Epoch 37/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3017 - recall: 0.5646 - val_loss: 0.3547 - val_recall: 0.4620
Epoch 38/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3007 - recall: 0.5649 - val_loss: 0.3558 - val_recall: 0.4653
Epoch 39/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3001 - recall: 0.5642 - val_loss: 0.3560 - val_recall: 0.4620
Epoch 40/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2995 - recall: 0.5681 - val_loss: 0.3564 - val_recall: 0.4620
Epoch 41/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2989 - recall: 0.5645 - val_loss: 0.3569 - val_recall: 0.4653
Epoch 42/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2982 - recall: 0.5668 - val_loss: 0.3578 - val_recall: 0.4653
Epoch 43/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2977 - recall: 0.5707 - val_loss: 0.3580 - val_recall: 0.4620
Epoch 44/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2970 - recall: 0.5709 - val_loss: 0.3581 - val_recall: 0.4686
Epoch 45/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2959 - recall: 0.5749 - val_loss: 0.3587 - val_recall: 0.4653
Epoch 46/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2954 - recall: 0.5761 - val_loss: 0.3594 - val_recall: 0.4653
Epoch 47/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2947 - recall: 0.5783 - val_loss: 0.3599 - val_recall: 0.4686
Epoch 48/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2941 - recall: 0.5767 - val_loss: 0.3604 - val_recall: 0.4719
Epoch 49/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2936 - recall: 0.5820 - val_loss: 0.3613 - val_recall: 0.4719
Epoch 50/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2930 - recall: 0.5823 - val_loss: 0.3617 - val_recall: 0.4719
```

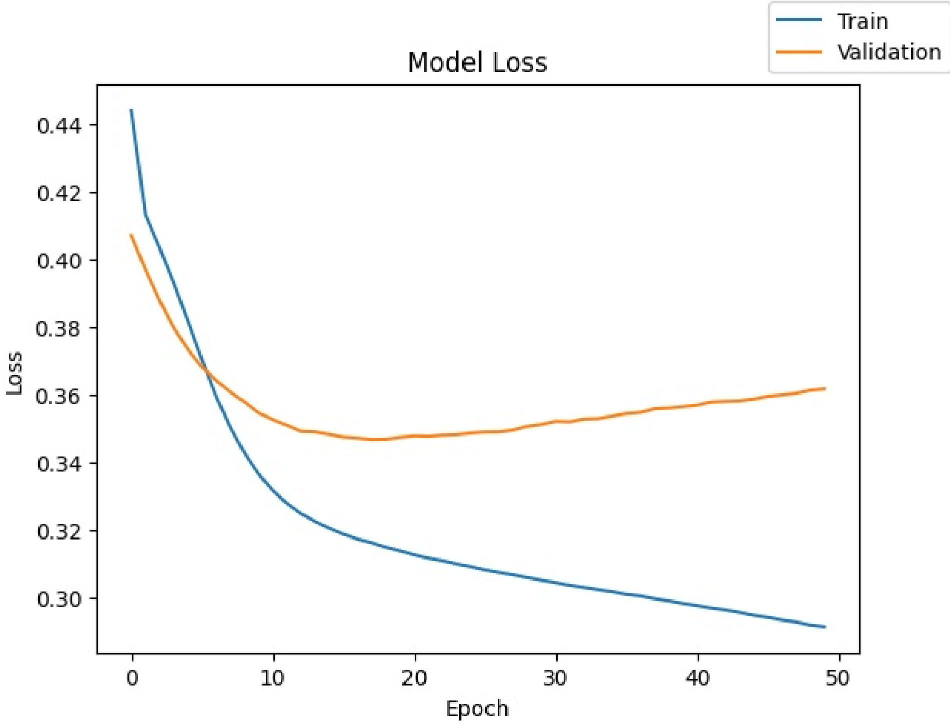
In [ ]:

```
print("Time taken in seconds: ",end-start)
```

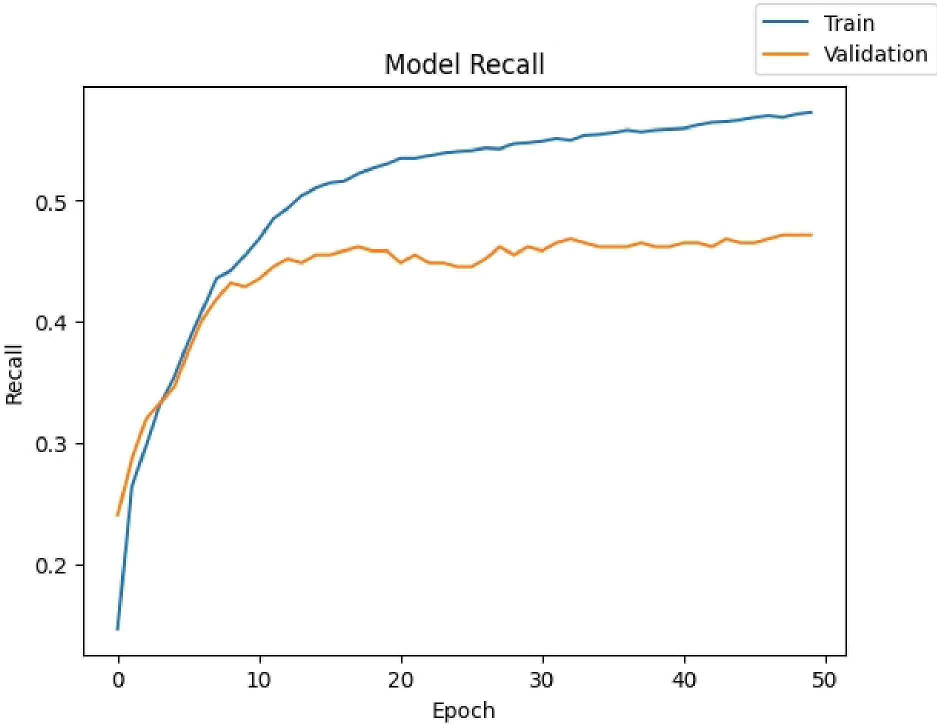
```
Time taken in seconds:  25.948427438735962
```

In [ ]:

```
plot(history,'loss')
```

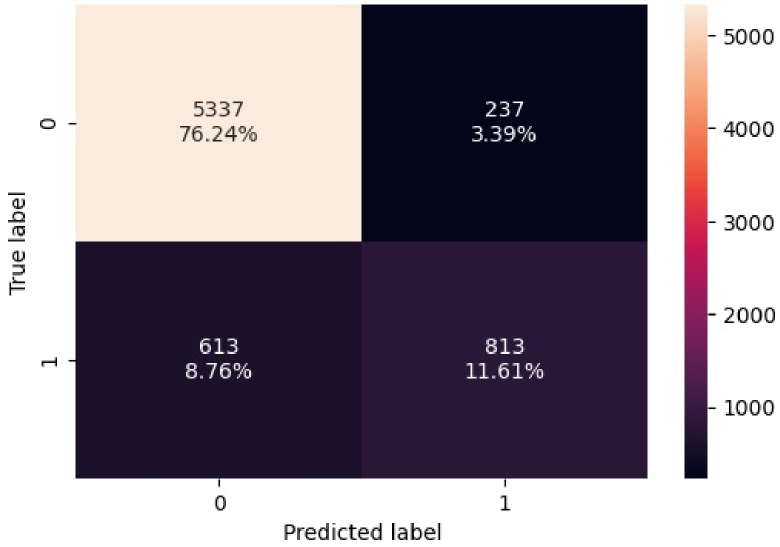


In [ ]:
```
plot(history,'recall')
```



In [ ]:
```
plot_confusion_matrix(model_1, X_train, y_train, 'train', model_name)
```

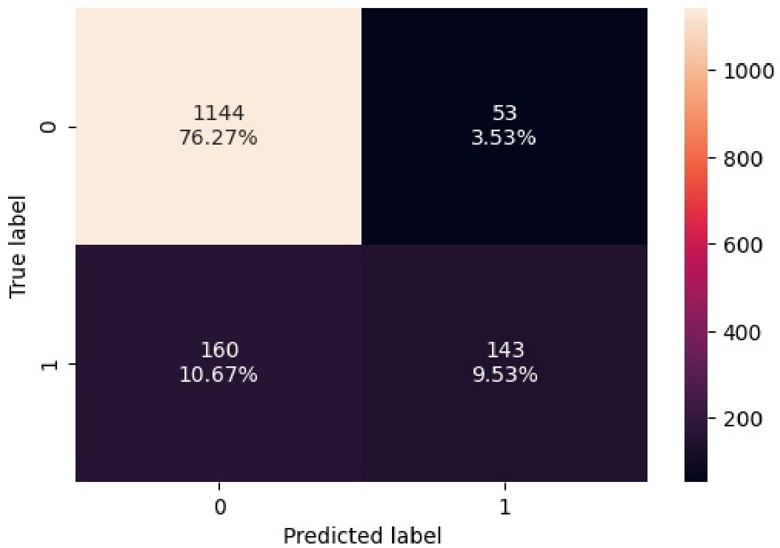
```
(7000, 11)
(7000, 1)
Confusion Matrix - NN with Adam - train
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.90      0.96      0.93      5574
         1.0       0.77      0.57      0.66      1426

    accuracy                           0.88      7000
   macro avg       0.84      0.76      0.79      7000
weighted avg       0.87      0.88      0.87      7000
```

In [ ]:
```
plot_confusion_matrix(model_1, X_val, y_val, 'validation', model_name)
```

```
(1500, 11)
(1500, 1)
Confusion Matrix - NN with Adam - validation
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```



```
              precision    recall  f1-score   support

         0.0       0.88      0.96      0.91      1197
         1.0       0.73      0.47      0.57       303

    accuracy                           0.86      1500
   macro avg       0.80      0.71      0.74      1500
weighted avg       0.85      0.86      0.85      1500
```

In [ ]:

```
results.drop([1], inplace=True, errors='ignore')
results.loc[1] = [2,[64, 32],["relu","relu"],50,25,"Adam",[0.001, "-"],"xavier","-",history.history["loss"][-1]
results
```

Out[ ]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	learning rate, momentum	weight initializer	regularization	train loss	validation loss	train recall	vali
0	2	[64, 32]	[relu, tanh]	100	50	sgd	[0.001, -]	xavier	-	0.353841	0.359846	0.414446	0.4
1	2	[64, 32]	[relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.291288	0.361662	0.572931	0.4

In [ ]:

```
eval_metric
```

Out[ ]:

	train-recall	validation-recall	train-f1-score	validation-f1-score	train-precision	validation-precision	train-roc-auc	validation-roc-auc	train-accurecy	validation-accurecy
NN with SGD	0.434081	0.432343	0.545134	0.544699	0.732544	0.735955	0.696768	0.696539	0.852429	0.854
NN with Adam	0.570126	0.471947	0.656704	0.573146	0.774286	0.729592	0.763804	0.713835	0.878571	0.858

Observations

- Loss and recall graph
  - The training loss decreases smoothly and continuously, showing effective convergence during training.
  - The validation loss flattens early and begins to increase slightly toward the end, suggesting overfitting.
  - The training recall curve rises steadily and stabilizes close to the end of the training cycle.
  - The divergence between training and validation loss/recal curves suggests overfitting.
- Evaluation metrics
  - The model shows an improvement in both train recall and validation recall compared to the first model using SGD.
  - Validation accuracy and precision have slightly increased, demonstrating better overall performance.
  - ROC-AUC metrics for both train and validation indicate moderate improvement over the previous model.

Neural Network with Adam Optimizer and Dropout

In [ ]:

```
tf.keras.backend.clear_session() #Clearing the session.
#Initializing the neural network
model_name = 'NN with Adam With Dropout';
batch_size=25
epochs=50
model_2 = Sequential()
input_dimention = X_train.shape[1]
print("Input Dimention:", input_dimention)
# Adding hidden layers and dropouts ratio
model_2.add(Dense(64, activation='relu', input_dim = input_dimention, kernel_regularizer=tf.keras.regularizers.
# Add dropout with ratio of 0.3
model_2.add(Dropout(0.3))
model_2.add(Dense(32, activation='relu'))
# Add dropout with ratio of 0.2
model_2.add(Dropout(0.2))
model_2.add(Dense(16, activation='relu'))
model_2.add(Dense(8, activation='relu'))
# Add dropout with ratio of 0.1
model_2.add(Dropout(0.1))
model_2.add(Dense(4, activation='relu'))
# Adding the output layer
model_2.add(Dense(1, activation = 'sigmoid'))
optimizer = keras.optimizers.Adam(learning_rate=0.001)   # defining Adam as the optimizer to be used
model_2.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["recall"])
model_2.summary()
```

```
Input Dimention: 11
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 8)	136
dropout_2 (Dropout)	(None, 8)	0
dense_4 (Dense)	(None, 4)	36
dense_5 (Dense)	(None, 1)	5

Total params: 3,553 (13.88 KB)

Trainable params: 3,553 (13.88 KB)

Non-trainable params: 0 (0.00 B)

In [ ]:

```
start = time.time()
history = model_2.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs, v
end=time.time()
```

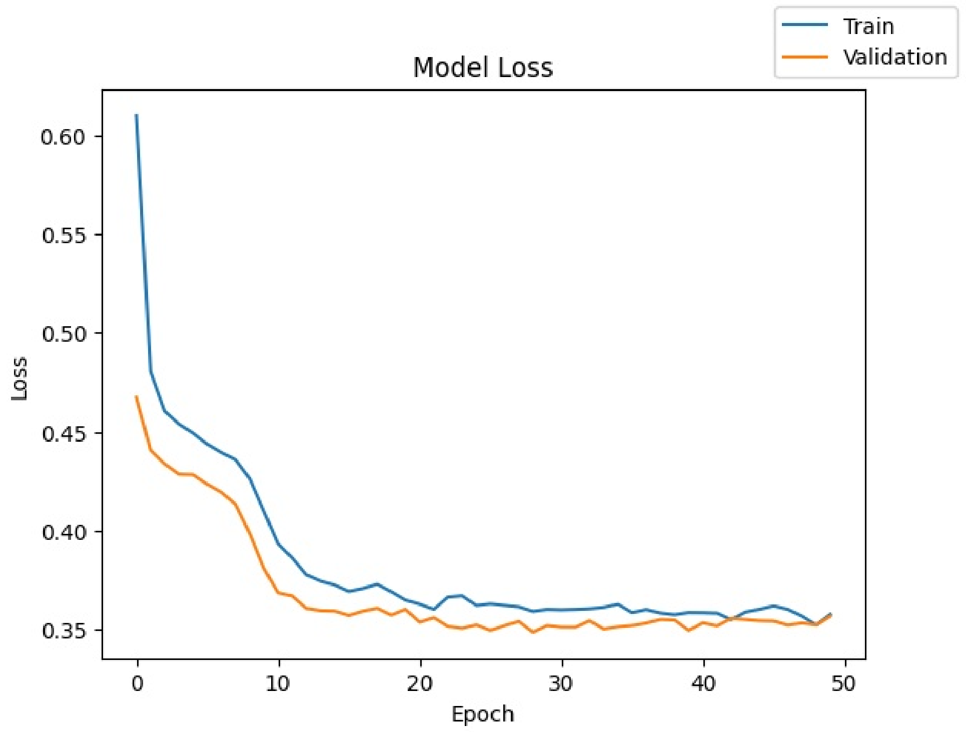
```
Epoch 1/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.7167 - recall: 0.1151 - val_loss: 0.4675 - val_recall: 0.0000
e+00
Epoch 2/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4935 - recall: 2.3139e-04 - val_loss: 0.4408 - val_recall: 0.
0000e+00
Epoch 3/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4640 - recall: 9.7243e-04 - val_loss: 0.4338 - val_recall: 0.
0000e+00
Epoch 4/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4618 - recall: 0.0000e+00 - val_loss: 0.4286 - val_recall: 0.
0000e+00
Epoch 5/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4550 - recall: 0.0016 - val_loss: 0.4284 - val_recall: 0.0000
e+00
Epoch 6/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4502 - recall: 0.0000e+00 - val_loss: 0.4235 - val_recall: 0.
0000e+00
Epoch 7/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4432 - recall: 0.0000e+00 - val_loss: 0.4195 - val_recall: 0.
0000e+00
Epoch 8/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4378 - recall: 0.0000e+00 - val_loss: 0.4134 - val_recall: 0.
0000e+00
Epoch 9/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4295 - recall: 0.0467 - val_loss: 0.3987 - val_recall: 0.4224
Epoch 10/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4139 - recall: 0.3509 - val_loss: 0.3812 - val_recall: 0.4323
Epoch 11/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3966 - recall: 0.4038 - val_loss: 0.3688 - val_recall: 0.4158
Epoch 12/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3887 - recall: 0.4242 - val_loss: 0.3672 - val_recall: 0.4719
Epoch 13/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3769 - recall: 0.4744 - val_loss: 0.3609 - val_recall: 0.4488
Epoch 14/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3734 - recall: 0.4474 - val_loss: 0.3597 - val_recall: 0.4257
Epoch 15/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3732 - recall: 0.4760 - val_loss: 0.3595 - val_recall: 0.4752
Epoch 16/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3731 - recall: 0.4601 - val_loss: 0.3574 - val_recall: 0.3861
Epoch 17/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3736 - recall: 0.4379 - val_loss: 0.3595 - val_recall: 0.4587
Epoch 18/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3735 - recall: 0.4644 - val_loss: 0.3610 - val_recall: 0.5050
Epoch 19/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3692 - recall: 0.4664 - val_loss: 0.3575 - val_recall: 0.4719
Epoch 20/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3701 - recall: 0.4706 - val_loss: 0.3604 - val_recall: 0.5347
Epoch 21/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3629 - recall: 0.4810 - val_loss: 0.3539 - val_recall: 0.4686
Epoch 22/50
```

```
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3650 - recall: 0.4819 - val_loss: 0.3562 - val_recall: 0.4785
Epoch 23/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3658 - recall: 0.4452 - val_loss: 0.3518 - val_recall: 0.4521
Epoch 24/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3715 - recall: 0.4499 - val_loss: 0.3508 - val_recall: 0.4620
Epoch 25/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3666 - recall: 0.4634 - val_loss: 0.3525 - val_recall: 0.4686
Epoch 26/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3647 - recall: 0.4731 - val_loss: 0.3498 - val_recall: 0.4554
Epoch 27/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3589 - recall: 0.4726 - val_loss: 0.3522 - val_recall: 0.4851
Epoch 28/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3627 - recall: 0.4693 - val_loss: 0.3543 - val_recall: 0.4455
Epoch 29/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3613 - recall: 0.4683 - val_loss: 0.3489 - val_recall: 0.4323
Epoch 30/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3643 - recall: 0.4686 - val_loss: 0.3522 - val_recall: 0.4653
Epoch 31/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3587 - recall: 0.4543 - val_loss: 0.3514 - val_recall: 0.4686
Epoch 32/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3644 - recall: 0.4549 - val_loss: 0.3514 - val_recall: 0.4587
Epoch 33/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3642 - recall: 0.4607 - val_loss: 0.3546 - val_recall: 0.4488
Epoch 34/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3588 - recall: 0.4686 - val_loss: 0.3504 - val_recall: 0.4455
Epoch 35/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3630 - recall: 0.4810 - val_loss: 0.3516 - val_recall: 0.4356
Epoch 36/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3588 - recall: 0.4596 - val_loss: 0.3522 - val_recall: 0.4785
Epoch 37/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3652 - recall: 0.4736 - val_loss: 0.3535 - val_recall: 0.4620
Epoch 38/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3588 - recall: 0.4735 - val_loss: 0.3553 - val_recall: 0.4224
Epoch 39/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3520 - recall: 0.4501 - val_loss: 0.3550 - val_recall: 0.4686
Epoch 40/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3583 - recall: 0.4695 - val_loss: 0.3497 - val_recall: 0.4752
Epoch 41/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3604 - recall: 0.4580 - val_loss: 0.3536 - val_recall: 0.4191
Epoch 42/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3577 - recall: 0.4484 - val_loss: 0.3521 - val_recall: 0.4719
Epoch 43/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3564 - recall: 0.4607 - val_loss: 0.3559 - val_recall: 0.3927
Epoch 44/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3593 - recall: 0.4401 - val_loss: 0.3553 - val_recall: 0.4092
Epoch 45/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3648 - recall: 0.4564 - val_loss: 0.3547 - val_recall: 0.4653
Epoch 46/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3621 - recall: 0.4704 - val_loss: 0.3545 - val_recall: 0.4224
Epoch 47/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3613 - recall: 0.4609 - val_loss: 0.3524 - val_recall: 0.4422
Epoch 48/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3521 - recall: 0.4906 - val_loss: 0.3535 - val_recall: 0.4587
Epoch 49/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3476 - recall: 0.5075 - val_loss: 0.3526 - val_recall: 0.4488
Epoch 50/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3563 - recall: 0.4970 - val_loss: 0.3569 - val_recall: 0.4719
```

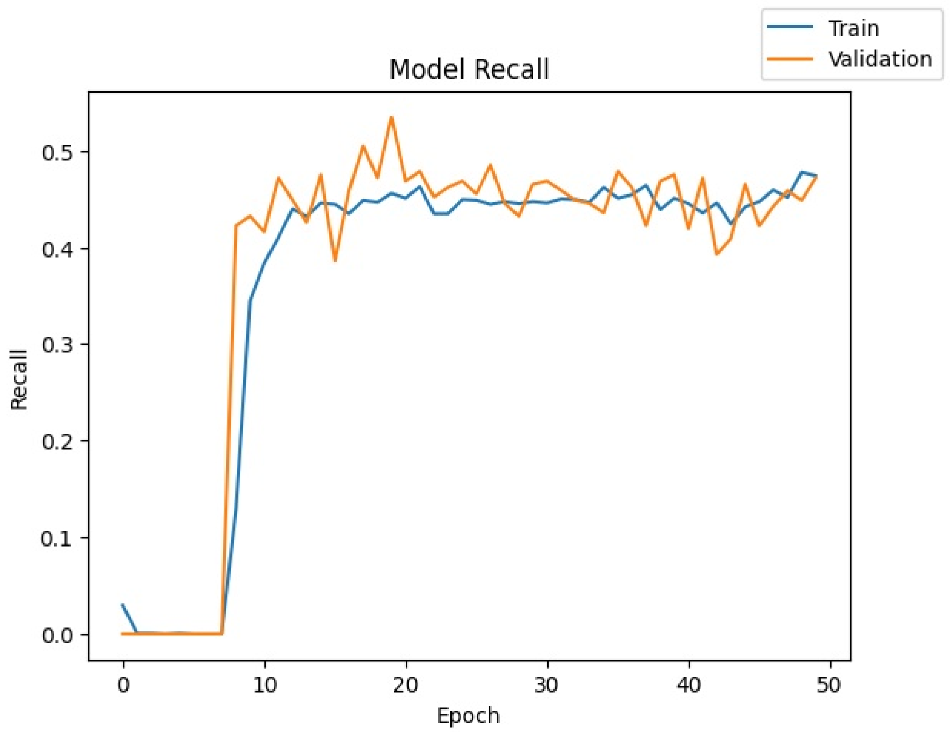
In [ ]:
```
print("Time taken in seconds: ",end-start)
```

```
Time taken in seconds:  35.98842406272888
```

In [ ]:
```
plot(history,'loss')
```

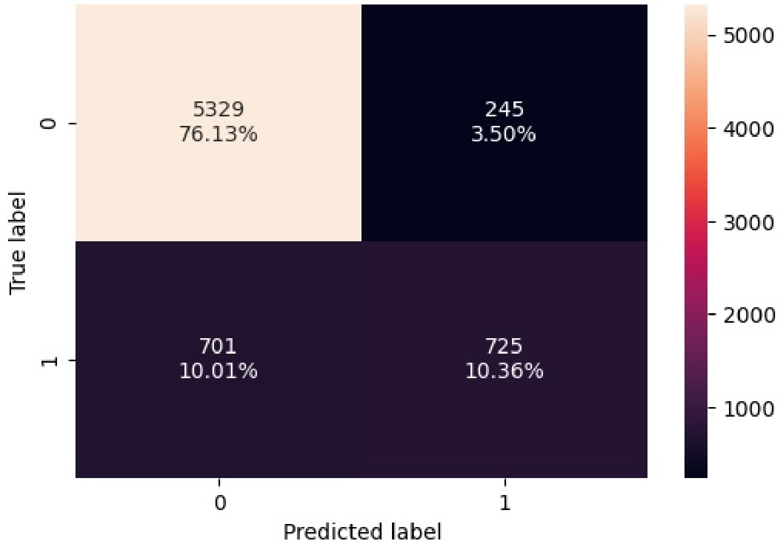



```
In [ ]: plot(history,'recall')
```




```
In [ ]: plot_confusion_matrix(model_2, X_train, y_train, 'train', model_name)
```

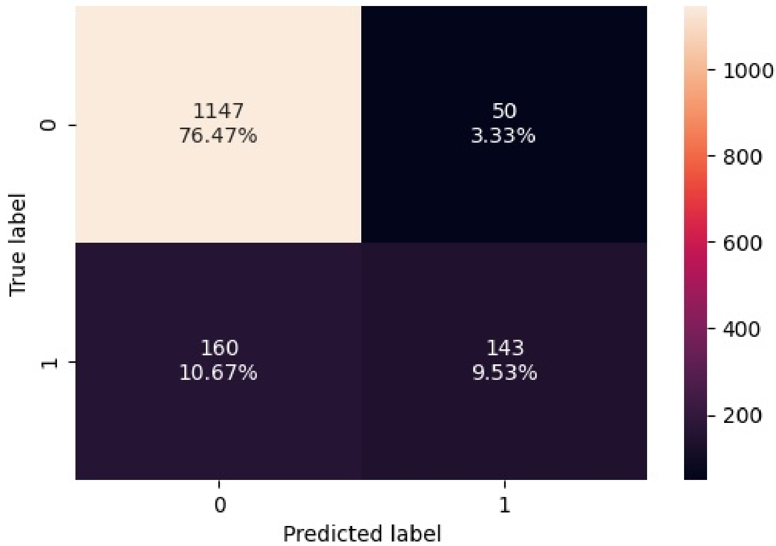
```
(7000, 11)
(7000, 1)
Confusion Matrix - NN with Adam With Dropout - train
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.88      0.96      0.92      5574
         1.0       0.75      0.51      0.61      1426

    accuracy                           0.86      7000
   macro avg       0.82      0.73      0.76      7000
weighted avg       0.86      0.86      0.85      7000
```

In [ ]:
```
plot_confusion_matrix(model_2, X_val, y_val, 'validation', model_name)
```

```
(1500, 11)
(1500, 1)
Confusion Matrix - NN with Adam With Dropout - validation
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



```
              precision    recall  f1-score   support

         0.0       0.88      0.96      0.92      1197
         1.0       0.74      0.47      0.58       303

    accuracy                           0.86      1500
   macro avg       0.81      0.72      0.75      1500
weighted avg       0.85      0.86      0.85      1500
```

In [ ]:
```
results.drop([2], inplace=True, errors='ignore')
results.loc[2] = [5,[64,32,16,8,4],["relu","relu","relu","relu","relu"],50,25,"Adam",[0.001, "-"],"xavier","-",l
results
```

Out[ ]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	learning rate, momentum	weight initializer	regularization	train loss	validation loss	train recall	vali
0	2	[64, 32]	[relu, tanh]	100	50	sgd	[0.001, -]	xavier	-	0.353841	0.359846	0.414446	0.4
1	2	[64, 32]	[relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.291288	0.361662	0.572931	0.4
2	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.357890	0.356914	0.474053	0.4

In [ ]:
```
eval_metric
```

Out[ ]:

	train-recall	validation-recall	train-f1-score	validation-f1-score	train-precision	validation-precision	train-roc-auc	validation-roc-auc	train-accurecy	validation-accurecy
NN with SGD	0.434081	0.432343	0.545134	0.544699	0.732544	0.735955	0.696768	0.696539	0.852429	0.854
NN with Adam	0.570126	0.471947	0.656704	0.573146	0.774286	0.729592	0.763804	0.713835	0.878571	0.858
NN with Adam With Dropout	0.508415	0.471947	0.605175	0.576613	0.747423	0.740933	0.732231	0.715088	0.864857	0.86

Observations

- Loss & Recall Plot:
  - Both training and validation loss decrease and stabilize around epoch 20-30.
  - The validation loss shows slight noise but aligns well with training loss, which confirms no significant overfitting.
  - The training recall improves steadily up to around epoch 20 and stabilizes afterward.
  - The validation recall fluctuates significantly but generally aligns with the training recall, suggesting room for improvement in stability.
- Evaluation Metrics
  - The model did not improve in detecting exited customers compared to previous iterations and still shows limitations in recall, which is critical for this project.
  - The training set metrics are aligned with the validation set, indicating no significant overfitting.

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

In [ ]:
```
# Generating Synthetic samples using Over Sampling Technique - SMOTE
sm = SMOTE(sampling_strategy=1, k_neighbors=5, random_state=1)
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)
print('After UpSampling, the shape of train_X: {}'.format(X_train_over.shape))
print('After UpSampling, the shape of train_y: {} \n'.format(X_train_over.shape))
```

```
After UpSampling, the shape of train_X: (11148, 11)
After UpSampling, the shape of train_y: (11148, 11)
```

In [ ]:
```
tf.keras.backend.clear_session() #Clearing the session.
#Initializing the neural network
model_name = 'NN with SGD With SMOTE OverSampled Data';
batch_size=32
epochs=50
model_3 = Sequential()
input_dimention = X_train.shape[1]
print("Input Dimention:", input_dimention)
# Adding hidden layers
model_3.add(Dense(64, activation='relu', input_dim = input_dimention, kernel_regularizer=tf.keras.regularizers.l
model_3.add(BatchNormalization())
model_3.add(Dense(32, activation='relu'))
```

```
model_3.add(BatchNormalization())
model_3.add(Dense(16, activation='relu'))
model_3.add(BatchNormalization())
model_3.add(Dense(8, activation='relu'))
model_3.add(BatchNormalization())
model_3.add(Dense(4, activation='relu'))
# Adding the output layer
model_3.add(Dense(1, activation = 'sigmoid'))
optimizer = keras.optimizers.SGD(learning_rate=0.001)   # defining SGD as the optimizer to be used
model_3.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["recall"])
model_3.summary()
```

```
Input Dimention: 11
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
batch_normalization (BatchNormalization)	(None, 64)	256
dense_1 (Dense)	(None, 32)	2,080
batch_normalization_1 (BatchNormalization)	(None, 32)	128
dense_2 (Dense)	(None, 16)	528
batch_normalization_2 (BatchNormalization)	(None, 16)	64
dense_3 (Dense)	(None, 8)	136
batch_normalization_3 (BatchNormalization)	(None, 8)	32
dense_4 (Dense)	(None, 4)	36
dense_5 (Dense)	(None, 1)	5

Total params: 4,033 (15.75 KB)

Trainable params: 3,793 (14.82 KB)

Non-trainable params: 240 (960.00 B)

In [ ]:

```
start = time.time()
# Training (fitting) the model and capturing training history
history = model_3.fit(X_train_over, y_train_over, validation_data=(X_val,y_val) , batch_size=batch_size, epochs
end=time.time()
```

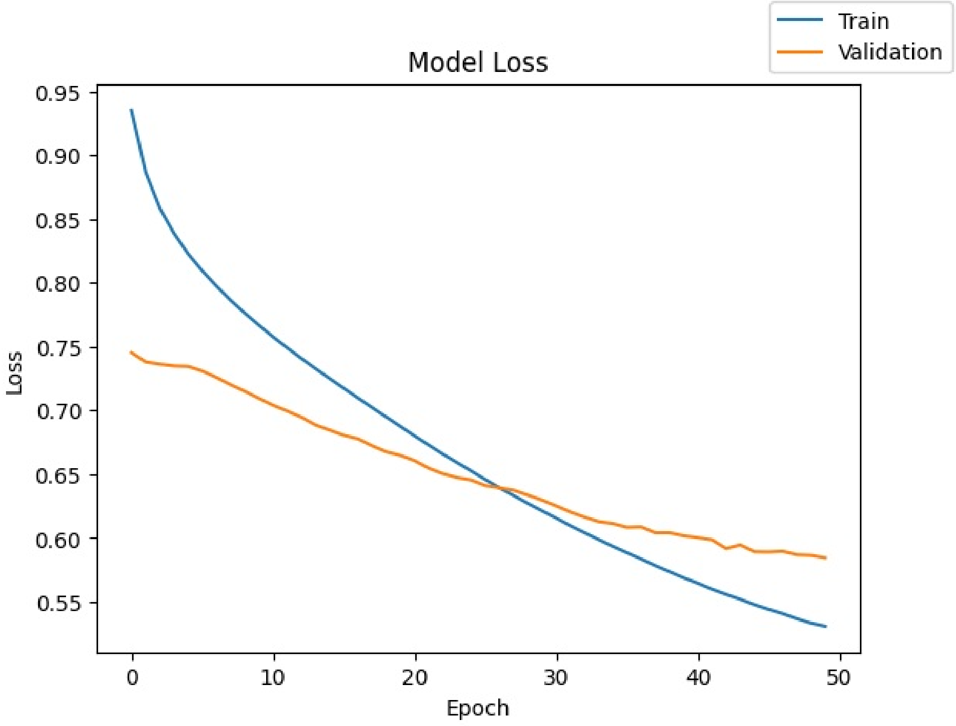
```
Epoch 1/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - loss: 0.9498 - recall: 0.0834 - val_loss: 0.7453 - val_recall: 0.1122
Epoch 2/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8938 - recall: 0.1699 - val_loss: 0.7381 - val_recall: 0.2475
Epoch 3/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8624 - recall: 0.2531 - val_loss: 0.7363 - val_recall: 0.3465
Epoch 4/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8415 - recall: 0.3416 - val_loss: 0.7349 - val_recall: 0.4026
Epoch 5/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8246 - recall: 0.4214 - val_loss: 0.7345 - val_recall: 0.4356
Epoch 6/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8115 - recall: 0.4926 - val_loss: 0.7309 - val_recall: 0.4884
Epoch 7/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7998 - recall: 0.5448 - val_loss: 0.7259 - val_recall: 0.5380
Epoch 8/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7887 - recall: 0.5981 - val_loss: 0.7202 - val_recall: 0.5908
Epoch 9/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7786 - recall: 0.6338 - val_loss: 0.7151 - val_recall: 0.6238
Epoch 10/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7693 - recall: 0.6541 - val_loss: 0.7095 - val_recall: 0.6304
Epoch 11/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7604 - recall: 0.6797 - val_loss: 0.7041 - val_recall: 0.6370
Epoch 12/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7518 - recall: 0.7002 - val_loss: 0.6996 - val_recall: 0.6568
Epoch 13/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7437 - recall: 0.7159 - val_loss: 0.6947 - val_recall: 0.6667
Epoch 14/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7359 - recall: 0.7279 - val_loss: 0.6886 - val_recall: 0.6634
Epoch 15/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7283 - recall: 0.7368 - val_loss: 0.6846 - val_recall: 0.6667
Epoch 16/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7207 - recall: 0.7480 - val_loss: 0.6806 - val_recall: 0.6799
```

```
Epoch 17/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7132 - recall: 0.7528 - val_loss: 0.6778 - val_recall: 0.6997
Epoch 18/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7059 - recall: 0.7572 - val_loss: 0.6723 - val_recall: 0.6964
Epoch 19/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6984 - recall: 0.7682 - val_loss: 0.6676 - val_recall: 0.7063
Epoch 20/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6910 - recall: 0.7721 - val_loss: 0.6648 - val_recall: 0.7096
Epoch 21/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6835 - recall: 0.7763 - val_loss: 0.6607 - val_recall: 0.7129
Epoch 22/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6760 - recall: 0.7785 - val_loss: 0.6547 - val_recall: 0.7129
Epoch 23/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6684 - recall: 0.7809 - val_loss: 0.6504 - val_recall: 0.7129
Epoch 24/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6615 - recall: 0.7826 - val_loss: 0.6474 - val_recall: 0.7129
Epoch 25/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6549 - recall: 0.7844 - val_loss: 0.6455 - val_recall: 0.7096
Epoch 26/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6480 - recall: 0.7854 - val_loss: 0.6412 - val_recall: 0.7096
Epoch 27/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6412 - recall: 0.7871 - val_loss: 0.6394 - val_recall: 0.7129
Epoch 28/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6347 - recall: 0.7911 - val_loss: 0.6375 - val_recall: 0.7261
Epoch 29/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6282 - recall: 0.7933 - val_loss: 0.6337 - val_recall: 0.7228
Epoch 30/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6220 - recall: 0.7953 - val_loss: 0.6297 - val_recall: 0.7195
Epoch 31/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6160 - recall: 0.7976 - val_loss: 0.6252 - val_recall: 0.7096
Epoch 32/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6095 - recall: 0.7984 - val_loss: 0.6205 - val_recall: 0.7129
Epoch 33/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6036 - recall: 0.8017 - val_loss: 0.6164 - val_recall: 0.7030
Epoch 34/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5978 - recall: 0.8039 - val_loss: 0.6129 - val_recall: 0.6964
Epoch 35/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5922 - recall: 0.8060 - val_loss: 0.6114 - val_recall: 0.6997
Epoch 36/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5870 - recall: 0.8059 - val_loss: 0.6084 - val_recall: 0.6964
Epoch 37/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5820 - recall: 0.8077 - val_loss: 0.6088 - val_recall: 0.6931
Epoch 38/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5765 - recall: 0.8089 - val_loss: 0.6042 - val_recall: 0.6898
Epoch 39/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5713 - recall: 0.8087 - val_loss: 0.6042 - val_recall: 0.6964
Epoch 40/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5662 - recall: 0.8096 - val_loss: 0.6019 - val_recall: 0.6964
Epoch 41/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5616 - recall: 0.8128 - val_loss: 0.6004 - val_recall: 0.6931
Epoch 42/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5565 - recall: 0.8152 - val_loss: 0.5987 - val_recall: 0.6997
Epoch 43/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5520 - recall: 0.8167 - val_loss: 0.5918 - val_recall: 0.6964
Epoch 44/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5483 - recall: 0.8142 - val_loss: 0.5947 - val_recall: 0.7063
Epoch 45/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5435 - recall: 0.8178 - val_loss: 0.5894 - val_recall: 0.6931
Epoch 46/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5398 - recall: 0.8155 - val_loss: 0.5892 - val_recall: 0.6931
Epoch 47/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5369 - recall: 0.8189 - val_loss: 0.5897 - val_recall: 0.6931
Epoch 48/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5323 - recall: 0.8199 - val_loss: 0.5870 - val_recall: 0.6931
Epoch 49/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5285 - recall: 0.8207 - val_loss: 0.5866 - val_recall: 0.6964
Epoch 50/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5269 - recall: 0.8207 - val_loss: 0.5845 - val_recall: 0.6997
```

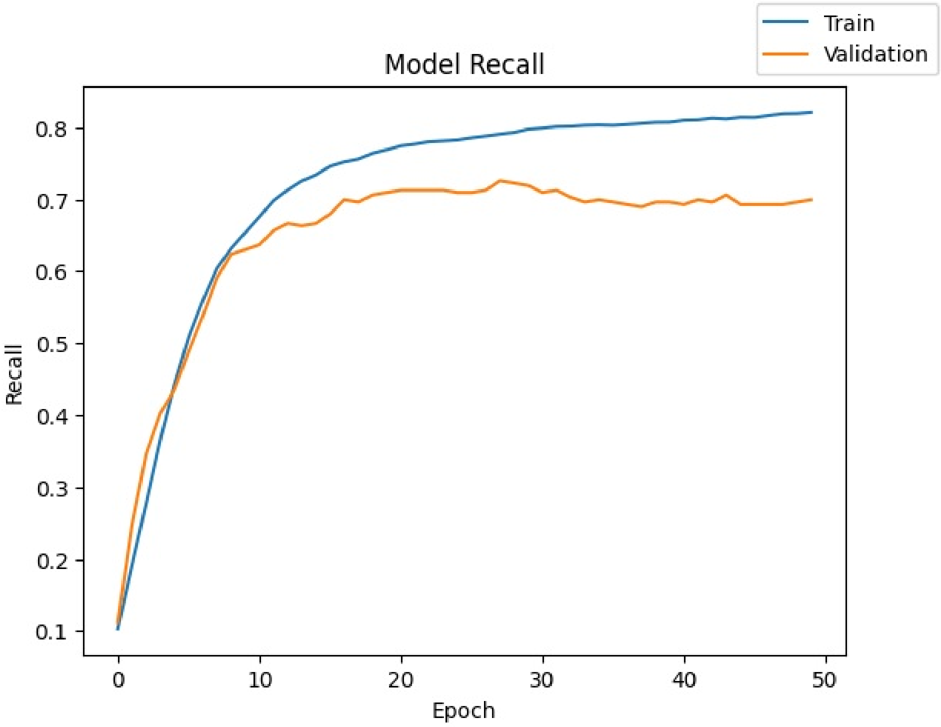
In [ ]:
```
print("Time taken in seconds: ",end-start)
```

```
Time taken in seconds:  39.71582245826721
```

In [ ]:
```
plot(history,'loss')
```

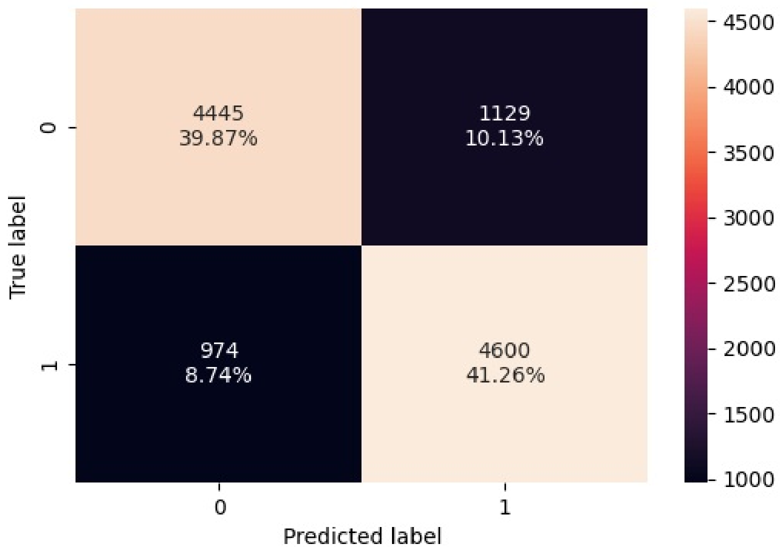



```
In [ ]: plot(history,'recall')
```




```
In [ ]: plot_confusion_matrix(model_3, X_train_over, y_train_over, 'train', model_name)
```

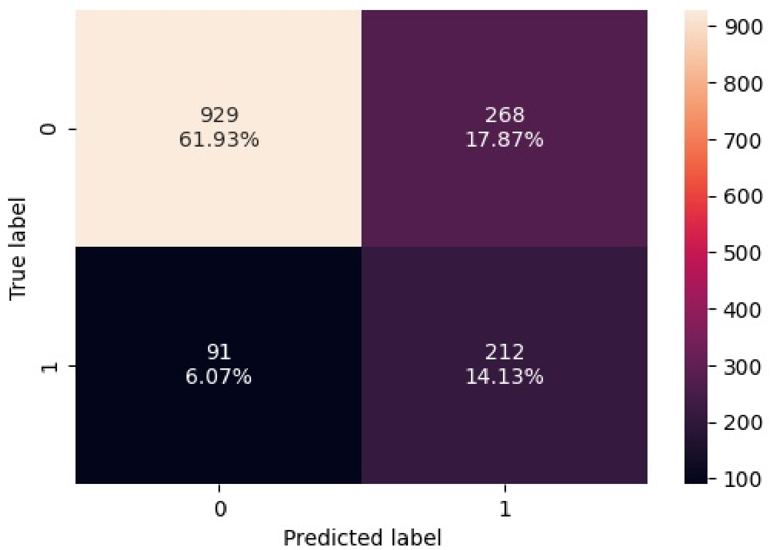
```
(11148, 11)
(11148, 1)
Confusion Matrix - NN with SGD With SMOTE OverSampled Data - train
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.82      0.80      0.81      5574
         1.0       0.80      0.83      0.81      5574

    accuracy                           0.81     11148
   macro avg       0.81      0.81      0.81     11148
weighted avg       0.81      0.81      0.81     11148
```

In [ ]:
```
plot_confusion_matrix(model_3, X_val, y_val, 'validation', model_name)
```

```
(1500, 11)
(1500, 1)
Confusion Matrix - NN with SGD With SMOTE OverSampled Data - validation
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```



```
              precision    recall  f1-score   support

         0.0       0.91      0.78      0.84      1197
         1.0       0.44      0.70      0.54       303

    accuracy                           0.76      1500
   macro avg       0.68      0.74      0.69      1500
weighted avg       0.82      0.76      0.78      1500
```

In [ ]:
```python
results.drop([3], inplace=True, errors='ignore')
results.loc[3] = [5,[64,32,16,8,4],["relu","relu","relu","relu","relu"],50,32,"sgd",[0.001, "-"],"xavier","-",h
results
```

Out[ ]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	learning rate, momentum	weight initializer	regularization	train loss	validation loss	train recall	vali
0	2	[64, 32]	[relu, tanh]	100	50	sgd	[0.001, -]	xavier	-	0.353841	0.359846	0.414446	0.4
1	2	[64, 32]	[relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.291288	0.361662	0.572931	0.4
2	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.357890	0.356914	0.474053	0.4
3	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	sgd	[0.001, -]	xavier	-	0.530878	0.584543	0.821313	0.6

In [ ]:
```python
eval_metric
```

Out[ ]:

	train-recall	validation-recall	train-f1-score	validation-f1-score	train-precision	validation-precision	train-roc-auc	validation-roc-auc	train-accurecy	validation-accurecy
NN with SGD	0.434081	0.432343	0.545134	0.544699	0.732544	0.735955	0.696768	0.696539	0.852429	0.854
NN with Adam	0.570126	0.471947	0.656704	0.573146	0.774286	0.729592	0.763804	0.713835	0.878571	0.858
NN with Adam With Dropout	0.508415	0.471947	0.605175	0.576613	0.747423	0.740933	0.732231	0.715088	0.864857	0.86
NN with SGD With SMOTE OverSampled Data	0.82526	0.69967	0.813943	0.541507	0.802932	0.441667	0.811356	0.737888	0.811356	0.760667

Observations

- Loss & Recall Plot:
  - Training Loss continues to decrease steadily, suggesting the model learns effectively from the training data.
  - Validation Loss decreases initially but slows down later, indicating the model might be hitting its performance ceiling due to the limited capability of the current configuration or issues from oversampling.
  - Training Recall steadily improves and stabilizes later, which is consistent with the balanced nature of the training data.
  - Validation Recall shows some fluctuations but stabilizes later towards the end. The gap between training and validation recall indicates mild overfitting, as the model performs better on the training set.
- Evaluation Metrics:
  - Recall for churn shows significant improvement over previous models but still leaves room for improvement in capturing the minority class.
  - Precision for non-churn, indicates a high performance for training set. But significantly low for validation set. This is most probebly because SMOTE oversampling shifts the balance of precision-recall trade-offs.
- Points to Note
  - The **Recall for Class 1** is significantly higher compared to previous models.
  - **Precision & Validation F1-Scores** are lower than expected, reflecting trade-offs made to increase recall for the minority class.
  - **Accuracy** is lower than previous models

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

In [ ]:
```python
tf.keras.backend.clear_session() #Clearing the session.
#Initializing the neural network
model_name = 'NN with Adam With SMOTE OverSampled Data';
batch_size=32
epochs=50
```

```python
model_4 = Sequential()
input_dimention = X_train.shape[1]
print("Input Dimention:", input_dimention)
# Adding the hidden layers & Batch Normalization
model_4.add(Dense(64, activation='relu', input_dim = input_dimention, kernel_regularizer=tf.keras.regularizers.
model_4.add(BatchNormalization())
model_4.add(Dense(32, activation='relu'))
model_4.add(BatchNormalization())
model_4.add(Dense(16, activation='relu'))
model_4.add(BatchNormalization())
model_4.add(Dense(8, activation='relu'))
model_4.add(BatchNormalization())
model_4.add(Dense(4, activation='relu'))
# Adding the output layer
model_4.add(Dense(1, activation = 'sigmoid'))
optimizer = keras.optimizers.Adam(learning_rate=0.001)    # defining Adam as the optimizer to be used
model_4.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["recall"])
model_4.summary()
```

Input Dimention: 11

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
batch_normalization (BatchNormalization)	(None, 64)	256
dense_1 (Dense)	(None, 32)	2,080
batch_normalization_1 (BatchNormalization)	(None, 32)	128
dense_2 (Dense)	(None, 16)	528
batch_normalization_2 (BatchNormalization)	(None, 16)	64
dense_3 (Dense)	(None, 8)	136
batch_normalization_3 (BatchNormalization)	(None, 8)	32
dense_4 (Dense)	(None, 4)	36
dense_5 (Dense)	(None, 1)	5

Total params: 4,033 (15.75 KB)

Trainable params: 3,793 (14.82 KB)

Non-trainable params: 240 (960.00 B)

In [ ]:

```python
start = time.time()
history = model_4.fit(X_train_over, y_train_over, validation_data=(X_val,y_val) , batch_size=batch_size, epochs
end=time.time()
```

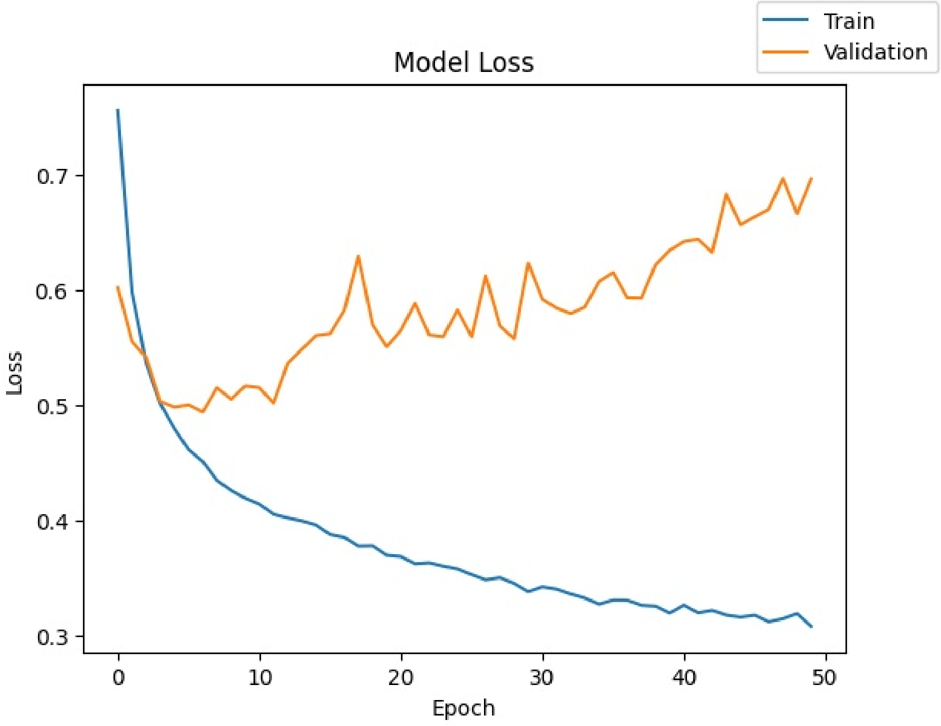
```
Epoch 1/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - loss: 0.8101 - recall: 0.6818 - val_loss: 0.6711 - val_recall: 0.7294
Epoch 2/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6382 - recall: 0.7689 - val_loss: 0.6060 - val_recall: 0.7063
Epoch 3/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5716 - recall: 0.7871 - val_loss: 0.6033 - val_recall: 0.7360
Epoch 4/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5302 - recall: 0.7926 - val_loss: 0.5569 - val_recall: 0.7261
Epoch 5/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4988 - recall: 0.8018 - val_loss: 0.5717 - val_recall: 0.7096
Epoch 6/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4805 - recall: 0.8149 - val_loss: 0.5160 - val_recall: 0.6766
Epoch 7/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4607 - recall: 0.8264 - val_loss: 0.5707 - val_recall: 0.7063
Epoch 8/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4446 - recall: 0.8298 - val_loss: 0.5308 - val_recall: 0.7030
Epoch 9/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4323 - recall: 0.8307 - val_loss: 0.5079 - val_recall: 0.6337
Epoch 10/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4216 - recall: 0.8361 - val_loss: 0.5513 - val_recall: 0.6865
Epoch 11/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4176 - recall: 0.8312 - val_loss: 0.5279 - val_recall: 0.6469
Epoch 12/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4218 - recall: 0.8299 - val_loss: 0.4861 - val_recall: 0.6469
Epoch 13/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4092 - recall: 0.8382 - val_loss: 0.5309 - val_recall: 0.6601
```

```
Epoch 14/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4099 - recall: 0.8360 - val_loss: 0.5250 - val_recall: 0.6535
Epoch 15/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3945 - recall: 0.8443 - val_loss: 0.5192 - val_recall: 0.6337
Epoch 16/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3934 - recall: 0.8406 - val_loss: 0.5195 - val_recall: 0.6502
Epoch 17/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3874 - recall: 0.8495 - val_loss: 0.5404 - val_recall: 0.6634
Epoch 18/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3897 - recall: 0.8464 - val_loss: 0.5245 - val_recall: 0.6535
Epoch 19/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3780 - recall: 0.8520 - val_loss: 0.5289 - val_recall: 0.6073
Epoch 20/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3831 - recall: 0.8492 - val_loss: 0.5361 - val_recall: 0.6568
Epoch 21/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3811 - recall: 0.8513 - val_loss: 0.5652 - val_recall: 0.6799
Epoch 22/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3671 - recall: 0.8555 - val_loss: 0.6421 - val_recall: 0.7162
Epoch 23/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3657 - recall: 0.8625 - val_loss: 0.5447 - val_recall: 0.6502
Epoch 24/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3654 - recall: 0.8589 - val_loss: 0.5545 - val_recall: 0.6601
Epoch 25/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3594 - recall: 0.8636 - val_loss: 0.5714 - val_recall: 0.6634
Epoch 26/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3532 - recall: 0.8685 - val_loss: 0.5721 - val_recall: 0.6403
Epoch 27/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3561 - recall: 0.8687 - val_loss: 0.6022 - val_recall: 0.6997
Epoch 28/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3517 - recall: 0.8703 - val_loss: 0.5792 - val_recall: 0.6502
Epoch 29/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3458 - recall: 0.8691 - val_loss: 0.5769 - val_recall: 0.6799
Epoch 30/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3471 - recall: 0.8752 - val_loss: 0.5935 - val_recall: 0.6634
Epoch 31/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3491 - recall: 0.8655 - val_loss: 0.5587 - val_recall: 0.6502
Epoch 32/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3334 - recall: 0.8744 - val_loss: 0.5820 - val_recall: 0.6535
Epoch 33/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3383 - recall: 0.8780 - val_loss: 0.5677 - val_recall: 0.6700
Epoch 34/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3325 - recall: 0.8781 - val_loss: 0.5922 - val_recall: 0.6139
Epoch 35/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3357 - recall: 0.8728 - val_loss: 0.5870 - val_recall: 0.6535
Epoch 36/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3353 - recall: 0.8798 - val_loss: 0.5640 - val_recall: 0.6436
Epoch 37/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3341 - recall: 0.8775 - val_loss: 0.5570 - val_recall: 0.6370
Epoch 38/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3266 - recall: 0.8803 - val_loss: 0.5643 - val_recall: 0.6205
Epoch 39/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3255 - recall: 0.8827 - val_loss: 0.5535 - val_recall: 0.6073
Epoch 40/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3328 - recall: 0.8732 - val_loss: 0.5718 - val_recall: 0.5776
Epoch 41/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3259 - recall: 0.8774 - val_loss: 0.5736 - val_recall: 0.5974
Epoch 42/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3246 - recall: 0.8798 - val_loss: 0.5769 - val_recall: 0.5974
Epoch 43/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3212 - recall: 0.8792 - val_loss: 0.5884 - val_recall: 0.6469
Epoch 44/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3246 - recall: 0.8822 - val_loss: 0.6025 - val_recall: 0.6337
Epoch 45/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3222 - recall: 0.8769 - val_loss: 0.5967 - val_recall: 0.6403
Epoch 46/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3227 - recall: 0.8905 - val_loss: 0.5816 - val_recall: 0.5578
Epoch 47/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3134 - recall: 0.8860 - val_loss: 0.5987 - val_recall: 0.5776
Epoch 48/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3220 - recall: 0.8803 - val_loss: 0.5632 - val_recall: 0.6073
Epoch 49/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3260 - recall: 0.8852 - val_loss: 0.5979 - val_recall: 0.6073
Epoch 50/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3089 - recall: 0.8889 - val_loss: 0.5863 - val_recall: 0.5512
```

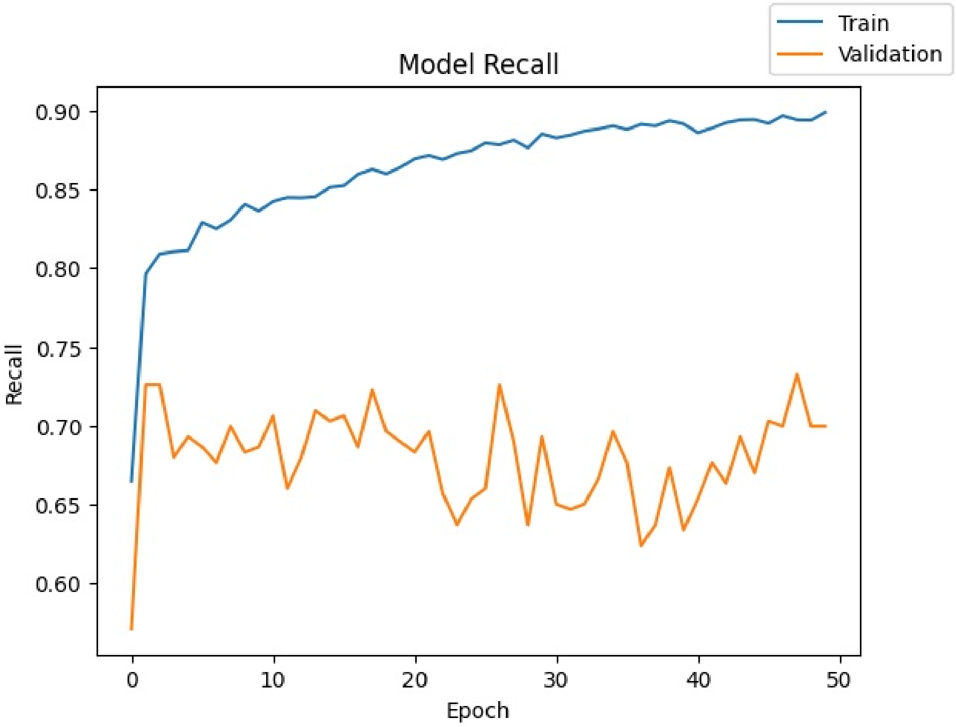
In [ ]:
```
print("Time taken in seconds: ",end-start)
```

```
Time taken in seconds:  46.0454261302948
```

In [ ]:
```
plot(history,'loss')
```

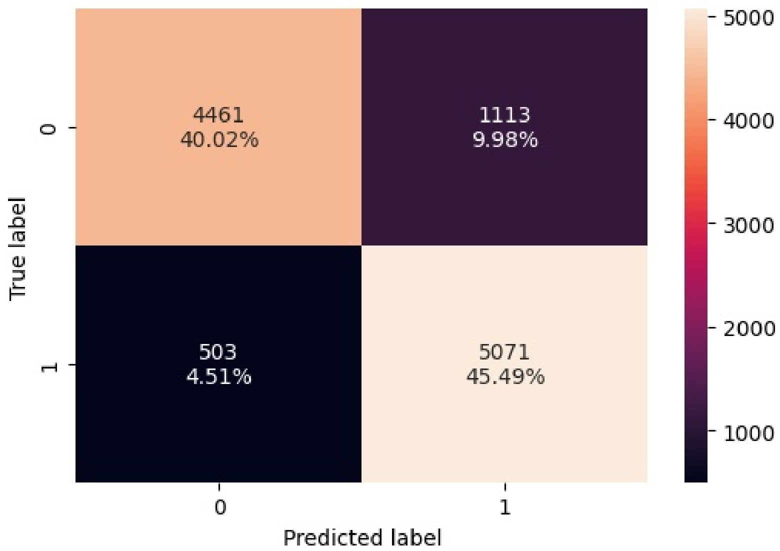



```
In [ ]: plot(history,'recall')
```




```
In [ ]: plot_confusion_matrix(model_4, X_train_over, y_train_over, 'train', model_name)
```

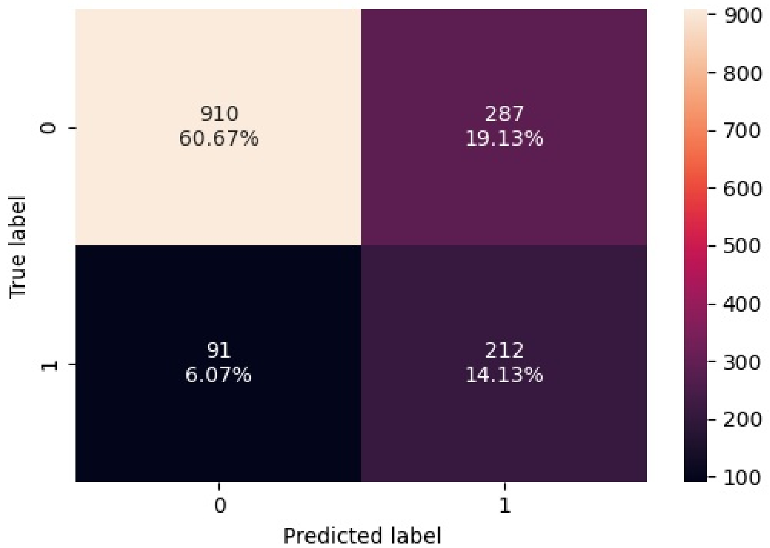
```
(11148, 11)
(11148, 1)
Confusion Matrix - NN with Adam With SMOTE OverSampled Data - train
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.90      0.80      0.85      5574
         1.0       0.82      0.91      0.86      5574

    accuracy                           0.86     11148
   macro avg       0.86      0.86      0.85     11148
weighted avg       0.86      0.86      0.85     11148
```

In [ ]:
```
plot_confusion_matrix(model_4, X_val, y_val, 'validation', model_name)
```

```
(1500, 11)
(1500, 1)
Confusion Matrix - NN with Adam With SMOTE OverSampled Data - validation
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```



```
              precision    recall  f1-score   support

         0.0       0.91      0.76      0.83      1197
         1.0       0.42      0.70      0.53       303

    accuracy                           0.75      1500

   macro avg       0.67      0.73      0.68      1500
weighted avg       0.81      0.75      0.77      1500
```

In [ ]:

```python
results.drop([4], inplace=True, errors='ignore')
results.loc[4] = [5,[64,32,16,8,4],["relu","relu","relu","relu","relu"],50,32,"Adam",[0.001, "-"],"xavier","-",
results
```

Out[ ]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	learning rate, momentum	weight initializer	regularization	train loss	validation loss	train recall	vali
0	2	[64, 32]	[relu, tanh]	100	50	sgd	[0.001, -]	xavier	-	0.353841	0.359846	0.414446	0.4
1	2	[64, 32]	[relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.291288	0.361662	0.572931	0.4
2	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.357890	0.356914	0.474053	0.4
3	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	sgd	[0.001, -]	xavier	-	0.530878	0.584543	0.821313	0.6
4	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	Adam	[0.001, -]	xavier	-	0.308397	0.696688	0.899175	0.6

In [ ]:

```python
eval_metric
```

Out[ ]:

	train-recall	validation-recall	train-f1-score	validation-f1-score	train-precision	validation-precision	train-roc-auc	validation-roc-auc	train-accuracy	validation-accuracy
NN with SGD	0.434081	0.432343	0.545134	0.544699	0.732544	0.735955	0.696768	0.696539	0.852429	0.854
NN with Adam	0.570126	0.471947	0.656704	0.573146	0.774286	0.729592	0.763804	0.713835	0.878571	0.858
NN with Adam With Dropout	0.508415	0.471947	0.605175	0.576613	0.747423	0.740933	0.732231	0.715088	0.864857	0.86
NN with SGD With SMOTE OverSampled Data	0.82526	0.69967	0.813943	0.541507	0.802932	0.441667	0.811356	0.737888	0.811356	0.760667
NN with Adam With SMOTE OverSampled Data	0.90976	0.69967	0.862562	0.528678	0.820019	0.42485	0.855041	0.729952	0.855041	0.748

Observations

- Loss & Recall Plot
  - The training loss is steadily decreasing, indicating that the model is learning well during the training phase. However, the validation loss increases after a certain point, suggesting potential overfitting.
  - The training recall continues to improve steadily, while validation recall fluctuates and remains lower. This could indicate overfitting, as the model generalizes less effectively on unseen data.
- Evaluation Metrics
  - The model performs well on the oversampled training dataset, achieving high precision, recall, and F1-score. This is expected as SMOTE balances the data and the model learns on it effectively.
  - The validation performance shows a decline in precision, and F1-score. Although recall has remained same leading to a moderate F1-score.
- Points to note
  - **Train vs Validation Gap**: The gap in metrics (e.g., recall, F1-score) between training and validation datasets highlights overfitting and warrants additional steps to improve generalization.

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

In [ ]:

```python
tf.keras.backend.clear_session() #Clearing the session.
```

```
#Initializing the neural network
model_name = 'NN with Adam With SMOTE OverSampled Data and DropOuts';
batch_size=32
epochs=50
model_5 = Sequential()
input_dimention = X_train.shape[1]
print("Input Dimention:", input_dimention)
# Adding hidden layers with droput ratios
model_5.add(Dense(64, activation='relu', input_dim = input_dimention, kernel_regularizer=tf.keras.regularizers.
model_5.add(Dropout(0.3))
model_5.add(Dense(32, activation='relu'))
model_5.add(Dropout(0.2))
model_5.add(Dense(16, activation='relu'))
model_5.add(Dense(8, activation='relu'))
model_5.add(Dropout(0.1))
model_5.add(Dense(4, activation='relu'))
# Adding the output layer
model_5.add(Dense(1, activation = 'sigmoid'))
optimizer = keras.optimizers.Adam(learning_rate=0.001)   # defining Adam as the optimizer to be used
model_5.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["recall"])
model_5.summary()
```

Input Dimention: 11

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 8)	136
dropout_2 (Dropout)	(None, 8)	0
dense_4 (Dense)	(None, 4)	36
dense_5 (Dense)	(None, 1)	5

Total params: 3,553 (13.88 KB)

Trainable params: 3,553 (13.88 KB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
start = time.time()
history = model_5.fit(X_train_over, y_train_over, validation_data=(X_val,y_val) , batch_size=batch_size, epochs
end=time.time()
```

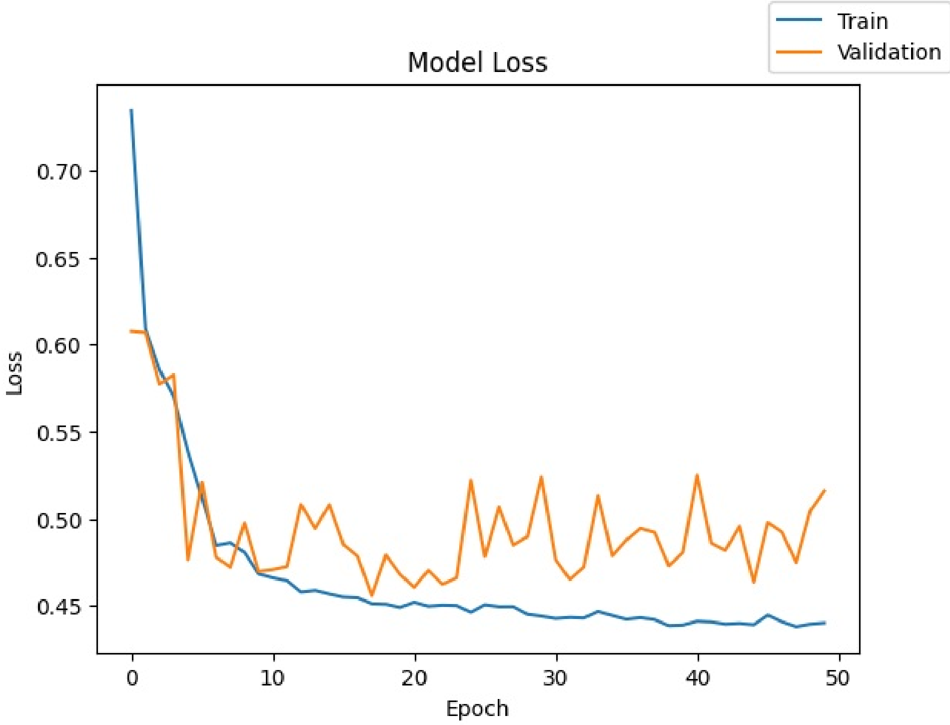
```
Epoch 1/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - loss: 0.8044 - recall: 0.3157 - val_loss: 0.6076 - val_recall: 0.755
8
Epoch 2/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6232 - recall: 0.7358 - val_loss: 0.6069 - val_recall: 0.7360
Epoch 3/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5887 - recall: 0.7357 - val_loss: 0.5772 - val_recall: 0.7129
Epoch 4/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5782 - recall: 0.7392 - val_loss: 0.5826 - val_recall: 0.7492
Epoch 5/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5505 - recall: 0.7459 - val_loss: 0.4763 - val_recall: 0.6436
Epoch 6/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5230 - recall: 0.7244 - val_loss: 0.5209 - val_recall: 0.6931
Epoch 7/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4860 - recall: 0.7589 - val_loss: 0.4778 - val_recall: 0.7096
Epoch 8/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4887 - recall: 0.7385 - val_loss: 0.4721 - val_recall: 0.6964
Epoch 9/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4825 - recall: 0.7401 - val_loss: 0.4976 - val_recall: 0.7261
Epoch 10/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4676 - recall: 0.7557 - val_loss: 0.4697 - val_recall: 0.7129
Epoch 11/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4591 - recall: 0.7732 - val_loss: 0.4707 - val_recall: 0.7030
Epoch 12/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4613 - recall: 0.7552 - val_loss: 0.4725 - val_recall: 0.6832
Epoch 13/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4566 - recall: 0.7611 - val_loss: 0.5081 - val_recall: 0.7393
Epoch 14/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4593 - recall: 0.7660 - val_loss: 0.4943 - val_recall: 0.7327
```

```
Epoch 15/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4556 - recall: 0.7764 - val_loss: 0.5080 - val_recall: 0.7492
Epoch 16/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4567 - recall: 0.7650 - val_loss: 0.4851 - val_recall: 0.7195
Epoch 17/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4537 - recall: 0.7728 - val_loss: 0.4784 - val_recall: 0.7228
Epoch 18/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4530 - recall: 0.7655 - val_loss: 0.4557 - val_recall: 0.6865
Epoch 19/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4494 - recall: 0.7593 - val_loss: 0.4791 - val_recall: 0.7360
Epoch 20/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4521 - recall: 0.7702 - val_loss: 0.4680 - val_recall: 0.6931
Epoch 21/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4507 - recall: 0.7710 - val_loss: 0.4605 - val_recall: 0.7030
Epoch 22/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4515 - recall: 0.7759 - val_loss: 0.4703 - val_recall: 0.7195
Epoch 23/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4490 - recall: 0.7743 - val_loss: 0.4623 - val_recall: 0.7030
Epoch 24/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4487 - recall: 0.7816 - val_loss: 0.4660 - val_recall: 0.7294
Epoch 25/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4468 - recall: 0.7840 - val_loss: 0.5222 - val_recall: 0.7558
Epoch 26/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4519 - recall: 0.7816 - val_loss: 0.4781 - val_recall: 0.7195
Epoch 27/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4492 - recall: 0.7837 - val_loss: 0.5068 - val_recall: 0.7624
Epoch 28/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4457 - recall: 0.7884 - val_loss: 0.4848 - val_recall: 0.7162
Epoch 29/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4458 - recall: 0.7787 - val_loss: 0.4896 - val_recall: 0.7360
Epoch 30/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4446 - recall: 0.7991 - val_loss: 0.5239 - val_recall: 0.7657
Epoch 31/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4414 - recall: 0.7829 - val_loss: 0.4764 - val_recall: 0.7096
Epoch 32/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4404 - recall: 0.7859 - val_loss: 0.4651 - val_recall: 0.7030
Epoch 33/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4399 - recall: 0.7897 - val_loss: 0.4722 - val_recall: 0.7162
Epoch 34/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4438 - recall: 0.7877 - val_loss: 0.5132 - val_recall: 0.7921
Epoch 35/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4408 - recall: 0.7985 - val_loss: 0.4786 - val_recall: 0.7327
Epoch 36/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4404 - recall: 0.7897 - val_loss: 0.4877 - val_recall: 0.7492
Epoch 37/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4399 - recall: 0.7980 - val_loss: 0.4944 - val_recall: 0.7558
Epoch 38/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4460 - recall: 0.8027 - val_loss: 0.4921 - val_recall: 0.7624
Epoch 39/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4371 - recall: 0.7956 - val_loss: 0.4729 - val_recall: 0.7261
Epoch 40/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4379 - recall: 0.7965 - val_loss: 0.4806 - val_recall: 0.7426
Epoch 41/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4458 - recall: 0.7811 - val_loss: 0.5249 - val_recall: 0.7987
Epoch 42/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4473 - recall: 0.7908 - val_loss: 0.4859 - val_recall: 0.7393
Epoch 43/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4411 - recall: 0.7912 - val_loss: 0.4818 - val_recall: 0.7558
Epoch 44/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4369 - recall: 0.7935 - val_loss: 0.4957 - val_recall: 0.7459
Epoch 45/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4384 - recall: 0.8031 - val_loss: 0.4634 - val_recall: 0.7360
Epoch 46/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4421 - recall: 0.7952 - val_loss: 0.4978 - val_recall: 0.7525
Epoch 47/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4417 - recall: 0.7960 - val_loss: 0.4923 - val_recall: 0.7657
Epoch 48/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4403 - recall: 0.7899 - val_loss: 0.4747 - val_recall: 0.7426
Epoch 49/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4410 - recall: 0.7948 - val_loss: 0.5045 - val_recall: 0.7690
Epoch 50/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4402 - recall: 0.7957 - val_loss: 0.5159 - val_recall: 0.7855
```

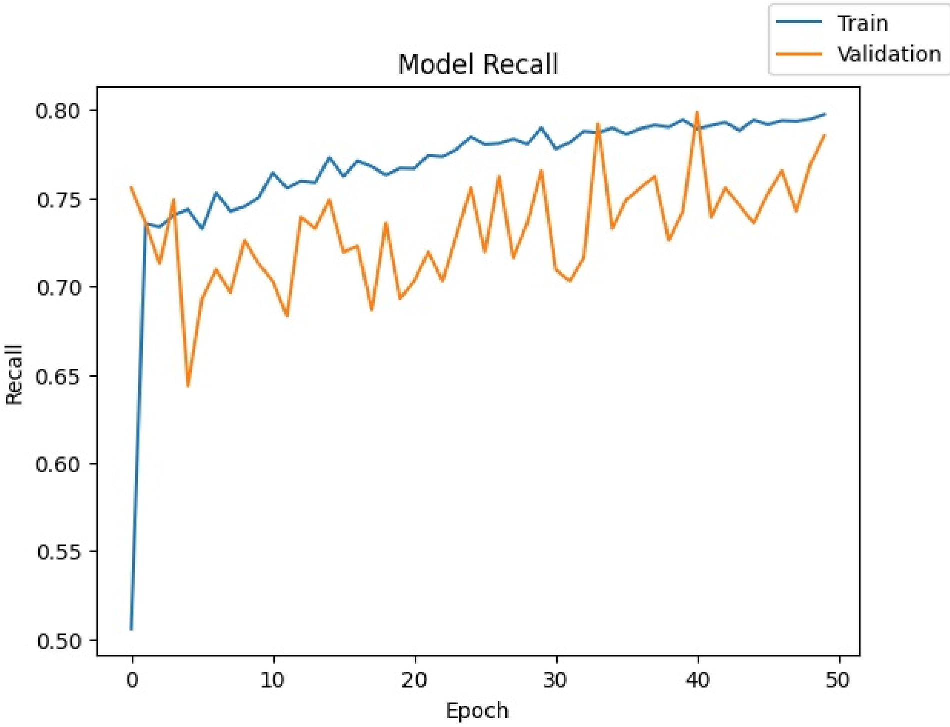
In [ ]:
```
print("Time taken in seconds: ",end-start)
```

```
Time taken in seconds:  43.570762634277344
```

In [ ]:
```
plot(history,'loss')
```

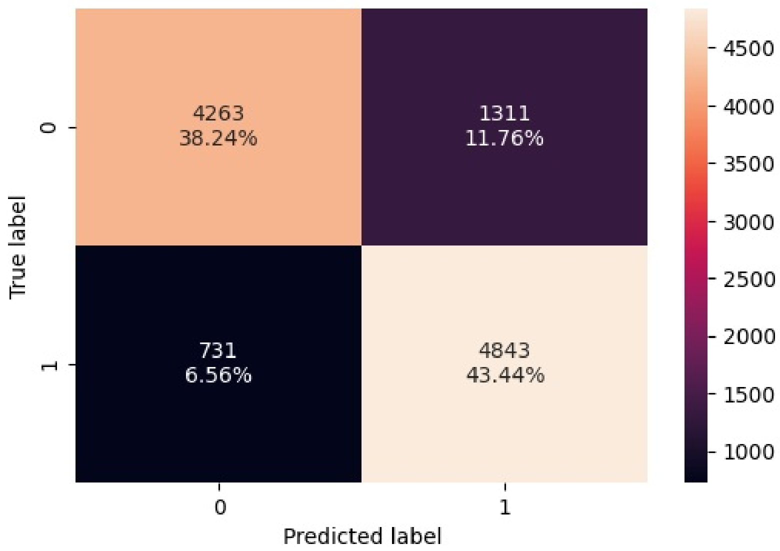


In [ ]:
```
plot(history,'recall')
```



In [ ]:
```
plot_confusion_matrix(model_5, X_train_over, y_train_over, 'train', model_name)
```

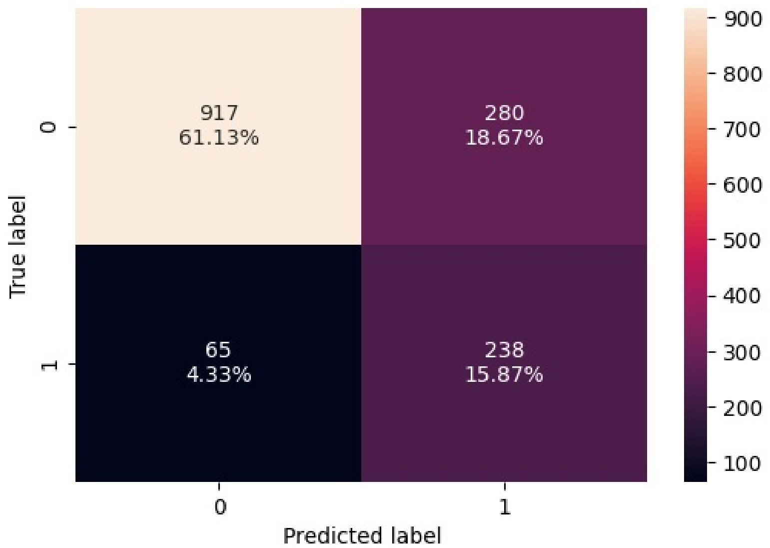
```
(11148, 11)
(11148, 1)
Confusion Matrix - NN with Adam With SMOTE OverSampled Data and DropOuts - train
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



	precision	recall	f1-score	support
0.0	0.85	0.76	0.81	5574
1.0	0.79	0.87	0.83	5574
accuracy			0.82	11148
macro avg	0.82	0.82	0.82	11148
weighted avg	0.82	0.82	0.82	11148

In [ ]:
```
plot_confusion_matrix(model_5, X_val, y_val, 'validation', model_name)
```

```
(1500, 11)
(1500, 1)
Confusion Matrix - NN with Adam With SMOTE OverSampled Data and DropOuts - validation
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```



```
              precision    recall  f1-score   support

         0.0       0.93      0.77      0.84      1197
         1.0       0.46      0.79      0.58       303

    accuracy                           0.77      1500

   macro avg       0.70      0.78      0.71      1500
weighted avg       0.84      0.77      0.79      1500
```

In [ ]:
```
results.drop([5], inplace=True, errors='ignore')
results.loc[5] = [5,[64,32,16,8,4],["relu","relu","relu","relu","relu"],50,32,"Adam",[0.001, "-"],"xavier","-",[
results
```

Out[ ]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	learning rate, momentum	weight initializer	regularization	train loss	validation loss	train recall	vali
0	2	[64, 32]	[relu, tanh]	100	50	sgd	[0.001, -]	xavier	-	0.353841	0.359846	0.414446	0.4
1	2	[64, 32]	[relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.291288	0.361662	0.572931	0.4
2	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.357890	0.356914	0.474053	0.4
3	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	sgd	[0.001, -]	xavier	-	0.530878	0.584543	0.821313	0.6
4	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	Adam	[0.001, -]	xavier	-	0.308397	0.696688	0.899175	0.6
5	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	Adam	[0.001, -]	xavier	-	0.439992	0.515863	0.797452	0.7

In [ ]:
```
eval_metric
```

Out[ ]:

	train-recall	validation-recall	train-f1-score	validation-f1-score	train-precision	validation-precision	train-roc-auc	validation-roc-auc	train-accurecy	validation-accurecy
NN with SGD	0.434081	0.432343	0.545134	0.544699	0.732544	0.735955	0.696768	0.696539	0.852429	0.854
NN with Adam	0.570126	0.471947	0.656704	0.573146	0.774286	0.729592	0.763804	0.713835	0.878571	0.858
NN with Adam With Dropout	0.508415	0.471947	0.605175	0.576613	0.747423	0.740933	0.732231	0.715088	0.864857	0.86
NN with SGD With SMOTE OverSampled Data	0.82526	0.69967	0.813943	0.541507	0.802932	0.441667	0.811356	0.737888	0.811356	0.760667
NN with Adam With SMOTE OverSampled Data	0.90976	0.69967	0.862562	0.528678	0.820019	0.42485	0.855041	0.729952	0.855041	0.748
NN with Adam With SMOTE OverSampled Data and DropOuts	0.868855	0.785479	0.825887	0.579781	0.786968	0.459459	0.816828	0.77578	0.816828	0.77

Observations

- Loss & Recall Plot
  - The training loss decreases steadily, which is expected, while the validation loss decreases but with fluctuations.
  - This plot indicates that the dropout layers successfully mitigate overfitting while maintaining generalization.
  - The recall for the training set stabilizes at the end, while the validation recall fluctuates but trends upward, indicating consistent learning and generalization on the validation set.
- Evaluation Metrics
  - The **training recall** is high at **0.868** and validation recall improved to **0.785**, which indicates a relatively well-learned model with reduced overfitting compared to prior models.
  - Validation F1-score and precision have slightly increased compared to previous Model, suggesting better handling of false positives and negatives.
- Points to Note

- **Validation Recall (0.7854)**: This is the best performance seen so far across all models. This improvement validates the impact of combining SMOTE, Dropout, Batch Normalization, and Adam optimizer.
- However for unseen dataset, the precision and F1-Score remains low, suggesting that the model will produce false positives.
- For the **training set**, both precision and recall are balanced demonstrating that the model is well-trained on the SMOTE-oversampled data.

Neural Network - With Leakey ReLU activation function

In [ ]:
```
from keras.layers import LeakyReLU
tf.keras.backend.clear_session() #Clearing the session.
#Initializing the neural network
model_name = 'NN using LeakyReLU with Adam With SMOTE OverSampled Data and DropOuts';
batch_size=32
epochs=50
model_6 = Sequential()
input_dimention = X_train.shape[1]
print("Input Dimention:", input_dimention)
# Adding hidden layers
model_6.add(Dense(64, input_dim = input_dimention, kernel_regularizer=tf.keras.regularizers.l2(0.01)))
# Using LeakeuReLU as activation function
model_6.add(LeakyReLU(alpha=0.05))
# Adding dropout ratio
model_6.add(Dropout(0.3))
model_6.add(Dense(32))
# Using LeakeuReLU as activation function
model_6.add(LeakyReLU(alpha=0.05))
# Adding dropout ratio
model_6.add(Dropout(0.2))
model_6.add(Dense(16))
# Using LeakeuReLU as activation function
model_6.add(LeakyReLU(alpha=0.05))
model_6.add(Dense(8))
# Using LeakeuReLU as activation function
model_6.add(LeakyReLU(alpha=0.05))
# Adding dropout ratio
model_6.add(Dropout(0.1))
model_6.add(Dense(4))
# Adding the output layer
model_6.add(Dense(1, activation = 'sigmoid'))
optimizer = keras.optimizers.Adam(learning_rate=0.001)   # defining Adam as the optimizer to be used
model_6.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["recall"])
model_6.summary()
```

```
Input Dimention: 11
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
leaky_re_lu (LeakyReLU)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
leaky_re_lu_1 (LeakyReLU)	(None, 32)	0
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
leaky_re_lu_2 (LeakyReLU)	(None, 16)	0
dense_3 (Dense)	(None, 8)	136
leaky_re_lu_3 (LeakyReLU)	(None, 8)	0
leaky_re_lu_4 (LeakyReLU)	(None, 8)	0
dropout_2 (Dropout)	(None, 8)	0
dense_4 (Dense)	(None, 4)	36
dense_5 (Dense)	(None, 1)	5

Total params: 3,553 (13.88 KB)
Trainable params: 3,553 (13.88 KB)
Non-trainable params: 0 (0.00 B)

```
In [ ]: start = time.time()
        # Training the model
        history = model_6.fit(X_train_over, y_train_over, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=
        end=time.time()
```

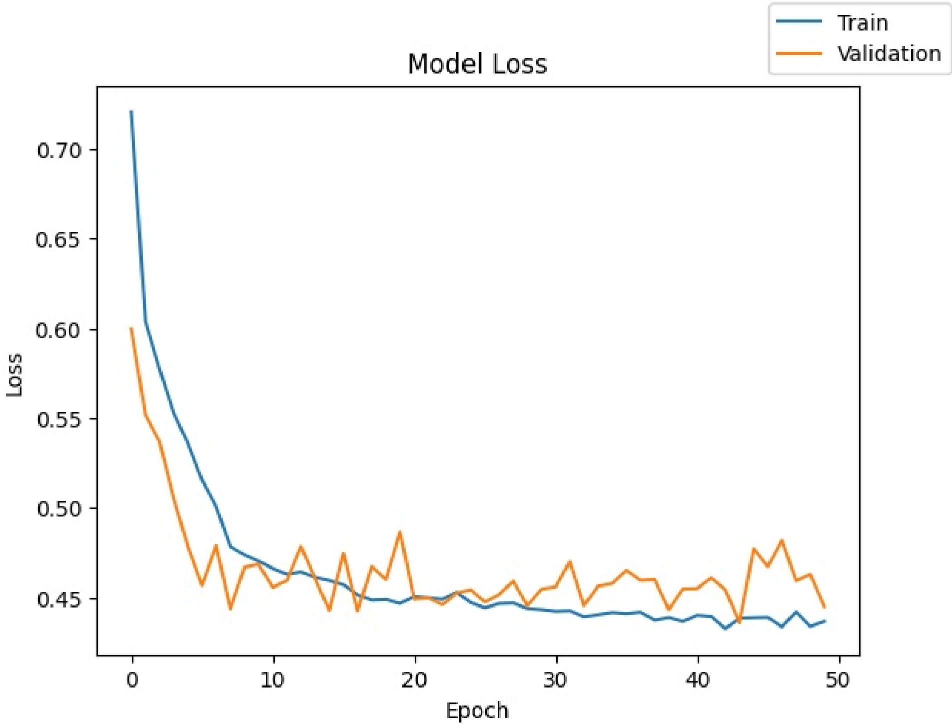
```
Epoch 1/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 9s 13ms/step - loss: 0.7921 - recall: 0.6205 - val_loss: 0.5997 - val_recall: 0.719
5
Epoch 2/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6202 - recall: 0.7152 - val_loss: 0.5516 - val_recall: 0.7162
Epoch 3/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5845 - recall: 0.7398 - val_loss: 0.5369 - val_recall: 0.7030
Epoch 4/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5593 - recall: 0.7456 - val_loss: 0.5052 - val_recall: 0.6469
Epoch 5/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5422 - recall: 0.7330 - val_loss: 0.4785 - val_recall: 0.6139
Epoch 6/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5248 - recall: 0.7329 - val_loss: 0.4569 - val_recall: 0.6304
Epoch 7/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5076 - recall: 0.7410 - val_loss: 0.4790 - val_recall: 0.7162
Epoch 8/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4789 - recall: 0.7700 - val_loss: 0.4437 - val_recall: 0.6931
Epoch 9/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4729 - recall: 0.7794 - val_loss: 0.4669 - val_recall: 0.7195
Epoch 10/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4666 - recall: 0.7900 - val_loss: 0.4686 - val_recall: 0.7393
Epoch 11/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4645 - recall: 0.7908 - val_loss: 0.4556 - val_recall: 0.7129
Epoch 12/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4630 - recall: 0.7783 - val_loss: 0.4595 - val_recall: 0.7030
Epoch 13/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4650 - recall: 0.7859 - val_loss: 0.4785 - val_recall: 0.7393
Epoch 14/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4576 - recall: 0.7946 - val_loss: 0.4605 - val_recall: 0.7294
Epoch 15/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4608 - recall: 0.7856 - val_loss: 0.4428 - val_recall: 0.7063
Epoch 16/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4522 - recall: 0.8025 - val_loss: 0.4746 - val_recall: 0.7360
Epoch 17/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4533 - recall: 0.7983 - val_loss: 0.4425 - val_recall: 0.7096
Epoch 18/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4456 - recall: 0.7876 - val_loss: 0.4673 - val_recall: 0.7360
Epoch 19/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4446 - recall: 0.7975 - val_loss: 0.4601 - val_recall: 0.7162
Epoch 20/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4458 - recall: 0.7961 - val_loss: 0.4865 - val_recall: 0.7492
Epoch 21/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4477 - recall: 0.8049 - val_loss: 0.4492 - val_recall: 0.6931
Epoch 22/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4492 - recall: 0.7847 - val_loss: 0.4498 - val_recall: 0.6898
Epoch 23/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4475 - recall: 0.7917 - val_loss: 0.4462 - val_recall: 0.7162
Epoch 24/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4488 - recall: 0.8078 - val_loss: 0.4524 - val_recall: 0.6799
Epoch 25/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4425 - recall: 0.7973 - val_loss: 0.4543 - val_recall: 0.6997
Epoch 26/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4416 - recall: 0.8039 - val_loss: 0.4476 - val_recall: 0.7063
Epoch 27/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4395 - recall: 0.8066 - val_loss: 0.4516 - val_recall: 0.6931
Epoch 28/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4451 - recall: 0.7965 - val_loss: 0.4592 - val_recall: 0.7393
Epoch 29/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4412 - recall: 0.8016 - val_loss: 0.4456 - val_recall: 0.7195
Epoch 30/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4391 - recall: 0.8101 - val_loss: 0.4546 - val_recall: 0.7096
Epoch 31/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4418 - recall: 0.8003 - val_loss: 0.4559 - val_recall: 0.7096
Epoch 32/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4420 - recall: 0.8058 - val_loss: 0.4700 - val_recall: 0.7129
Epoch 33/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4379 - recall: 0.8024 - val_loss: 0.4455 - val_recall: 0.6700
Epoch 34/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4395 - recall: 0.7936 - val_loss: 0.4564 - val_recall: 0.7096
Epoch 35/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4449 - recall: 0.7988 - val_loss: 0.4579 - val_recall: 0.6700
Epoch 36/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4391 - recall: 0.8005 - val_loss: 0.4651 - val_recall: 0.7030
Epoch 37/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4425 - recall: 0.7972 - val_loss: 0.4596 - val_recall: 0.7129
Epoch 38/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4345 - recall: 0.7972 - val_loss: 0.4600 - val_recall: 0.7129
Epoch 39/50
```

```
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4387 - recall: 0.7953 - val_loss: 0.4433 - val_recall: 0.6931
Epoch 40/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4378 - recall: 0.7958 - val_loss: 0.4548 - val_recall: 0.7129
Epoch 41/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4378 - recall: 0.8075 - val_loss: 0.4550 - val_recall: 0.6964
Epoch 42/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4391 - recall: 0.8053 - val_loss: 0.4609 - val_recall: 0.7327
Epoch 43/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4345 - recall: 0.7979 - val_loss: 0.4543 - val_recall: 0.6997
Epoch 44/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4356 - recall: 0.8027 - val_loss: 0.4359 - val_recall: 0.6502
Epoch 45/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4393 - recall: 0.7905 - val_loss: 0.4773 - val_recall: 0.7294
Epoch 46/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4387 - recall: 0.8023 - val_loss: 0.4673 - val_recall: 0.7195
Epoch 47/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4340 - recall: 0.8029 - val_loss: 0.4819 - val_recall: 0.7327
Epoch 48/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4404 - recall: 0.7945 - val_loss: 0.4593 - val_recall: 0.7063
Epoch 49/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4322 - recall: 0.7904 - val_loss: 0.4629 - val_recall: 0.7030
Epoch 50/50
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4395 - recall: 0.8055 - val_loss: 0.4449 - val_recall: 0.6931
```

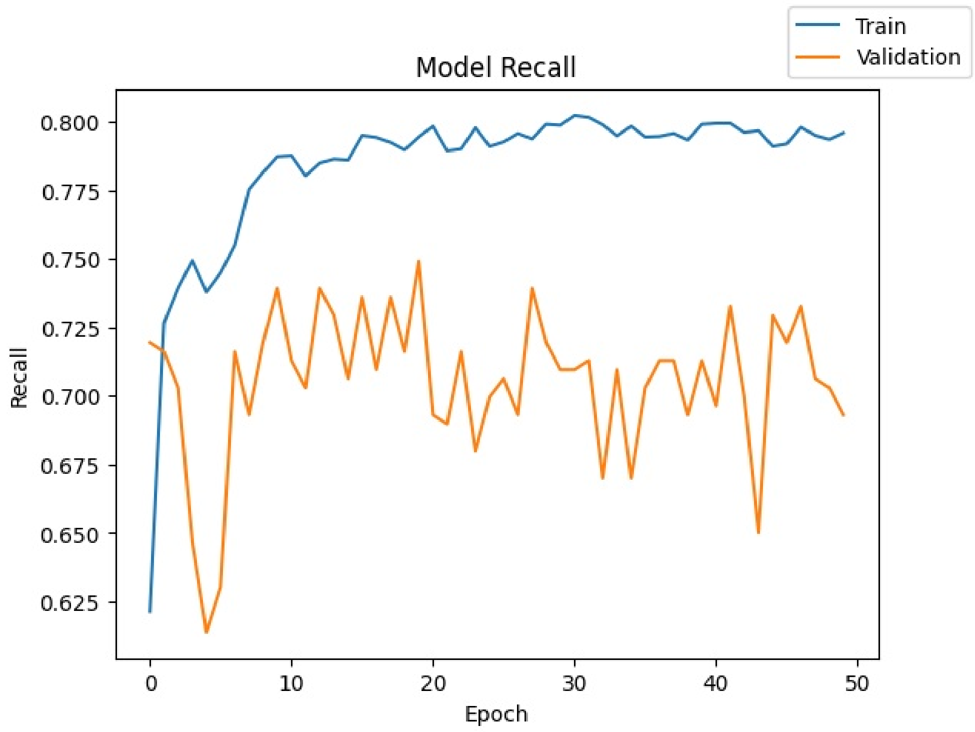
In [ ]:
```
print("Time taken in seconds: ",end-start)
```

```
Time taken in seconds:  47.214338064193726
```

In [ ]:
```
plot(history,'loss')
```

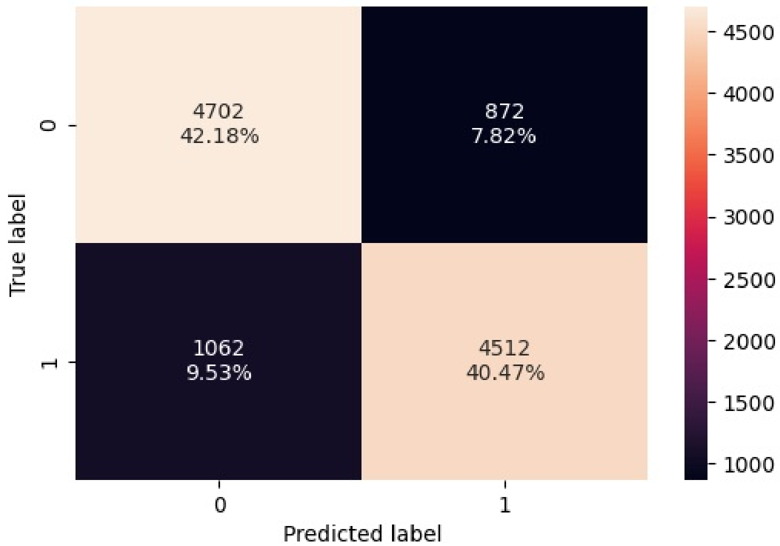


In [ ]:
```
plot(history,'recall')
```



In [ ]:
```
plot_confusion_matrix(model_6, X_train_over, y_train_over, 'train', model_name)
```

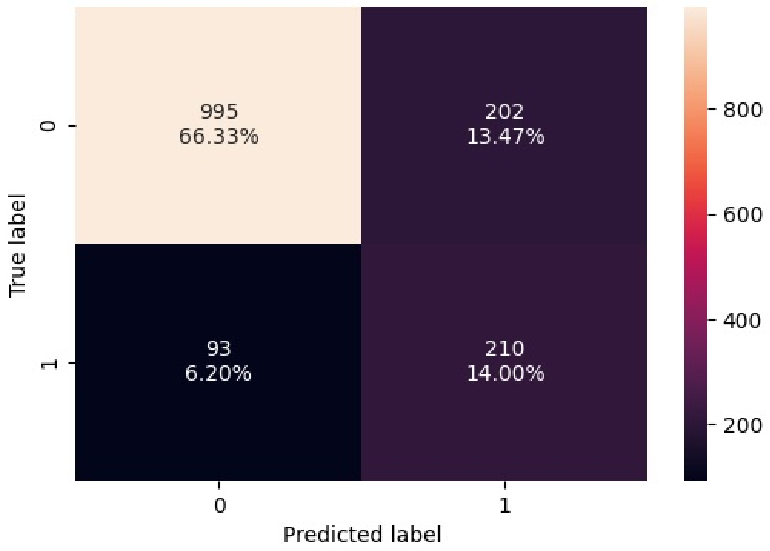
```
(11148, 11)
(11148, 1)
Confusion Matrix - NN using LeakyReLU with Adam With SMOTE OverSampled Data and DropOuts - train
349/349 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.82      0.84      0.83      5574
         1.0       0.84      0.81      0.82      5574

    accuracy                           0.83     11148
   macro avg       0.83      0.83      0.83     11148
weighted avg       0.83      0.83      0.83     11148
```

In [ ]:
```
plot_confusion_matrix(model_6, X_val, y_val, 'validation', model_name)
```

```
(1500, 11)
(1500, 1)
Confusion Matrix - NN using LeakyReLU with Adam With SMOTE OverSampled Data and DropOuts - validation
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.91      0.83      0.87      1197
         1.0       0.51      0.69      0.59       303

    accuracy                           0.80      1500
   macro avg       0.71      0.76      0.73      1500
weighted avg       0.83      0.80      0.81      1500
```

In [ ]:
```
results.drop([6], inplace=True, errors='ignore')
results.loc[6] = [5,[64,32,16,8,4],["LeakyReLU","LeakyReLU","LeakyReLU","LeakyReLU","LeakyReLU"],50,32,"Adam",[
results
```

Out[ ]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	learning rate, momentum	weight initializer	regularization	train loss	validation loss	train recall	v
0	2	[64, 32]	[relu, tanh]	100	50	sgd	[0.001, -]	xavier	-	0.353841	0.359846	0.414446	
1	2	[64, 32]	[relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.291288	0.361662	0.572931	
2	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	25	Adam	[0.001, -]	xavier	-	0.357890	0.356914	0.474053	
3	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	sgd	[0.001, -]	xavier	-	0.530878	0.584543	0.821313	
4	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	Adam	[0.001, -]	xavier	-	0.308397	0.696688	0.899175	
5	5	[64, 32, 16, 8, 4]	[relu, relu, relu, relu, relu]	50	32	Adam	[0.001, -]	xavier	-	0.439992	0.515863	0.797452	
6	5	[64, 32, 16, 8, 4]	[LeakyReLU, LeakyReLU, LeakyReLU, LeakyReLU, L...	50	32	Adam	[0.001, -]	xavier	-	0.436706	0.444946	0.795838	

In [ ]:
```
eval_metric
```

Out[ ]:

	train-recall	validation-recall	train-f1-score	validation-f1-score	train-precision	validation-precision	train-roc-auc	validation-roc-auc	train-accurecy	validation-accurecy
NN with SGD	0.434081	0.432343	0.545134	0.544699	0.732544	0.735955	0.696768	0.696539	0.852429	0.854
NN with Adam	0.570126	0.471947	0.656704	0.573146	0.774286	0.729592	0.763804	0.713835	0.878571	0.858
NN with Adam With Dropout	0.508415	0.471947	0.605175	0.576613	0.747423	0.740933	0.732231	0.715088	0.864857	0.86
NN with SGD With SMOTE OverSampled Data	0.82526	0.69967	0.813943	0.541507	0.802932	0.441667	0.811356	0.737888	0.811356	0.760667
NN with Adam With SMOTE OverSampled Data	0.90976	0.69967	0.862562	0.528678	0.820019	0.42485	0.855041	0.729952	0.855041	0.748
NN with Adam With SMOTE OverSampled Data and DropOuts	0.868855	0.785479	0.825887	0.579781	0.786968	0.459459	0.816828	0.77578	0.816828	0.77
NN using LeakyReLU with Adam With SMOTE OverSampled Data and DropOuts	0.809473	0.693069	0.823508	0.587413	0.838039	0.509709	0.826516	0.762157	0.826516	0.803333

Observations

- Loss & Recall Plots
    - The training and validation loss decrease steadily and align well, suggesting no significant overfitting.
    - The validation loss exhibits some oscillations after a certain point, indicating potential instability in learning but overall convergence.
    - The recall for training data improves consistently and stabilizes over epochs, showing the model's ability to generalize well to the training data.
    - Validation recall fluctuates but stays close to the training recall, indicating acceptable generalization.
- Evaluation Metrics
    - The model achieves a good balance between recall, precision, and F1-score on both training and validation sets.
    - Validation accuracy and recall show consistent performance with acceptable differences from the training metrics.
- Points to Note
    - The model performs well on training data and shows balanced metrics, suggesting effective learning from the oversampled data. -The overall accuracy is boosted by correct predictions for the majority class (class 0), but minority class performance (recall) remains the primary area of concern.
    - The class imbalance is still affecting the model's performance on the minority class, even with SMOTE oversampling.

Model Performance Comparison and Final Model Selection

Utility function to plot confusion Matrix and Other metrics for test data-set

In [ ]:

```
test_metric = pd.DataFrame(columns=["recall", 'f1-score', 'precision', 'roc-auc', 'accurecy'])
def plot_confusion_matrix_for_test(model, X, y, model_name):
    """
    To plot the confusion_matrix with percentages

    actual_targets: actual target (dependent) variable values
    predicted_targets: predicted target (dependent) variable values
    """
    print(X.shape)
    print(y.shape)

    print('Confusion Matrix - ' + model_name + ' - TEST')
    y_pred = model.predict(X)
    y_pred = (y_pred > 0.5)

    test_metric.loc[model_name, 'recall'] = recall_score(y, y_pred)
    test_metric.loc[model_name, 'f1-score'] = f1_score(y, y_pred)
    test_metric.loc[model_name, 'precision'] = precision_score(y, y_pred)
    test_metric.loc[model_name, 'roc-auc'] = roc_auc_score(y, y_pred)
    test_metric.loc[model_name, 'accurecy'] = accuracy_score(y, y_pred)
```

```
    cm = confusion_matrix(y, y_pred)
    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")
    plt.show()
    cr = classification_report(y, y_pred)
    print(cr)
```

Model Comparison & Selection

Based on the evaluation metrics and our prioritization criteria (Recall > Precision > F1-Score > ROC-AUC > Accuracy), below are the model comparison:

1. **NN with SGD**:
    - **Validation Recall**: 0.4323 (lowest among all models).
    - **Validation Precision**: 0.7359 (second highest).
    - **Validation F1-Score**: 0.5451.
    - **Validation ROC-AUC**: 0.6965.
    - **Validation Accuracy**: 0.854.
    - **Conclusion**: Poor recall performance disqualifies this model for production use.
2. **NN with Adam**:
    - **Validation Recall**: 0.4719 (still low, but better than SGD).
    - **Validation Precision**: 0.7296.
    - **Validation F1-Score**: 0.5731.
    - **Validation ROC-AUC**: 0.7138.
    - **Validation Accuracy**: 0.858.
    - **Conclusion**: While precision and accuracy are decent, low recall makes this model less suitable for recall-prioritized scenarios.
3. **NN with Adam and Dropouts**:
    - **Validation Recall**: 0.4719 (same as NN with Adam).
    - **Validation Precision**: 0.7409.
    - **Validation F1-Score**: 0.5766.
    - **Validation ROC-AUC**: 0.7151.
    - **Validation Accuracy**: 0.86.
    - **Conclusion**: Minor improvements in precision and F1-score compared to "NN with Adam" but recall remains unchanged.
4. **NN with SGD with SMOTE Oversampled Data**:
    - **Validation Recall**: 0.6996 (significantly better recall).
    - **Validation Precision**: 0.4417 (very low, indicating false positives are high).
    - **Validation F1-Score**: 0.5415.
    - **Validation ROC-AUC**: 0.7379.
    - **Validation Accuracy**: 0.7607.
    - **Conclusion**: High recall but poor precision and F1-score make this model less desirable despite recall priority.
5. **NN with Adam with SMOTE Oversampled Data**:
    - **Validation Recall**: 0.6996 (same recall as SGD with SMOTE).
    - **Validation Precision**: 0.4248 (lower than SGD with SMOTE).
    - **Validation F1-Score**: 0.5287.
    - **Validation ROC-AUC**: 0.7299.
    - **Validation Accuracy**: 0.748.
    - **Conclusion**: Slightly worse overall than "NN with SGD with SMOTE."
6. **NN with Adam with SMOTE Oversampled Data and DropOuts**:
    - **Validation Recall**: 0.7855 (highest among all models).
    - **Validation Precision**: 0.4594.
    - **Validation F1-Score**: 0.5798.
    - **Validation ROC-AUC**: 0.7757 (best among all models).
    - **Validation Accuracy**: 0.77.
    - **Conclusion**: This model balances high recall with a reasonable trade-off in other metrics. The highest recall makes it a strong candidate for production if recall is prioritized.

7. **NN using LeakyReLU with Adam with SMOTE Oversampled Data and DropOuts**:

   - **Validation Recall**: 0.6930 (lower than model 6 but still strong).
   - **Validation Precision**: 0.5097 (best among all models).
   - **Validation F1-Score**: 0.5874.
   - **Validation ROC-AUC**: 0.7622.
   - **Validation Accuracy**: 0.8033.
   - **Conclusion**: Improved precision over previous model, but recall is slightly lower.

Model Selection

Given the prioritization criteria **(Recall > Precision > F1-Score > ROC-AUC > Accuracy)**:

- **Best Model**: **NN with Adam with SMOTE Oversampled Data and DropOuts**
  - Highest **validation recall**: 0.7855.
  - Reasonable precision: 0.4594.
  - Balanced F1-score: 0.5798.
  - Strong ROC-AUC: 0.7757.
- **Alternative Option**: **NN using LeakyReLU and Adam with SMOTE Oversampled Data and DropOuts**
  - Slightly lower recall (0.6930) but better precision (0.5097) and higher F1-score (0.5874).

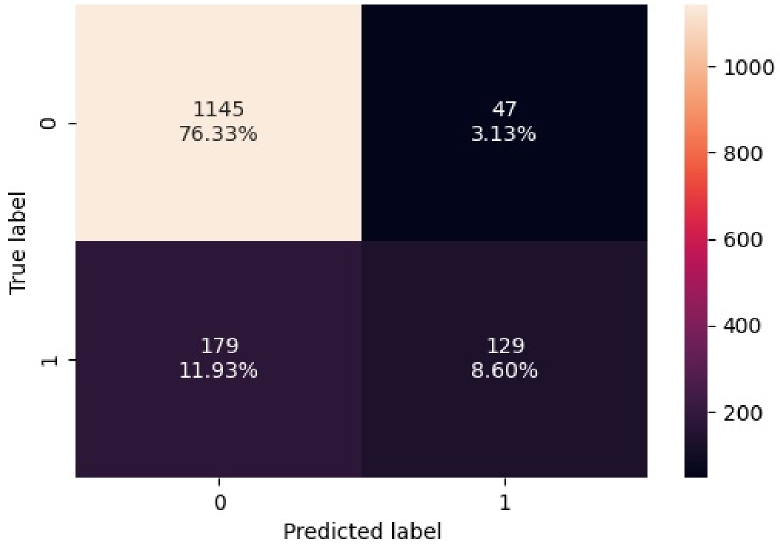
Final Decision:

- Will choose **NN with Adam with SMOTE Oversampled Data and DropOuts** for scenarios where maximizing recall is critical.
- Will consider **NN using LeakyReLU with Adam with SMOTE Oversampled Data and DropOuts** if a better balance between recall and precision is required.

Evaluation of all models on Test Data-Set

In [ ]:
```python
plot_confusion_matrix_for_test(model_0, X_test, y_test, 'model_0')
```

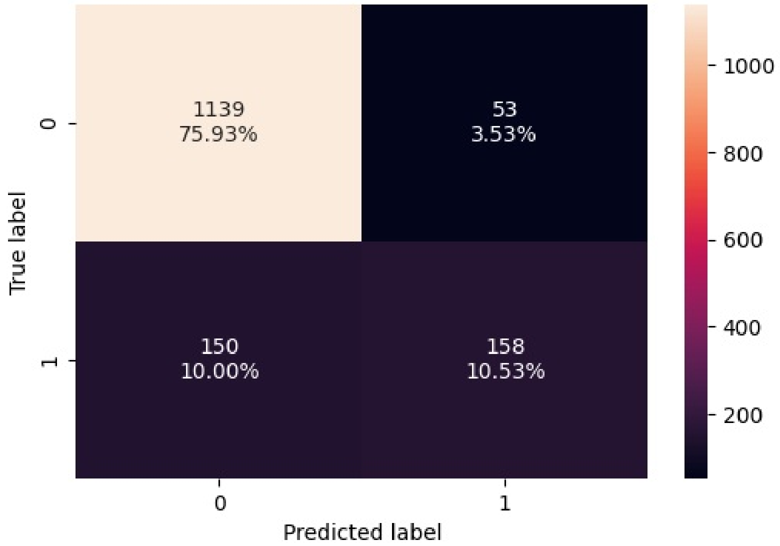
```
(1500, 11)
(1500, 1)
Confusion Matrix - model_0 - TEST
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.86      0.96      0.91      1192
         1.0       0.73      0.42      0.53       308

    accuracy                           0.85      1500
   macro avg       0.80      0.69      0.72      1500
weighted avg       0.84      0.85      0.83      1500
```

In [ ]:
```python
plot_confusion_matrix_for_test(model_1, X_test, y_test, 'model_1')
```

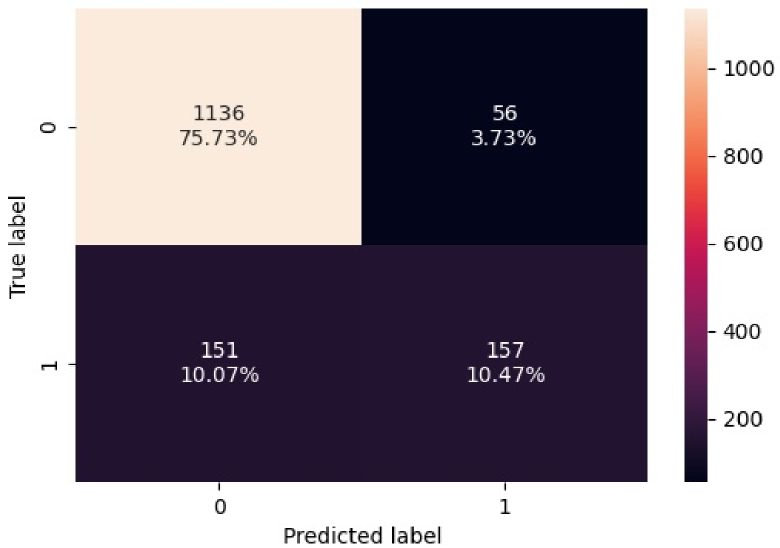
```
(1500, 11)
(1500, 1)
Confusion Matrix - model_1 - TEST
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.88      0.96      0.92      1192
         1.0       0.75      0.51      0.61       308

    accuracy                           0.86      1500
   macro avg       0.82      0.73      0.76      1500
weighted avg       0.86      0.86      0.85      1500
```

In [ ]:
```
plot_confusion_matrix_for_test(model_2, X_test, y_test, 'model_2')
```

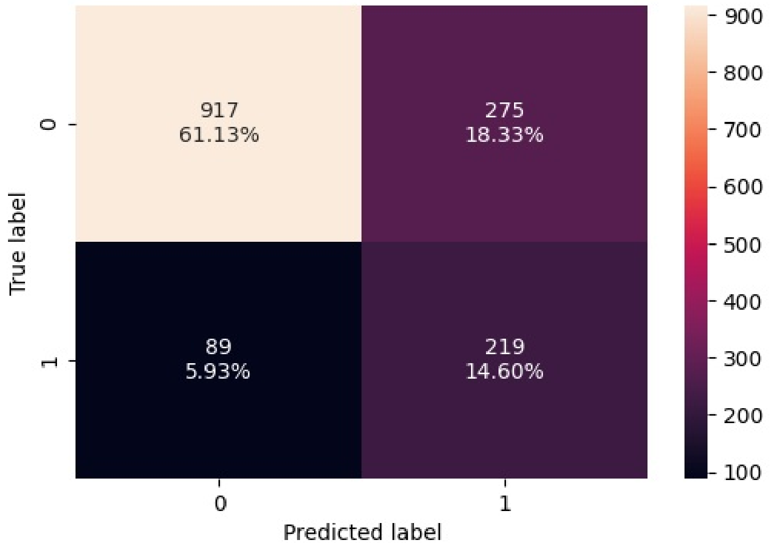
```
(1500, 11)
(1500, 1)
Confusion Matrix - model_2 - TEST
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.88      0.95      0.92      1192
         1.0       0.74      0.51      0.60       308

    accuracy                           0.86      1500
   macro avg       0.81      0.73      0.76      1500
weighted avg       0.85      0.86      0.85      1500
```

In [ ]:
```
plot_confusion_matrix_for_test(model_3, X_test, y_test, 'model_3')
```

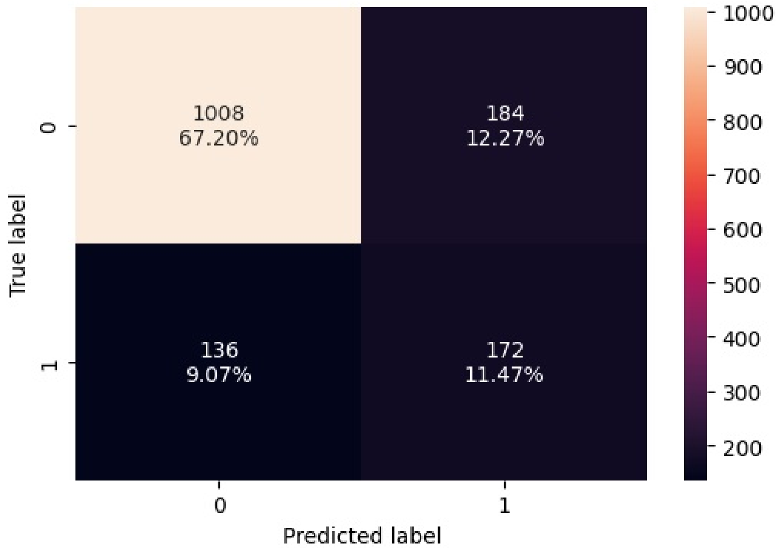
```
(1500, 11)
(1500, 1)
Confusion Matrix - model_3 - TEST
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.91      0.77      0.83      1192
         1.0       0.44      0.71      0.55       308

    accuracy                           0.76      1500
   macro avg       0.68      0.74      0.69      1500
weighted avg       0.82      0.76      0.78      1500
```

In [ ]:
```
plot_confusion_matrix_for_test(model_4, X_test, y_test, 'model_4')
```

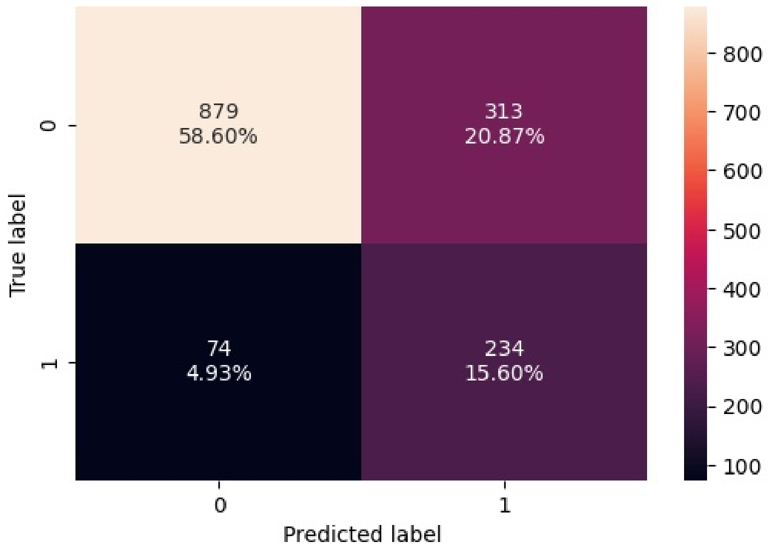
```
(1500, 11)
(1500, 1)
Confusion Matrix - model_4 - TEST
47/47 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.88      0.85      0.86      1192
         1.0       0.48      0.56      0.52       308

    accuracy                           0.79      1500
   macro avg       0.68      0.70      0.69      1500
weighted avg       0.80      0.79      0.79      1500
```

In [ ]:
```
plot_confusion_matrix_for_test(model_5, X_test, y_test, 'model_5')
```

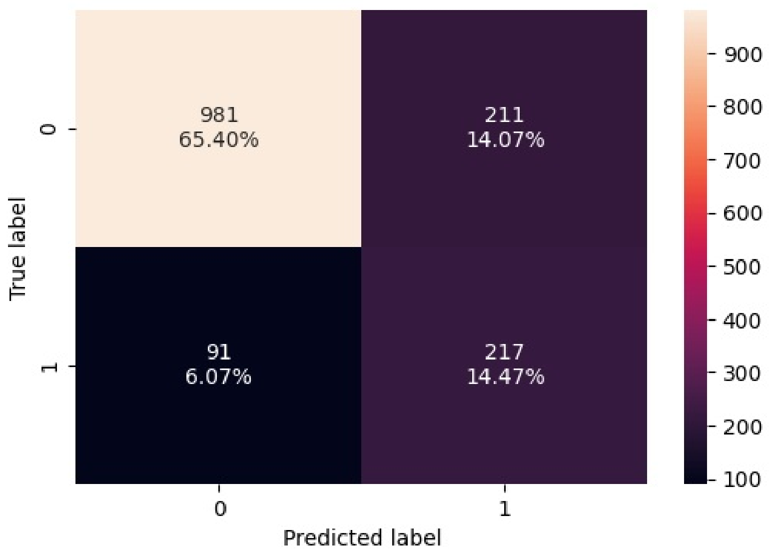
```
(1500, 11)
(1500, 1)
Confusion Matrix - model_5 - TEST
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.92      0.74      0.82      1192
         1.0       0.43      0.76      0.55       308

    accuracy                           0.74      1500
   macro avg       0.68      0.75      0.68      1500
weighted avg       0.82      0.74      0.76      1500
```

In [ ]:
```
plot_confusion_matrix_for_test(model_6, X_test, y_test, 'model_6')
```

```
(1500, 11)
(1500, 1)
Confusion Matrix - model_6 - TEST
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




```
              precision    recall  f1-score   support

         0.0       0.92      0.82      0.87      1192
         1.0       0.51      0.70      0.59       308

    accuracy                           0.80      1500
   macro avg       0.71      0.76      0.73      1500
weighted avg       0.83      0.80      0.81      1500
```

In [ ]:
```
test_metric
```

Out[ ]:

	recall	f1-score	precision	roc-auc	accurecy
model_1	0.512987	0.608863	0.748815	0.734262	0.864667
model_0	0.418831	0.533058	0.732955	0.689701	0.849333
model_2	0.50974	0.602687	0.737089	0.73138	0.862
model_3	0.711039	0.546135	0.44332	0.740167	0.757333
model_4	0.558442	0.518072	0.483146	0.70204	0.786667
model_5	0.75974	0.547368	0.427788	0.748578	0.742
model_6	0.704545	0.589674	0.507009	0.763766	0.798667

Analysis of Test Dataset Evaluation Metrics

Model 0 (NN with SGD)

- **Recall**: 0.419 (lowest recall).
- **Precision**: 0.733.
- **F1-Score**: 0.533 (lowest).
- **ROC-AUC**: 0.690 (lowest).
- **Accuracy**: 0.849.
- **Summary**: Model 0 underperforms across most metrics and is not suitable for production.

Model 1 (NN with Adam)

- **Recall**: 0.513 (moderate).
- **Precision**: 0.749 (highest precision among all models).
- **F1-Score**: 0.609.
- **ROC-AUC**: 0.734.
- **Accuracy**: 0.865 (highest accuracy among all models).
- **Summary**: Model 1 balances precision and accuracy but sacrifices recall, making it less suitable for scenarios prioritizing recall.

Model 2 (NN with Adam With Dropout)

- **Recall**: 0.510.
- **Precision**: 0.737.
- **F1-Score**: 0.603.
- **ROC-AUC**: 0.731.
- **Accuracy**: 0.862.
- **Summary**: Similar to Model 1 but with slightly lower recall and overall weaker metrics.

Model 3 (NN with SGD With SMOTE OverSampled Data)

- **Recall**: 0.711 ((second highest recall)).
- **Precision**: 0.443
- **F1-Score**: 0.546.
- **ROC-AUC**: 0.740.
- **Accuracy**: 0.757.
- **Summary**: High recall but significantly low precision, resulting in poor F1-score. Suitable for recall-dominant predictions despite trade-offs.

Model 4 (NN with Adam With SMOTE OverSampled Data)

- **Recall**: 0.558.
- **Precision**: 0.483.
- **F1-Score**: 0.518.
- **ROC-AUC**: 0.702 (second lowest).
- **Accuracy**: 0.787.
- **Summary**: Balanced but mediocre metrics across the board; underwhelming for both recall and precision priorities.

Model 5 (NN with Adam and SMOTE OverSampled Data with DropOuts)

- **Recall**: 0.760 (highest recall)
- **Precision**: 0.428 (lowest precision)
- **F1-Score**: 0.547.
- **ROC-AUC**: 0.749 (highest ROC-AUC).
- **Accuracy**: 0.742 (low).
- **Summary**: High recall and ROC-AUC but low precision and accuracy. A viable option for recall-dominant use cases.

Model 6 (NN using LeakyReLU with Adam With SMOTE OverSampled Data and DropOuts)

- **Recall**: 0.705 (third highest recall).
- **Precision**: 0.507.
- **F1-Score**: 0.590 (second highest F1-score).
- **ROC-AUC**: 0.764.
- **Accuracy**: 0.799.
- **Summary**: Balanced metrics with strong recall and acceptable precision. A solid contender for general use with recall emphasis.

Best Model for Recall Prioritization:

- **(NN with Adam and SMOTE OverSampled Data with DropOuts)** (highest recall: 0.760, despite low precision).

Best Balanced Model:

- **(NN using LeakyReLU with Adam with SMOTE Oversampled Data and DropOuts)** (strong recall: 0.705, with improved precision and F1-score).

Actionable Insights and Business Recommendations

- **Insights**
  - Models such as **(NN with Adam and SMOTE OverSampled Data with DropOuts)** and **(NN using LeakyReLU with Adam with SMOTE Oversampled Data and DropOuts)**, which prioritize recall, are effective in identifying customers at risk of churn. These models minimize false negatives, ensuring most at-risk customers are flagged for intervention.
  - The use of SMOTE oversampling improved the model's performance on the minority class (customers likely to churn). Future iterations should continue addressing class imbalance for robust predictions.
  - Incorporating LeakyReLU activation in the last model resulted in better recall and F1-score while maintaining reasonable recall.
  - Regularization techniques like dropout reduced overfitting and enhanced generalization.
  - Performance metrics such as validation recall and F1-score indicate potential improvements. Monitoring model performance over time with new data is critical to ensure relevance.
- **Business Recommendations**
  - Proactive Customer Retention Campaigns: Use high-recall models to identify customers at risk of churn and prioritize them for retention strategies.
  - Risked Customer Segmentation: Segment flagged customers by their churn probability (e.g., high, medium, low). Focus specialized retention strategies on customers with the highest likelihood of churn.
  - Analyze customers journey: Identify the features most correlated with churn predictions (e.g., tenure, product usage) to identify pain points in the customer journey. Address these systematically to reduce churn.
  - It is important to do Cost-Benefit Analysis of Interventions and Retention Campaigns.
  - Continuously monitor the model's performance over time. It is important to gather more relevant and updated customer data to revise the models for better prediction capabilities.
