Bank Churn Prediction

Problem Statement

Context

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

Objective

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

Data Dictionary

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

Importing necessary libraries

In [ ]:
```
#Installing the libraries with the specified version.
!pip install tensorflow==2.15.0 scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.24.1 pandas==1.5
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 475.2/475.2 MB 3.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.6/9.6 MB 34.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 55.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.1/12.1 MB 42.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 30.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 36.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 39.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 442.0/442.0 kB 12.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 2.2 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This
behaviour is the source of the following dependency conflicts.
albumentations 1.4.14 requires numpy>=1.24.4, but you have numpy 1.24.1 which is incompatible.
cudf-cu12 24.4.1 requires pandas<2.2.2dev0,>=2.0, but you have pandas 1.5.3 which is incompatible.
google-colab 1.0.0 requires pandas==2.1.4, but you have pandas 1.5.3 which is incompatible.
pandas-stubs 2.1.4.231227 requires numpy>=1.26.0; python_version < "3.13", but you have numpy 1.24.1 which is i
ncompatible.
tensorstore 0.1.65 requires ml-dtypes>=0.3.1, but you have ml-dtypes 0.2.0 which is incompatible.
tf-keras 2.17.0 requires tensorflow<2.18,>=2.17, but you have tensorflow 2.15.0 which is incompatible.
xarray 2024.6.0 requires pandas>=2.0, but you have pandas 1.5.3 which is incompatible.
```

In [1]:
```
# Library for data manipulation and analysis.
```

```
import pandas as pd
# Fundamental package for scientific computing.
import numpy as np
#splitting datasets into training and testing sets.
from sklearn.model_selection import train_test_split
#Imports tools for data preprocessing including label encoding, one-hot encoding, and standard scaling
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,StandardScaler
#Imports a class for imputing missing values in datasets.
from sklearn.impute import SimpleImputer
#Imports the Matplotlib library for creating visualizations.
import matplotlib.pyplot as plt
# Imports the Seaborn library for statistical data visualization.
import seaborn as sns
# Time related functions.
import time
#Imports functions for evaluating the performance of machine learning models
from sklearn.metrics import confusion_matrix, f1_score,accuracy_score, recall_score, precision_score, classific


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

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

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

Loading the dataset

In [3]:
```
# loading the dataset
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Churn.csv")
```

Data Overview

In [4]:
```
# Displaying the first 5 rows of the dataset
df.head()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMemb
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	

In [5]:
```
# Displaying the last 5 rows of the data set
df.tail()
```

Out[5]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

In [6]:
```
# Checking the shape of the dataset
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns.")
```

```
There are 10000 rows and 14 columns.
```

In [7]:
```
# Displaying 10 random rows from the dataset
df.sample(n=10, random_state=1)
```

Out[7]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsA
9953	9954	15655952	Burke	550	France	Male	47	2	0.00	2	1	
3850	3851	15775293	Stephenson	680	France	Male	34	3	143292.95	1	1	
4962	4963	15665088	Gordon	531	France	Female	42	2	0.00	2	0	
3886	3887	15720941	Tien	710	Germany	Male	34	8	147833.30	2	0	
5437	5438	15733476	Gonzalez	543	Germany	Male	30	6	73481.05	1	1	
8517	8518	15671800	Robinson	688	France	Male	20	8	137624.40	2	1	
2041	2042	15709846	Yeh	840	France	Female	39	1	94968.97	1	1	
1989	1990	15622454	Zaitsev	695	Spain	Male	28	0	96020.86	1	1	
1933	1934	15815560	Bogle	666	Germany	Male	74	7	105102.50	1	1	
9984	9985	15696175	Echezonachukwu	602	Germany	Male	35	7	90602.42	2	1	

In [8]:

```
# Checking the data types & non-null values of the dataset
df.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
```

- There are 0 missing values throughout the dataset.
- The datatypes that are present are: int, object, and float.

Checking for any duplicate values

In [9]:

```
df.duplicated().sum()
```

Out[9]:

```
0
```

- There are not values that are exactly the same throughout the data.

Checking for missing values

In [10]:

```
df.isnull().sum()
```

Out[10]:

	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

- There are no missing values present in any of the columns

In [11]:

```
df["Exited"].value_counts()
```

Out[11]:

Exited	count
0	7963
1	2037

dtype: int64

- There is a significant imballance between those who are still with the bank and customers who have left. An approx 80/20 ratio.

In [12]:

```
# Statistical summary of the numerical columns in the data
df.describe().T
```

Out[12]:

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

- The max age is 92, which is a significant high age to be a part of a bank.
- RowNumber & CustomerID are columns that should be removed due to their statiscital summary consisting of numbers thata are random and irrelevant.

In [13]:

```
# Check the unique values in each column
df.nunique()
```

Out[13]:

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

dtype: int64

- For the surname unique count being a low amount compared ot the total number of rows in teh data set, the shows that there are a number of customers who share the same surnames.

In [14]:
```
for i in df.describe(include=["object"]).columns:
    print("Unique values in", i, "are :")
    print(df[i].value_counts())
    print("*" * 50)
```

```
Unique values in Surname are :
Surname
Smith       32
Scott       29
Martin      29
Walker      28
Brown       26
            ..
Izmailov     1
Bold         1
Bonham       1
Poninski     1
Burbidge     1
Name: count, Length: 2932, dtype: int64
**************************************************
Unique values in Geography are :
Geography
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
**************************************************
Unique values in Gender are :
Gender
Male      5457
Female    4543
Name: count, dtype: int64
**************************************************
```

- France has the most customers that reside, almost double the amount of each germany and spain residents.
- There is almost a thousand indivual differnce between male and female cusotmers in this data set.

Exploratory Data Analysis

Functions to help with visualizing the EDA

In [15]:
```
# 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))
```

```python
    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(
        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 [16]:

```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 [17]:

```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(
```

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

In [18]:
```python
# 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 star will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

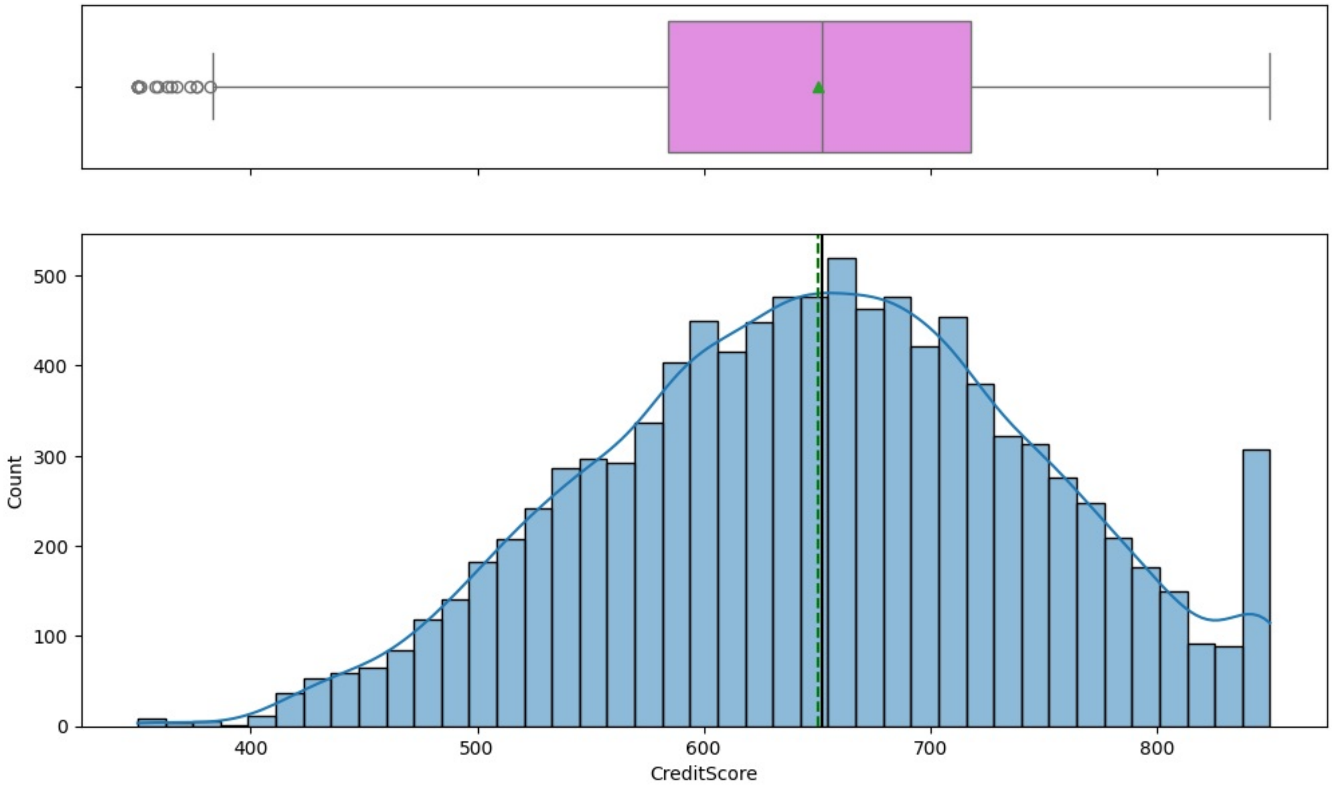
In [19]:
```python
### Function to plot distributions


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

`CreditScore`

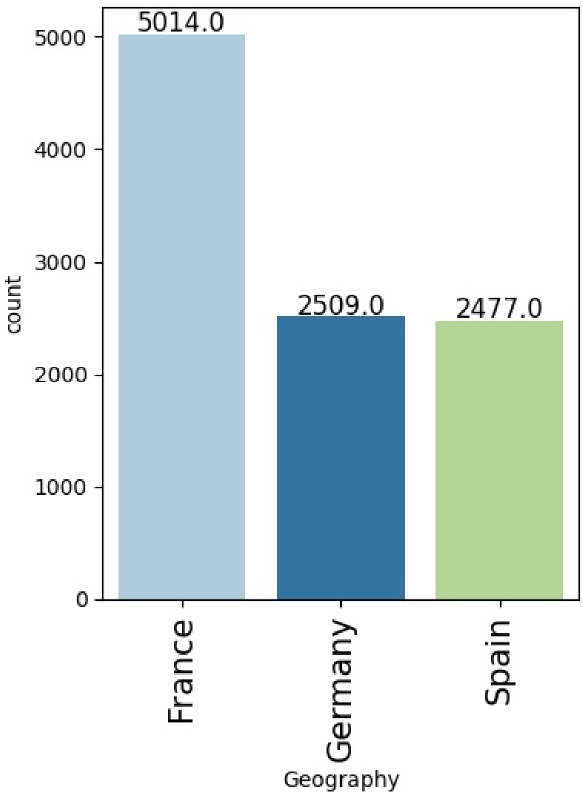
In [20]:
```
histogram_boxplot(df, "CreditScore", kde=True)
```



- The distribution is skewed a little towards the left as well as slighltly normally distributed.

`Geography`

In [21]:
```
labeled_barplot(df, "Geography")
```

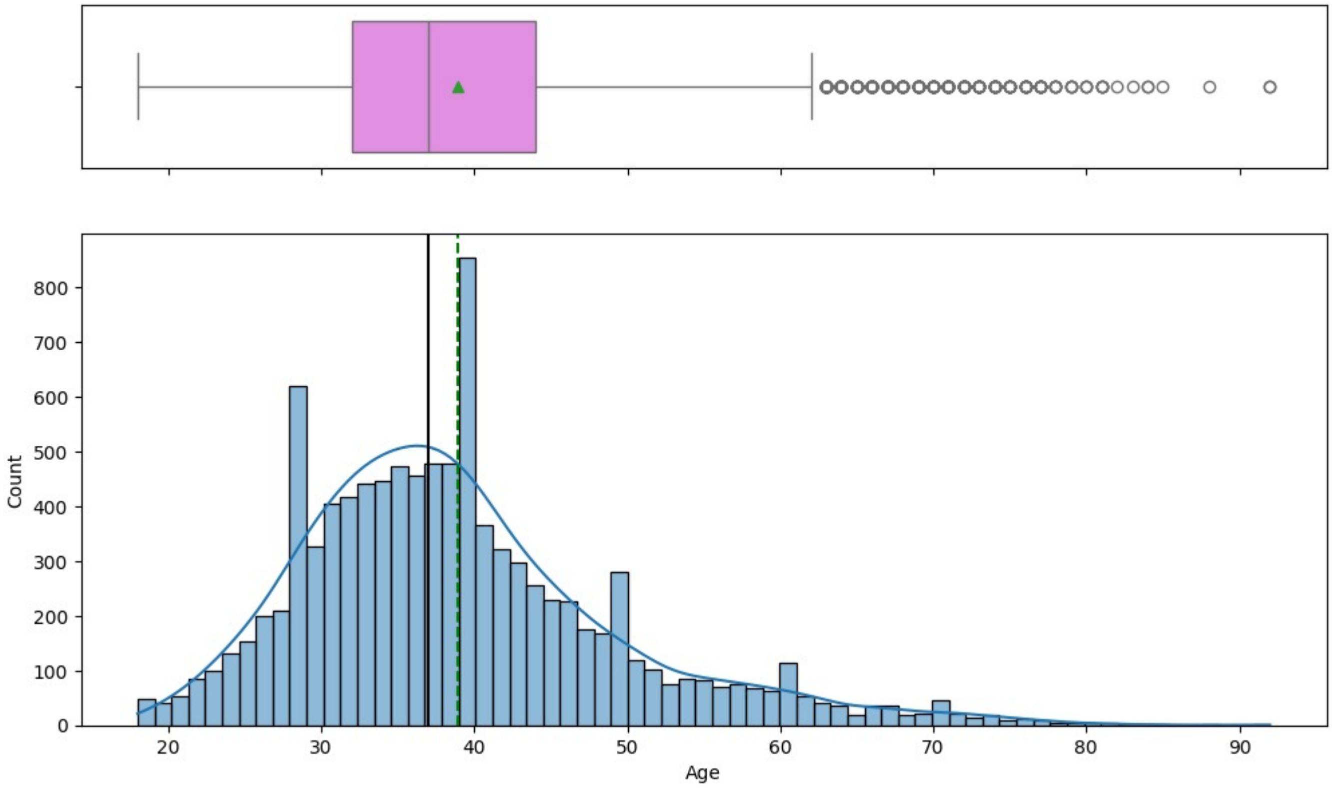


- From the dataset, there are twice the amount of customers living in france compared to Germany & Spain

- From the dataset, there are twice the amount of customers living in france compared to Germany & Spain.

`Age`

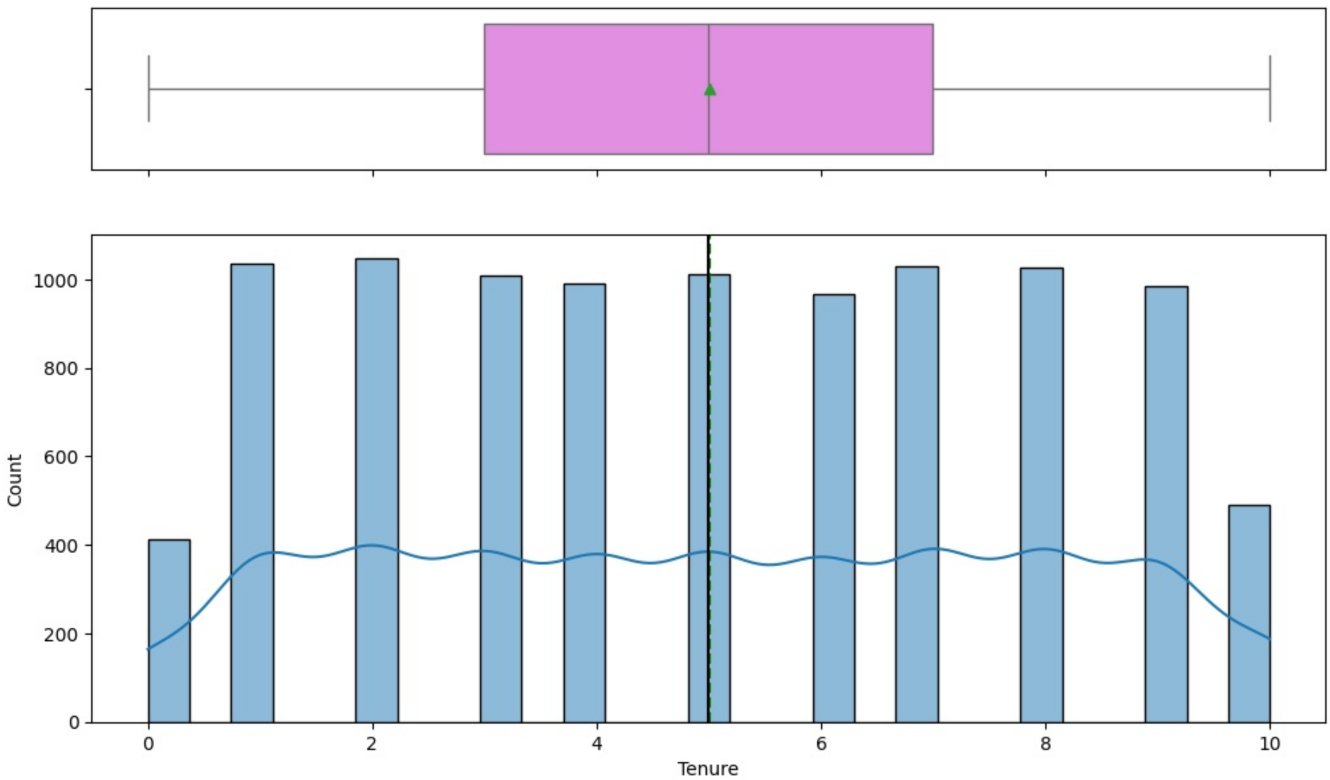
In [22]:
```
histogram_boxplot(df, "Age", kde=True)
```



- The data is skewed to the right.
- The average age of customers is around 35-38 years old.

`Tenure`

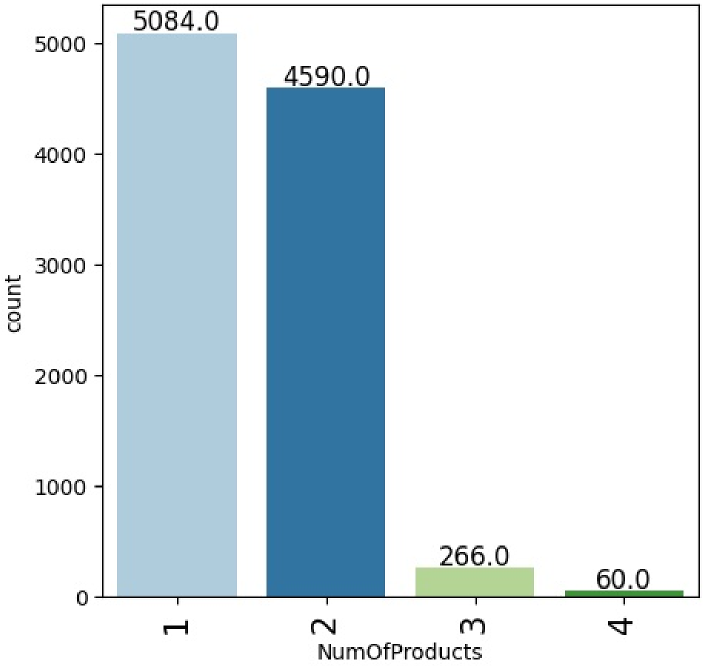
In [23]:
```
histogram_boxplot(df, "Tenure", kde=True)
```



- The tenure columns seems to be normally distributed with the average years for which teh customer has been with the bank is 5.

`NumOfProducts`

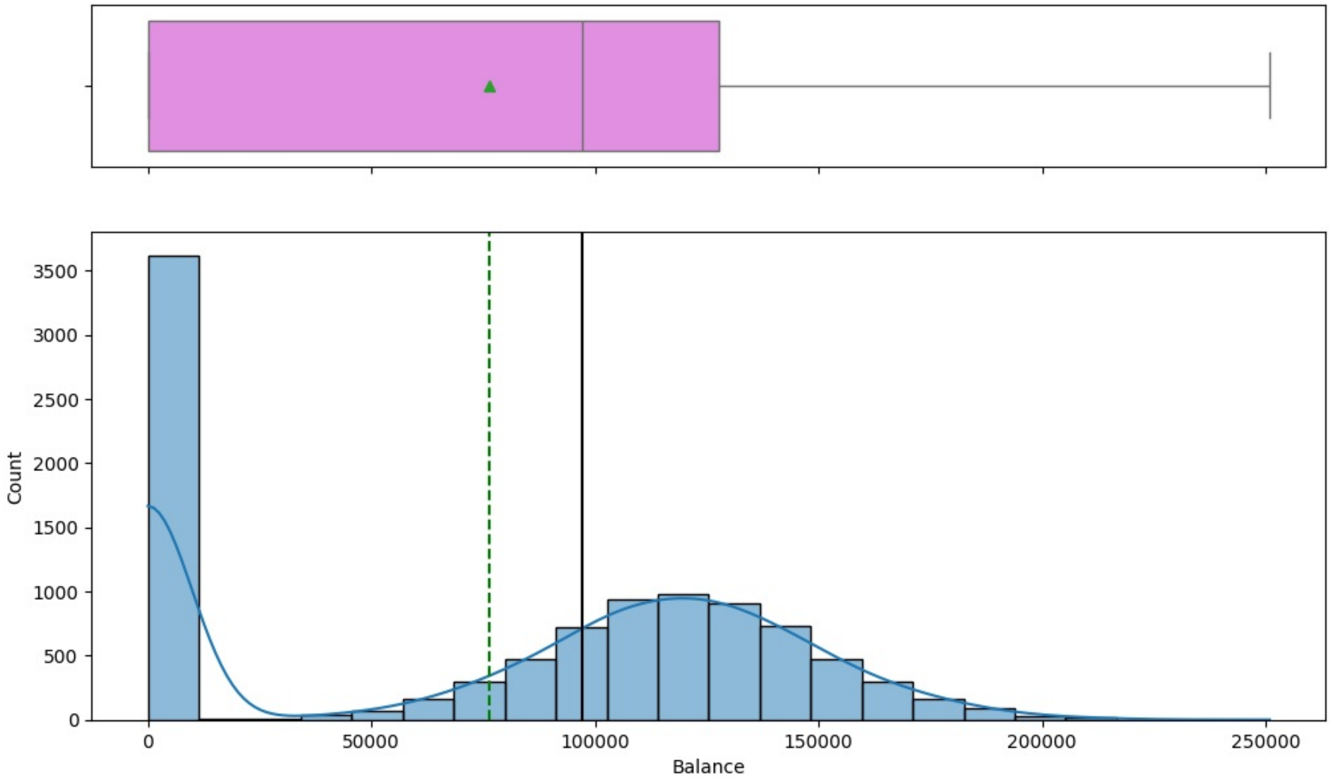
In [24]:
```
labeled_barplot(df, "NumOfProducts")
```



- As seen above, there a significant amount of customers who purchase between 1 to 2 procuts through the bank, most customers do not buy more than that amount.

`Balance`

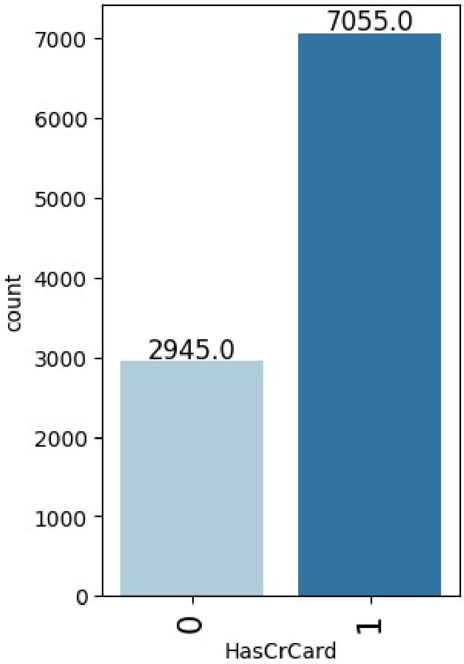
In [25]:
```
histogram_boxplot(df, "Balance", kde=True)
```



- From the above histogram & boxplot, there are a vast amount of customers who have a balance of 0 which seems to be a mistake in the dataset.
- Excluding the 0 count, the data is normally distributed with no skewness, as seen in the histogram.

`HasCrCard`

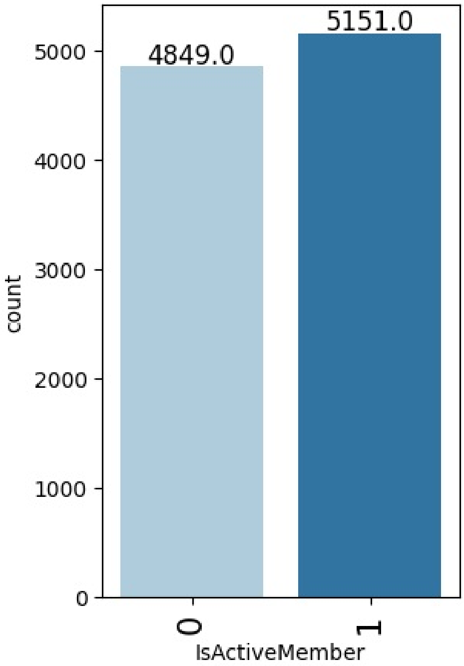
In [26]:
```
labeled_barplot(df, "HasCrCard")
```



- There is more than twice the amount of customers who have credit cards than those who do not.

`isActiveMember`

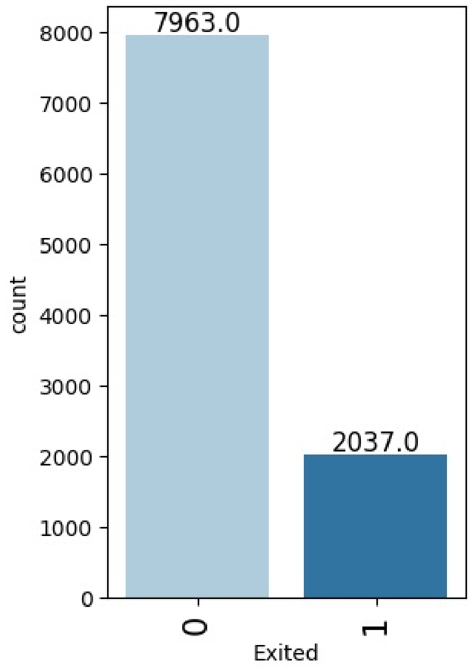
In [27]:
```
labeled_barplot(df, "IsActiveMember")
```



- There is almost a balance between customers who are active compared to those who are not.

`Exited`

In [28]:
```
labeled_barplot(df, "Exited")
```

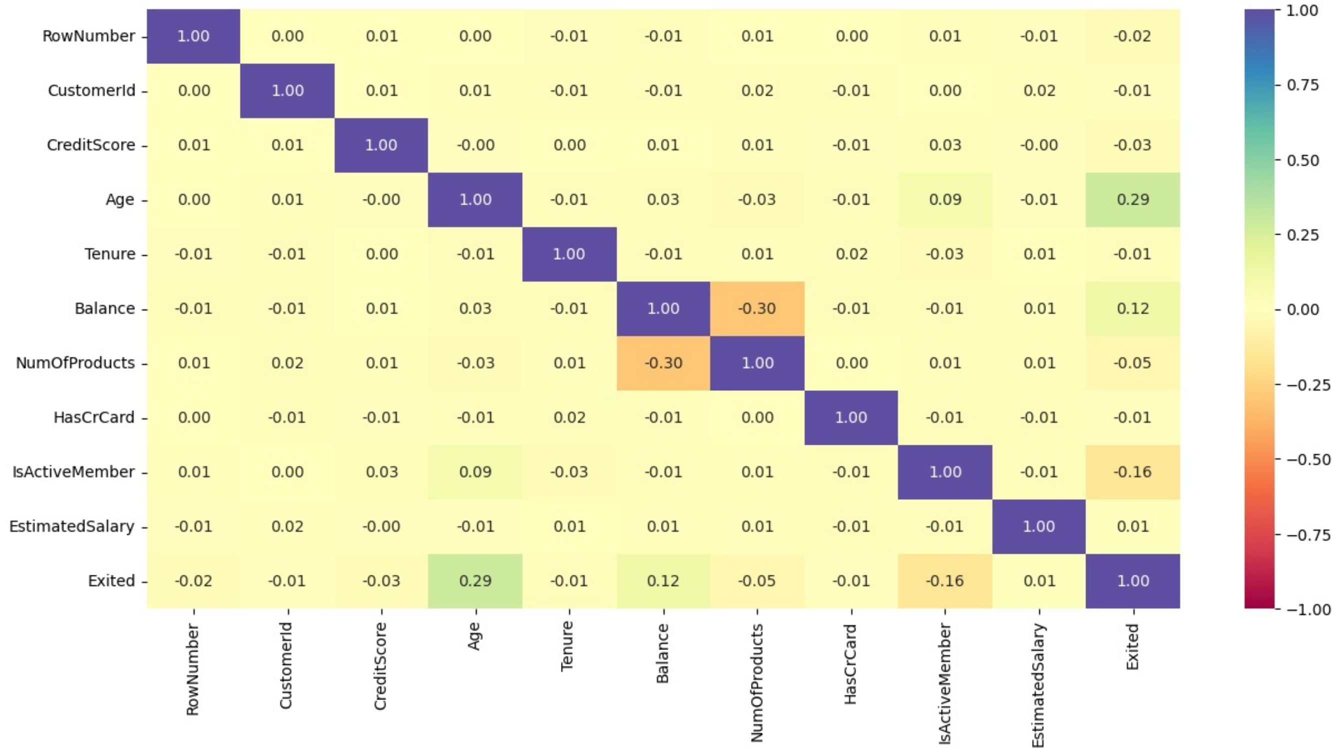


- Compared to the previous variable, the exited variable has a 75 to 25 ratio for those who have not left the bank to those who have.
- For this being the target variable, and more than 3 times the amount of customers have not left the bank in this dataset, this is a good result of the bank.

Bivariate Analysis

Correlation between numerical variables

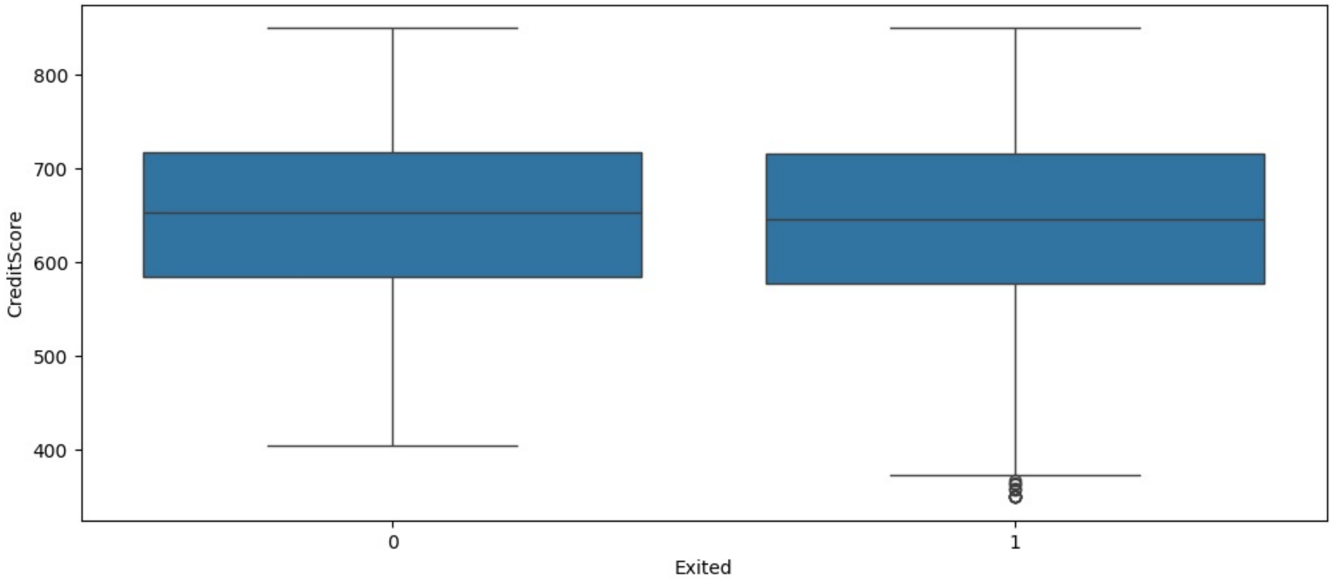
In [29]:
```
numerical_df = df.select_dtypes(include=['number'])
plt.figure(figsize=(15, 7))
sns.heatmap(
    numerical_df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



- As seen in the above heatmap, there are no pairs of variables that are highly correlated.
- However, the Age & Exited variables are the only outstanding from the group that contains the highest correlation.
  - That being said, the more aged the customer is the more likely they will not leave the bank. On the contrary, the less the age of the customer is the more likely the customer will leave the bank.

`Exited` vs `CreditScore`

In [30]:
```
plt.figure(figsize=(12, 5))
sns.boxplot(x="Exited", y="CreditScore", data=df)  # Use sns.boxplot for a boxplot
plt.show()
```

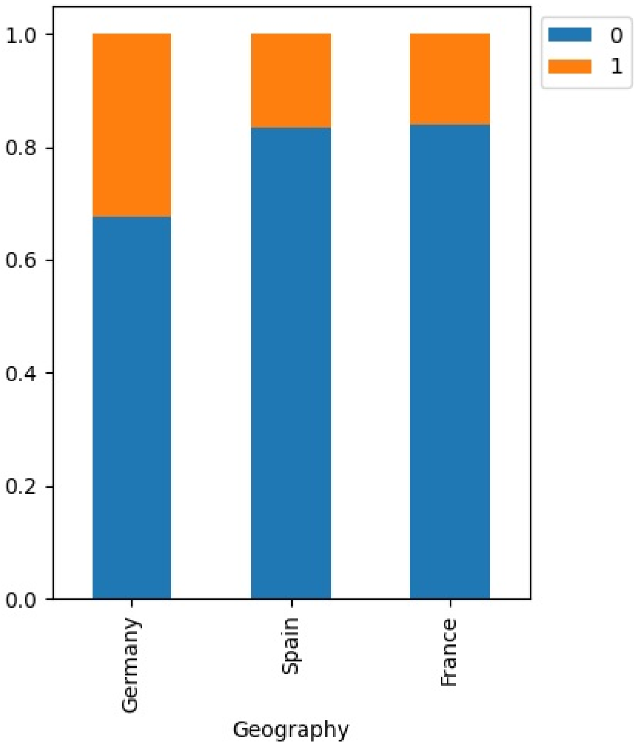


- As seen above, there is no difference on the credit score for customers who are still apart with the bank and those who are not.

`Exited` vs `Geography`

In [31]:
```
stacked_barplot(df, "Geography", "Exited")
```

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

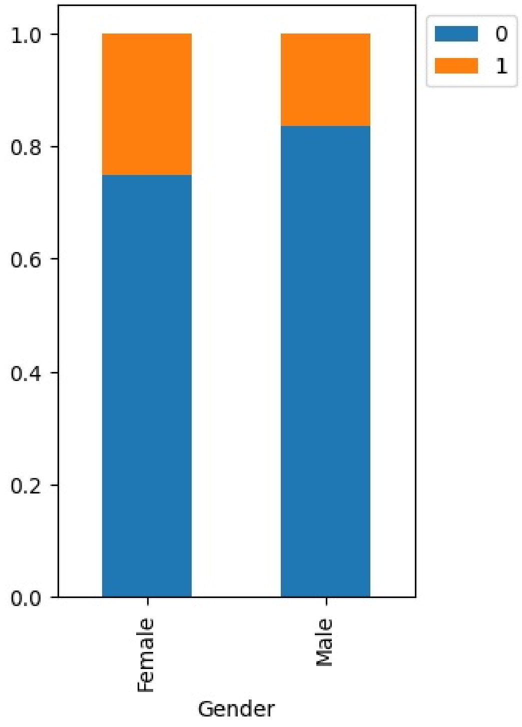


- From the above barplot, those customers who reside in Germany have the most amount of leaving the bank.

`Exited` vs `Gender`

In [32]:
```
# Exited vs gender analysis
stacked_barplot(df, "Gender", "Exited")
```

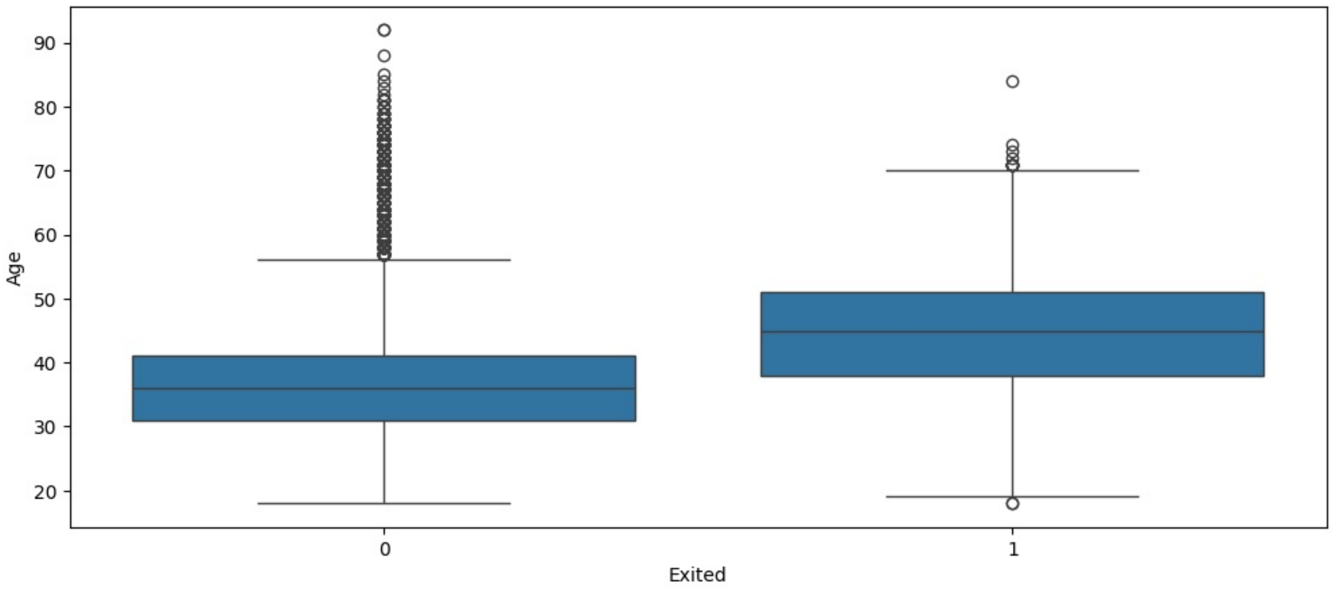
```
Exited     0     1    All
Gender
All     7963  2037  10000
Female  3404  1139   4543
Male    4559   898   5457
--------------------------------------------------------------------------------------------------------------
```



- The male and femal for those who are still with the bank and are not do not have too much of a differne. Although, there are more males who are still with the bank than those who are not.

`Exited` vs `Age`

In [33]:
```
# Exited and age analyis
plt.figure(figsize=(12, 5))
sns.boxplot(x="Exited", y="Age", data=df)  # Use sns.boxplot for a boxplot
plt.show()
```

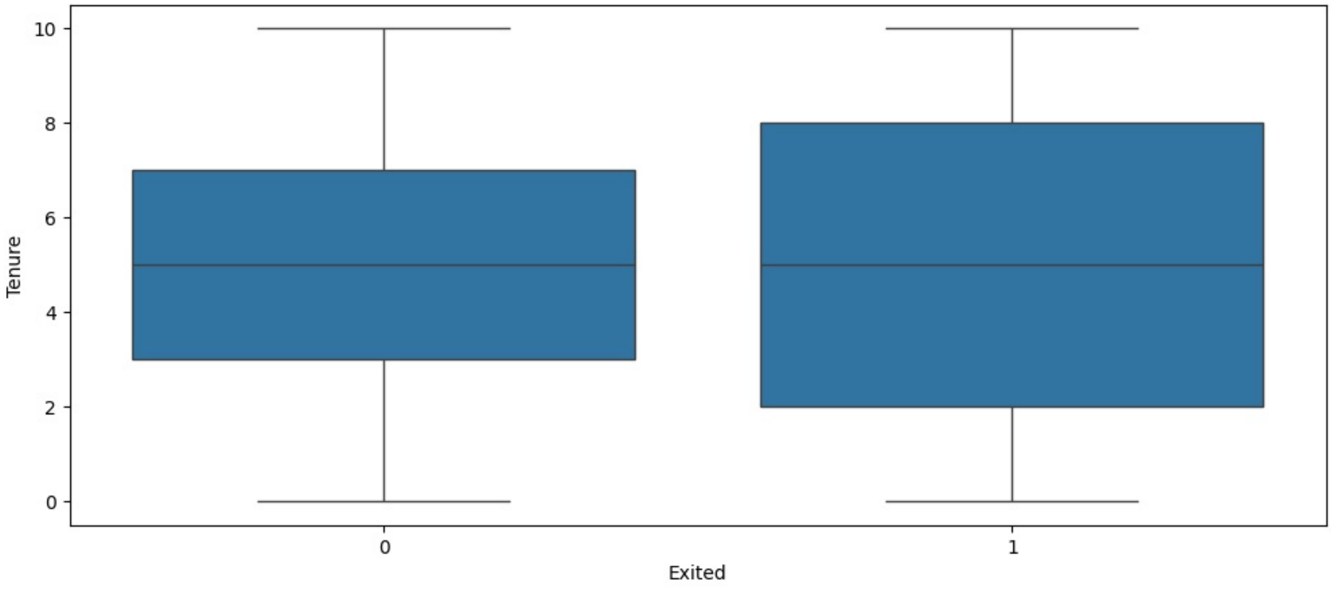


- From the above bar plot, those customers who have left the bank tend to be much older than those who are still with the bank.
- This could be because they have spent more time with the bank, an those who are still with the bank are new and young.
- There are also a significant amount of outliers with those who are still with the bank, which will need to be looked at.

`Exited` vs `Tenure`

In [34]:
```
# Tenure vs age analysis
plt.figure(figsize=(12, 5))
sns.boxplot(x="Exited", y="Tenure", data=df)  # Use sns.boxplot for a boxplot
```

```
plt.show()
```

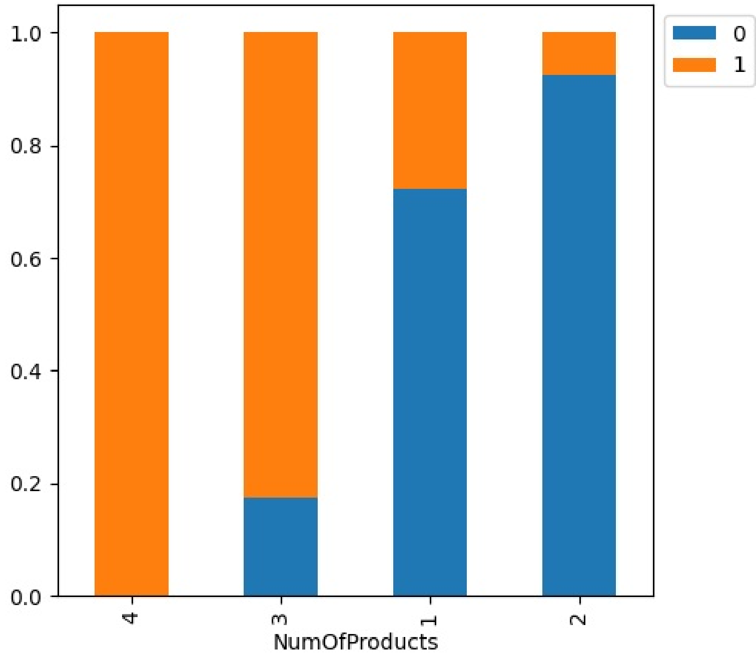


- The customers who are still with the bank has a smaller range of loyality with the bank (3,7).
- Those customers who are not with the bank have a larger range of years with the customer (2-8), this could be because they are much older and experienced.

`Exited` vs `NumOfProducts`

In [35]:
```
# Analysis between num of products and exited
stacked_barplot(df, "NumOfProducts", "Exited")
```

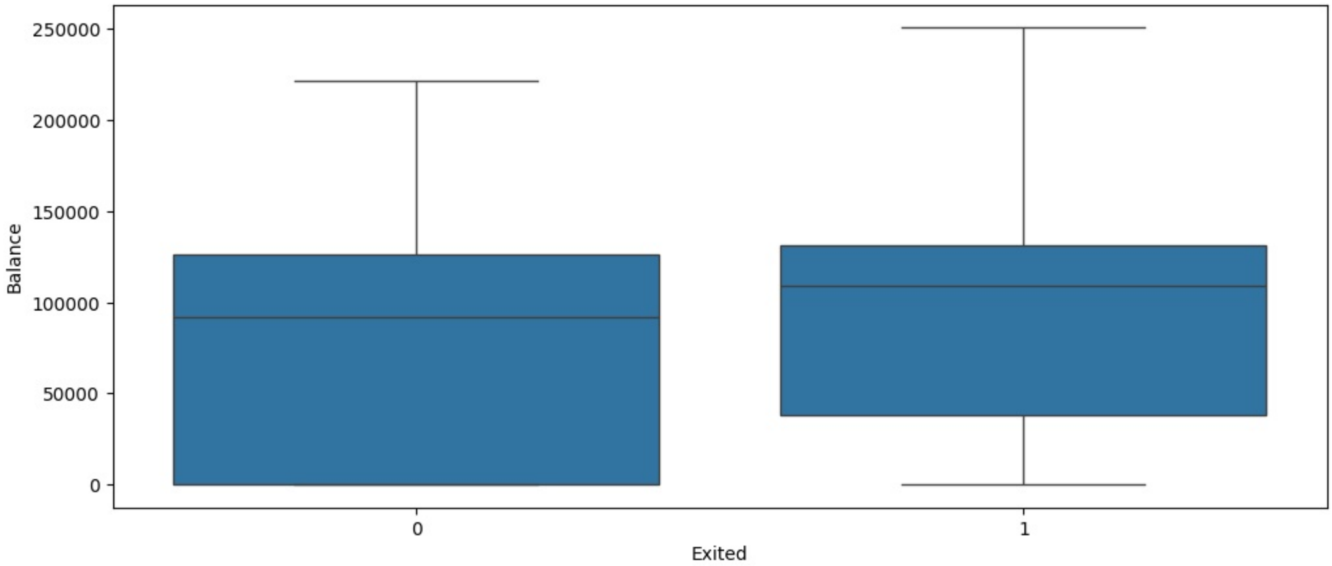
```
Exited            0     1    All
NumOfProducts
All            7963  2037  10000
1              3675  1409   5084
2              4242   348   4590
3                46   220    266
4                 0    60     60
------------------------------------------------------------------------------------------------------------------------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



- Those custoemrs who are not with the bank have a much more amount of products with the bank between 3-4. On the contrary those customers who are still with the bank have a more smaller amount 1-2.

`Exited` vs `Balance`

In [36]:
```
#Exited and balance anlyiss
plt.figure(figsize=(12, 5))
sns.boxplot(x="Exited", y="Balance", data=df)  # Use sns.boxplot for a boxplot
plt.show()
```

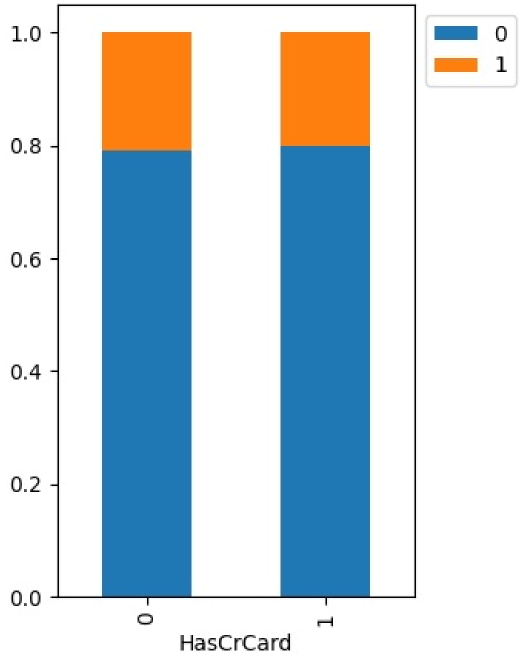


- The customers who are with the bank have many memebrs with a balance of zero, this could be due to a data error that needs to be fixed.
- All customers have around the same balance universly.

`Exited` vs `HasCrCard`

In [37]:
```
# exited and hascrcard
stacked_barplot(df, "HasCrCard", "Exited")
```

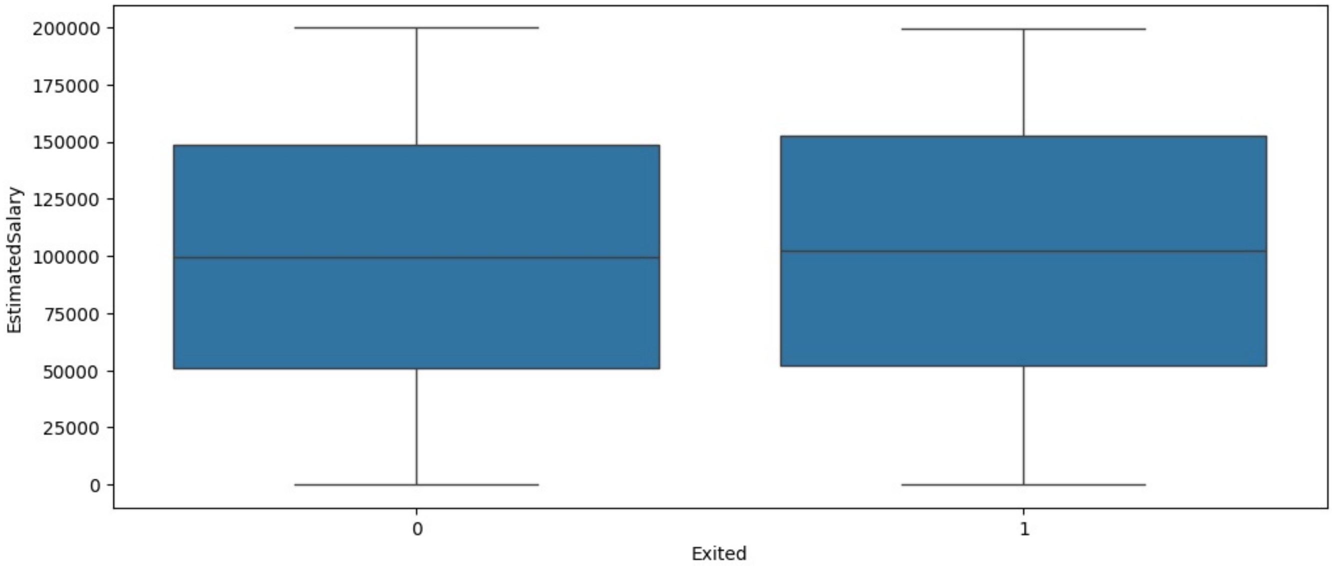
```
Exited        0     1    All
HasCrCard
All        7963  2037  10000
1          5631  1424   7055
0          2332   613   2945
------------------------------------------------------------------------------------------------------------------------
```



- Customers who are with the bank and customers who are not with the bank have teh same ratio of those who have credit cards an not.

`Exited` vs `EstimatedSalary`

In [38]:
```
# Anlysis on exited and EstimatedSalary
plt.figure(figsize=(12, 5))
sns.boxplot(x="Exited", y="EstimatedSalary", data=df)  # Use sns.boxplot for a boxplot
plt.show()
```

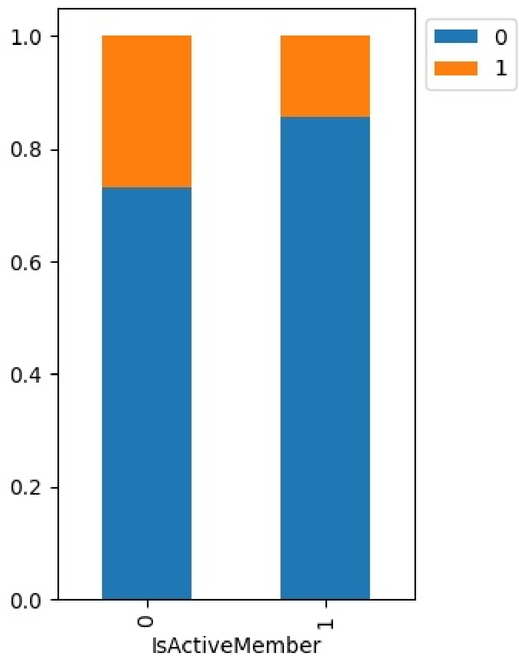


- Both categories of customers who are and are not with the bank have the same range of esitmated salary, as well as the same avreages. Ranging from 50K-150K, and averaging from 100K.

`Exited` vs `isActiveMember`

In [39]:
```
# Exited vs isactive memebr
stacked_barplot(df, "IsActiveMember", "Exited")
```

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



- Customers who are with the bank are more active than those who are not.

Data Preprocessing

Column Bining

Binning the "Tenure" Column

In [40]:
```
bins = [0, 1, 3, 5, 7, 10]
labels = ['<1 year', '1-3 years', '3-5 years', '5-7 years', '7+ years']
df['TenureGroup'] = pd.cut(df['Tenure'], bins=bins, labels=labels)
```

In [41]:
```
# Drop tenure column
```

```python
df.drop("Tenure", axis=1, inplace=True)
```

In [42]:
```python
df
```

Out[42]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Balance	NumOfProducts	HasCrCard	IsActiveMember	E
0	1	15634602	Hargrave	619	France	Female	42	0.00	1	1	1	
1	2	15647311	Hill	608	Spain	Female	41	83807.86	1	0	1	
2	3	15619304	Onio	502	France	Female	42	159660.80	3	1	0	
3	4	15701354	Boni	699	France	Female	39	0.00	2	0	0	
4	5	15737888	Mitchell	850	Spain	Female	43	125510.82	1	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	0.00	2	1	0	
9996	9997	15569892	Johnstone	516	France	Male	35	57369.61	1	1	1	
9997	9998	15584532	Liu	709	France	Female	36	0.00	1	0	1	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	75075.31	2	1	0	
9999	10000	15628319	Walker	792	France	Female	28	130142.79	1	1	0	

10000 rows × 14 columns

Binning the "CreditScore" Column

In [43]:
```python
bins = [300, 579, 669, 739, 799, 850]  # These are common ranges for credit scores
labels = ['Poor', 'Fair', 'Good', 'Very Good', 'Excellent']
df['CreditScore'] = pd.cut(df['CreditScore'], bins=bins, labels=labels)
```

In [44]:
```python
df
```

Out[44]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Balance	NumOfProducts	HasCrCard	IsActiveMember	E
0	1	15634602	Hargrave	Fair	France	Female	42	0.00	1	1	1	
1	2	15647311	Hill	Fair	Spain	Female	41	83807.86	1	0	1	
2	3	15619304	Onio	Poor	France	Female	42	159660.80	3	1	0	
3	4	15701354	Boni	Good	France	Female	39	0.00	2	0	0	
4	5	15737888	Mitchell	Excellent	Spain	Female	43	125510.82	1	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	Very Good	France	Male	39	0.00	2	1	0	
9996	9997	15569892	Johnstone	Poor	France	Male	35	57369.61	1	1	1	
9997	9998	15584532	Liu	Good	France	Female	36	0.00	1	0	1	
9998	9999	15682355	Sabbatini	Very Good	Germany	Male	42	75075.31	2	1	0	
9999	10000	15628319	Walker	Very Good	France	Female	28	130142.79	1	1	0	

10000 rows × 14 columns

Encoding categorical variables

In [45]:
```python
# Encoding the categorical variables using one-hot encoding
df = pd.get_dummies(df, drop_first=True)
```

In [46]:
```python
df.head()
```

Out[46]:

	RowNumber	CustomerId	Age	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Surname_Abbie	...	Crec
0	1	15634602	42	0.00	1	1	1	101348.88	1	False	...	
1	2	15647311	41	83807.86	1	0	1	112542.58	0	False	...	
2	3	15619304	42	159660.80	3	1	0	113931.57	1	False	...	
3	4	15701354	39	0.00	2	0	0	93826.63	0	False	...	
4	5	15737888	43	125510.82	1	1	1	79084.10	0	False	...	

5 rows × 2951 columns

Dummy Variable Creation

In [47]:
```
# Create dummy variables
df = pd.get_dummies(df, drop_first=True)
```

In [48]:
```
df.columns
```

Out[48]:
```
Index(['RowNumber', 'CustomerId', 'Age', 'Balance', 'NumOfProducts',
       'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Exited',
       'Surname_Abbie',
       ...
       'CreditScore_Good', 'CreditScore_Very Good', 'CreditScore_Excellent',
       'Geography_Germany', 'Geography_Spain', 'Gender_Male',
       'TenureGroup_1-3 years', 'TenureGroup_3-5 years',
       'TenureGroup_5-7 years', 'TenureGroup_7+ years'],
      dtype='object', length=2951)
```

In [49]:
```
df.shape
```

Out[49]:
```
(10000, 2951)
```

In [50]:
```
# Make the columns 1s and 0s instead of true and false
df = df.replace({False: 0, True: 1})
```

In [51]:
```
df.head()
```

Out[51]:

	RowNumber	CustomerId	Age	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Surname_Abbie	...	Cred
0	1	15634602	42	0.00	1	1	1	101348.88	1	0	...	
1	2	15647311	41	83807.86	1	0	1	112542.58	0	0	...	
2	3	15619304	42	159660.80	3	1	0	113931.57	1	0	...	
3	4	15701354	39	0.00	2	0	0	93826.63	0	0	...	
4	5	15737888	43	125510.82	1	1	1	79084.10	0	0	...	

5 rows × 2951 columns

Train-validation-test Split

In [52]:
```
# defining the dependent and independent variables
X = df.drop(["Exited", "RowNumber", "CustomerId"], axis=1) # Pass a list of column names as strings.
y = df["Exited"]
```

In [53]:
```
# splitting the data in 80:20 ratio for train and temporary data
x_train, x_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2,random_state=1)
```

In [54]:
```
# splitting the temporary data in 50:50 ratio for validation and test data
x_val,x_test,y_val,y_test = train_test_split(x_temp,y_temp,test_size=0.5,random_state=1)
```

In [55]:
```
print("Number of Rows in Train Data =", x_train.shape[0])
print("Number of Rows in Validation Data =", x_val.shape[0])
print("Number of Rows in Test Data =", x_test.shape[0])
```

```
Number of Rows in Train Data = 8000
Number of Rows in Validation Data = 1000
Number of Rows in Test Data = 1000
```

In [56]:
```
X.columns
```

Out[56]:
```
Index(['Age', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember',
       'EstimatedSalary', 'Surname_Abbie', 'Surname_Abbott',
       'Surname_Abdullah', 'Surname_Abdulov',
       ...
       'CreditScore_Good', 'CreditScore_Very Good', 'CreditScore_Excellent',
       'Geography_Germany', 'Geography_Spain', 'Gender_Male',
       'TenureGroup_1-3 years', 'TenureGroup_3-5 years',
       'TenureGroup_5-7 years', 'TenureGroup_7+ years'],
      dtype='object', length=2948)
```

Data Normalization

In [57]:
```
# List of numerical features to scale (e.g., numerical features)
num_columns = ['Age', 'Balance', 'EstimatedSalary', 'NumOfProducts']
```

In [58]:
```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler to the selected columns in the x_train data
scaler.fit(x_train[num_columns])
```

Out[58]:

```
▾ StandardScaler
StandardScaler()
```

In [59]:

```python
# Transform selected columns in x_train, x_val, and x_test using the fitted scaler

x_train[num_columns] = scaler.transform(x_train[num_columns])

x_val[num_columns] = scaler.transform(x_val[num_columns])

x_test[num_columns] = scaler.transform(x_test[num_columns])
```

In [60]:

```python
x_train.head()
```

Out[60]:

	Age	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Surname_Abbie	Surname_Abbott	Surname_Abdulla
2694	-0.944500	0.588173	0.802257	0	1	0.427394	0	0	
5140	-0.944500	0.469849	0.802257	0	0	-1.025487	0	0	
2568	0.774987	0.858788	-0.911510	1	1	-0.944798	0	0	
3671	1.252622	0.565604	0.802257	0	1	-0.551946	0	0	
7427	-0.562392	0.730395	-0.911510	0	0	1.083383	0	0	

5 rows × 2948 columns

Utility functions

In [61]:

```python
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 [62]:

```python
# function to compute adjusted R-squared
def adj_r2_score(predictors, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = predictors.shape[0]
    k = predictors.shape[1]
    return 1 - ((1 - r2) * (n - 1) / (n - k - 1))


# function to compute MAPE
def mape_score(targets, predictions):
    return np.mean(np.abs(targets - predictions) / targets) * 100


# function to compute different metrics to check performance of a neural network model
def model_performance(model,predictors,target):
    """
    Function to compute different metrics to check regression model performance

    model: regressor
    predictors: independent variables
    target: dependent variable
    """
    # predicting using the independent variables
    pred = model.predict(predictors).reshape(-1)

    r2 = r2_score(target, pred)  # to compute R-squared
    adjr2 = adj_r2_score(predictors, target, pred)  # to compute adjusted R-squared
    rmse = np.sqrt(mean_squared_error(target, pred))  # to compute RMSE
    mae = mean_absolute_error(target, pred)  # to compute MAE
    mape = mape_score(target, pred)  # to compute MAPE

    # creating a dataframe of metrics
    df_perf =  {
            "RMSE": [rmse],
            "MAE": [mae],
            "R-squared": [r2],
            "Adj. R-squared": [adjr2],
            "MAPE": [mape]}
```

```
    return df_perf

columns = ["# hidden layers","# neurons - hidden layer","activation function - hidden layer ","# epochs","batch

results = pd.DataFrame(columns=columns)
```

In [63]:
```
# defining a function to compute different metrics to check performance of a classification model built using s
def model_performance_classification(
    model, predictors, target, threshold=0.5
):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    threshold: threshold for classifying the observation as class 1
    """

    # checking which probabilities are greater than threshold
    pred = model.predict(predictors) > threshold
    # pred_temp = model.predict(predictors) > threshold
    # # rounding off the above values to get classes
    # pred = np.round(pred_temp)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred, average='weighted')  # to compute Recall
    precision = precision_score(target, pred, average='weighted')  # to compute Precision
    f1 = f1_score(target, pred, average='weighted')  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1 Score": f1,},
        index=[0],
    )

    return df_perf
```

Model Building

Model Evaluation Criterion

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

- Recall is the best metric for this business scenario of identifying customers that will churn or not over half a year. Missing out on customers who are about to leave (false negatives) could be very vital, as retaining existing customers is typically less expensive than acquiring new ones. While precision and accuracy are important, the cost of missed churn cases (false negatives) outweighs the cost of incorrectly predicting churn (false positives).

Since the target variable is imbalanced, ues class eights to allow the model to give a sort of importance to the minority classes.

In [64]:
```
# Calculate class weights for imbalanced dataset
cw = (y_train.shape[0]) / np.bincount(y_train)

# Create a dictionary mapping class indices to their respective class weights
cw_dict = {}
for i in range(cw.shape[0]):
    cw_dict[i] = cw[i]

cw_dict
```

Out[64]:
```
{0: 1.2543116964565695, 1: 4.932182490752158}
```

In [65]:
```
# use recall metric to be used for all models
metrics = [tf.keras.metrics.Recall(name="recall")]
```

Neural Network with SGD Optimizer

In [66]:
```
# Clears the current Keras Session, Resets all layers and models previously created, and frees up memory and re
tf.keras.backend.clear_session()
```

In [67]:
```
#Initializing the Neural Network
model = Sequential()
model.add(Dense(1,input_dim=x_train.shape[1]))
```

In [68]:
```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	2,949

Total params: 2,949 (11.52 KB)

Trainable params: 2,949 (11.52 KB)

Non-trainable params: 0 (0.00 B)

In [69]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
import tensorflow.keras as keras

optimizer = keras.optimizers.SGD()
model.compile(loss="mean_squared_error", optimizer=optimizer, metrics=metrics,run_eagerly=True)
```

In [70]:

```
# Initiate the epochs and batch_size
epochs = 25
batch_size = 64
```

In [71]:

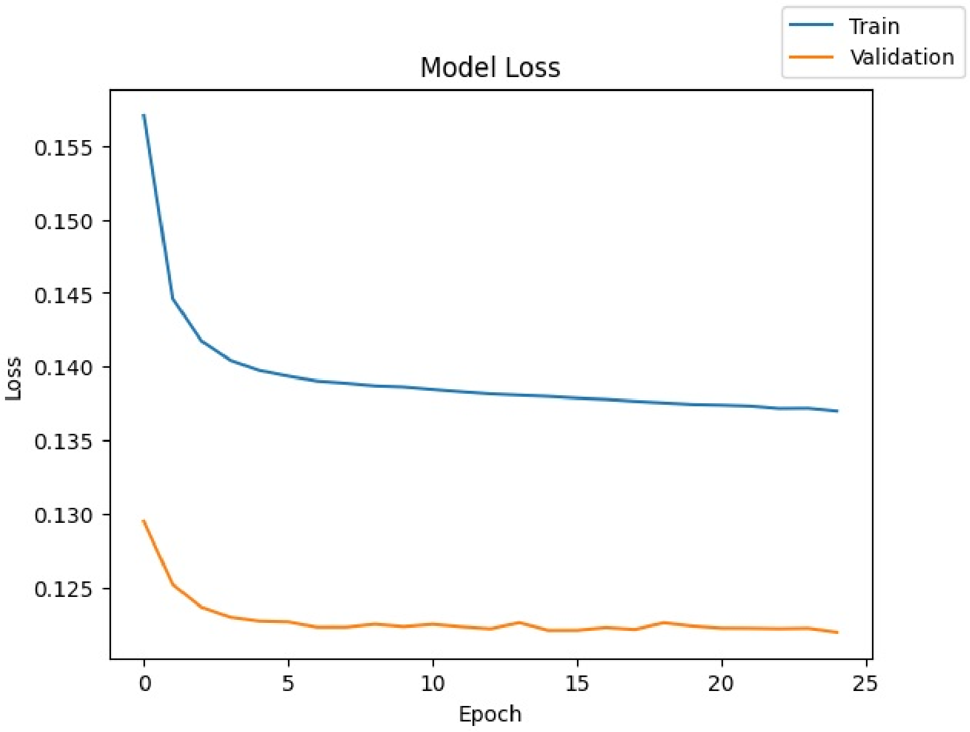
```
start = time.time()
history = model.fit(x_train, y_train, validation_data=(x_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()
```

```
Epoch 1/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 38ms/step - loss: 0.1678 - recall: 0.0016 - val_loss: 0.1295 - val_recall: 0.01
68
Epoch 2/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.1442 - recall: 0.0192 - val_loss: 0.1252 - val_recall: 0.03
91
Epoch 3/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - loss: 0.1419 - recall: 0.0451 - val_loss: 0.1237 - val_recall: 0.05
03
Epoch 4/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - loss: 0.1399 - recall: 0.0641 - val_loss: 0.1230 - val_recall: 0.09
50
Epoch 5/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 33ms/step - loss: 0.1401 - recall: 0.0833 - val_loss: 0.1227 - val_recall: 0.10
61
Epoch 6/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.1329 - recall: 0.0768 - val_loss: 0.1227 - val_recall: 0.13
97
Epoch 7/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.1409 - recall: 0.1065 - val_loss: 0.1223 - val_recall: 0.10
61
Epoch 8/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 59ms/step - loss: 0.1411 - recall: 0.0951 - val_loss: 0.1223 - val_recall: 0.10
61
Epoch 9/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 59ms/step - loss: 0.1423 - recall: 0.1108 - val_loss: 0.1225 - val_recall: 0.11
73
Epoch 10/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 71ms/step - loss: 0.1398 - recall: 0.1049 - val_loss: 0.1223 - val_recall: 0.11
73
Epoch 11/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 27ms/step - loss: 0.1406 - recall: 0.1182 - val_loss: 0.1225 - val_recall: 0.11
17
Epoch 12/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 13s 94ms/step - loss: 0.1366 - recall: 0.1015 - val_loss: 0.1223 - val_recall: 0.1
285
Epoch 13/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - loss: 0.1380 - recall: 0.1131 - val_loss: 0.1222 - val_recall: 0.12
85
Epoch 14/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 8s 52ms/step - loss: 0.1374 - recall: 0.1200 - val_loss: 0.1226 - val_recall: 0.12
85
Epoch 15/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 11s 58ms/step - loss: 0.1380 - recall: 0.1104 - val_loss: 0.1221 - val_recall: 0.1
173
Epoch 16/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 25ms/step - loss: 0.1376 - recall: 0.1010 - val_loss: 0.1221 - val_recall: 0.12
85
Epoch 17/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.1413 - recall: 0.1027 - val_loss: 0.1223 - val_recall: 0.15
64
Epoch 18/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - loss: 0.1386 - recall: 0.1266 - val_loss: 0.1221 - val_recall: 0.13
41
Epoch 19/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 59ms/step - loss: 0.1370 - recall: 0.1296 - val_loss: 0.1226 - val_recall: 0.13
97
Epoch 20/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - loss: 0.1387 - recall: 0.1277 - val_loss: 0.1224 - val_recall: 0.13
41
Epoch 21/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 31ms/step - loss: 0.1398 - recall: 0.1151 - val_loss: 0.1222 - val_recall: 0.13
41
Epoch 22/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 46ms/step - loss: 0.1383 - recall: 0.1120 - val_loss: 0.1222 - val_recall: 0.13
97
Epoch 23/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 12s 64ms/step - loss: 0.1391 - recall: 0.1384 - val_loss: 0.1222 - val_recall: 0.1
117
Epoch 24/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - loss: 0.1358 - recall: 0.1162 - val_loss: 0.1222 - val_recall: 0.13
97
Epoch 25/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 11s 50ms/step - loss: 0.1365 - recall: 0.1160 - val_loss: 0.1220 - val_recall: 0.1
229
```

In [72]:
```
print("Time Taken (Seconds): ",end-start)
```

```
Time Taken (Seconds):  160.58779788017273
```

In [73]:
```
plot(history,'loss')
```



In [74]:
```
model_0_train_perf = model_performance_classification(model, x_train, y_train)
model_0_train_perf
```

```
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

Out[74]:

	Accuracy	Recall	Precision	F1 Score
0	0.807625	0.807625	0.779839	0.748994

In [75]:
```
model_0_valid_perf = model_performance_classification(model, x_val, y_val)
model_0_valid_perf
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

Out[75]:

	Accuracy	Recall	Precision	F1 Score
0	0.834	0.834	0.815012	0.782367

In [76]:
```
# Function for the chart
results.loc["Model 0"]=['-','-','-',epochs,batch_size,'SGD',(end-start),history.history["loss"][-1],history.his
```

In [77]:
```
# Displaying the results
results
```

Out[77]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	time(secs)	Train_loss	Valid_loss
Model 0	-	-	-	25	64	SGD	160.587798	0.136982	0.121964

- The train and validation recall scores being 0.81 and 0.83 indicate a relatively consistent performance, although the score isn't the best to have.
- The time taken is significantly longer due to the model parents being updated more often.
- The loss scores are relatively low as well, which is a good statistic.

Model Performance Improvement

Neural Network with Adam Optimizer

In [78]:
```
# clears the current Keras session, resetting all layers and models previously created, freeing up memory and r
tf.keras.backend.clear_session()
```

In [79]:
```
#Initializing the neural network
model = Sequential()
model.add(Dense(14,activation="relu",input_dim=x_train.shape[1]))
model.add(Dense(7,activation="relu"))
model.add(Dense(1,activation="sigmoid"))
```

In [80]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 14)	41,286
dense_1 (Dense)	(None, 7)	105
dense_2 (Dense)	(None, 1)	8

Total params: 41,399 (161.71 KB)

Trainable params: 41,399 (161.71 KB)

Non-trainable params: 0 (0.00 B)

In [81]:

```
# Import the Adam optimizer from Keras
from tensorflow.keras.optimizers import Adam

optimizer = tf.keras.optimizers.Adam()    # defining Adam as the optimizer to be used
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=metrics)
```

In [82]:

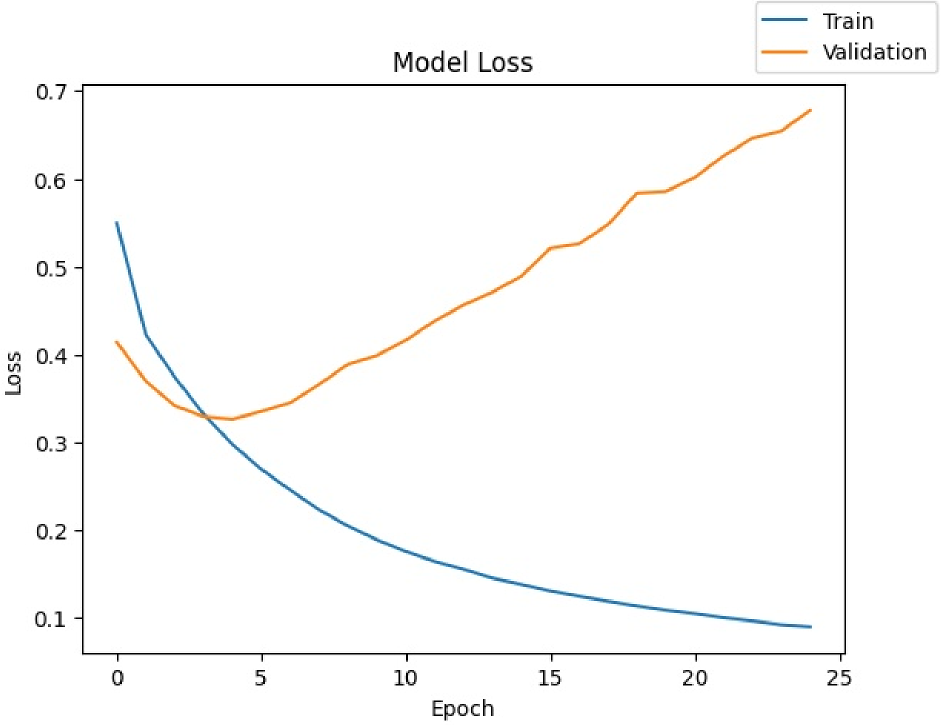
```
start = time.time()
history = model.fit(x_train, y_train, validation_data=(x_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()
```

```
Epoch 1/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - loss: 0.6196 - recall: 0.0381 - val_loss: 0.4141 - val_recall: 0.000
0e+00
Epoch 2/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.4188 - recall: 0.0571 - val_loss: 0.3698 - val_recall: 0.257
0
Epoch 3/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.3845 - recall: 0.3077 - val_loss: 0.3419 - val_recall: 0.318
4
Epoch 4/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.3377 - recall: 0.4204 - val_loss: 0.3295 - val_recall: 0.379
9
Epoch 5/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.3011 - recall: 0.5048 - val_loss: 0.3265 - val_recall: 0.435
8
Epoch 6/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.2736 - recall: 0.5936 - val_loss: 0.3354 - val_recall: 0.385
5
Epoch 7/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.2459 - recall: 0.6220 - val_loss: 0.3452 - val_recall: 0.407
8
Epoch 8/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2228 - recall: 0.6932 - val_loss: 0.3660 - val_recall: 0.441
3
Epoch 9/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2061 - recall: 0.7247 - val_loss: 0.3890 - val_recall: 0.525
1
Epoch 10/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1894 - recall: 0.7642 - val_loss: 0.3991 - val_recall: 0.463
7
Epoch 11/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1734 - recall: 0.7753 - val_loss: 0.4164 - val_recall: 0.407
8
Epoch 12/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1632 - recall: 0.7989 - val_loss: 0.4382 - val_recall: 0.413
4
Epoch 13/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1563 - recall: 0.8130 - val_loss: 0.4564 - val_recall: 0.407
8
Epoch 14/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1403 - recall: 0.8240 - val_loss: 0.4712 - val_recall: 0.413
4
Epoch 15/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1369 - recall: 0.8290 - val_loss: 0.4892 - val_recall: 0.402
2
Epoch 16/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1278 - recall: 0.8434 - val_loss: 0.5210 - val_recall: 0.379
9
Epoch 17/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1248 - recall: 0.8571 - val_loss: 0.5261 - val_recall: 0.424
6
Epoch 18/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1196 - recall: 0.8540 - val_loss: 0.5478 - val_recall: 0.435
8
Epoch 19/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1073 - recall: 0.8864 - val_loss: 0.5839 - val_recall: 0.357
5
Epoch 20/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.1031 - recall: 0.8728 - val_loss: 0.5857 - val_recall: 0.486
0
Epoch 21/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1051 - recall: 0.8902 - val_loss: 0.6015 - val_recall: 0.391
1
Epoch 22/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0971 - recall: 0.8925 - val_loss: 0.6261 - val_recall: 0.402
2
Epoch 23/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0924 - recall: 0.8973 - val_loss: 0.6463 - val_recall: 0.385
5
Epoch 24/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0879 - recall: 0.9180 - val_loss: 0.6545 - val_recall: 0.385
5
Epoch 25/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0858 - recall: 0.9067 - val_loss: 0.6781 - val_recall: 0.374
3
```

In [83]:
```
print("Time taken in seconds ",end-start)
```

```
Time taken in seconds  25.44655203819275
```

In [84]:
```
plot(history,'loss')
```



In [85]:
```
results.loc["Model 1"]=[2,[14,7],['relu','relu'],epochs,batch_size,'Adam',(end-start),history.history["loss"][-
```

In [86]:
```
# Displaying the results
results
```

Out[86]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	time(secs)	Train_loss	Valid_loss
Model 0	-	-	-	25	64	SGD	160.587798	0.136982	0.121964
Model 1	2	[14, 7]	[relu, relu]	25	64	Adam	25.446552	0.090293	0.678093

In [87]:
```
model_1_train_perf = model_performance_classification(model, x_train, y_train)
model_1_train_perf
```

250/250 ━━━━━━━━━━━━━━━━━━━━ **1s** 2ms/step

Out[87]:

	Accuracy	Recall	Precision	F1 Score
0	0.972875	0.972875	0.972731	0.972518

In [88]:
```
model_1_valid_perf = model_performance_classification(model, x_val, y_val)
model_1_valid_perf
```

32/32 ━━━━━━━━━━━━━━━━━━━━ **0s** 3ms/step

Out[88]:

	Accuracy	Recall	Precision	F1 Score
0	0.821	0.821	0.80432	0.81052

- From Model 0 to Model 1, the time taken to run through the model has drastically decreased by over 90 seconds.
- However, the train and validation loss scores have increased, tripling from the validation especially.
- The training recall score in Model 1 has increased, however the validation score only slightly improved. Thus showing a sign of overfitting where the gap between the train and validation recall score is significant.

Neural Network with Adam Optimizer and Dropout

In [89]:
```
# clears the current Keras session, resetting all layers and models previously created, freeing up memory and r
tf.keras.backend.clear_session()
```

In [90]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Define the model
model = Sequential()

# Input layer
model.add(Dense(64, activation='relu', input_shape=(2948,)))  # Adjust input_shape to match your feature count
```

```
# Hidden layer 1 with Dropout
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # 50% dropout rate

# Hidden layer 2 with Dropout
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))  # 50% dropout rate

# Output layer for binary classification
model.add(Dense(1, activation='sigmoid'))
```

In [91]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	188,736
dense_1 (Dense)	(None, 128)	8,320
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Total params: 205,377 (802.25 KB)

Trainable params: 205,377 (802.25 KB)

Non-trainable params: 0 (0.00 B)

In [92]:
```
# Compile the model with Adam optimizer and binary cross-entropy loss
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=[tf.keras.metrics.Recall(name="recall")])
```

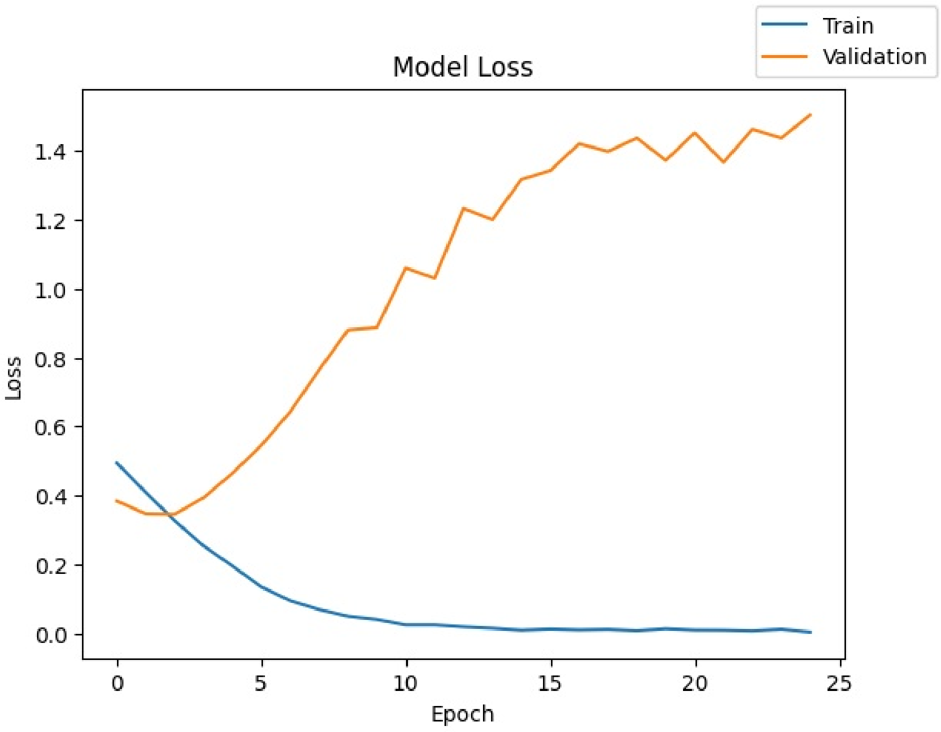
In [93]:
```
start = time.time()
history = model.fit(x_train, y_train, validation_data=(x_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()
```

```
Epoch 1/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.5549 - recall: 0.0812 - val_loss: 0.3841 - val_recall: 0.000
0e+00
Epoch 2/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.4070 - recall: 0.0879 - val_loss: 0.3470 - val_recall: 0.385
5
Epoch 3/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.3346 - recall: 0.4751 - val_loss: 0.3457 - val_recall: 0.547
5
Epoch 4/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 0.2494 - recall: 0.6810 - val_loss: 0.3937 - val_recall: 0.43
58
Epoch 5/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.1831 - recall: 0.8085 - val_loss: 0.4646 - val_recall: 0.45
81
Epoch 6/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.1339 - recall: 0.8560 - val_loss: 0.5453 - val_recall: 0.441
3
Epoch 7/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0802 - recall: 0.9152 - val_loss: 0.6431 - val_recall: 0.413
4
Epoch 8/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0661 - recall: 0.9283 - val_loss: 0.7643 - val_recall: 0.413
4
Epoch 9/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0516 - recall: 0.9454 - val_loss: 0.8802 - val_recall: 0.396
6
Epoch 10/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0420 - recall: 0.9558 - val_loss: 0.8875 - val_recall: 0.34
08
Epoch 11/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.0203 - recall: 0.9742 - val_loss: 1.0600 - val_recall: 0.36
87
Epoch 12/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0239 - recall: 0.9730 - val_loss: 1.0307 - val_recall: 0.446
9
Epoch 13/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0165 - recall: 0.9853 - val_loss: 1.2330 - val_recall: 0.50
84
Epoch 14/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0211 - recall: 0.9871 - val_loss: 1.2006 - val_recall: 0.39
11
Epoch 15/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0065 - recall: 0.9915 - val_loss: 1.3172 - val_recall: 0.368
7
Epoch 16/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0133 - recall: 0.9907 - val_loss: 1.3429 - val_recall: 0.413
4
Epoch 17/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0085 - recall: 0.9949 - val_loss: 1.4209 - val_recall: 0.435
8
Epoch 18/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0138 - recall: 0.9884 - val_loss: 1.3985 - val_recall: 0.424
6
Epoch 19/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0052 - recall: 0.9946 - val_loss: 1.4374 - val_recall: 0.385
5
Epoch 20/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0130 - recall: 0.9899 - val_loss: 1.3727 - val_recall: 0.407
8
Epoch 21/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0072 - recall: 0.9926 - val_loss: 1.4525 - val_recall: 0.324
0
Epoch 22/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0072 - recall: 0.9932 - val_loss: 1.3668 - val_recall: 0.41
90
Epoch 23/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0048 - recall: 0.9962 - val_loss: 1.4630 - val_recall: 0.39
66
Epoch 24/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0151 - recall: 0.9875 - val_loss: 1.4375 - val_recall: 0.41
90
Epoch 25/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0047 - recall: 0.9940 - val_loss: 1.5045 - val_recall: 0.385
5
```

In [94]:
```
print("Time taken in seconds ",end-start)
```

```
Time taken in seconds  40.763280630111694
```

In [95]:
```
plot(history,'loss')
```



In [96]:
```
# Function for the chart

results.loc["Model 2"]=[3,[64,128,64],['relu','relu','relu'],epochs,batch_size,'Adam',(end-start),history.histo
```

In [97]:
```
# Displaying the results
results
```

Out[97]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	time(secs)	Train_loss	Valid_loss
Model 0	-	-	-	25	64	SGD	160.587798	0.136982	0.121964
Model 1	2	[14, 7]	[relu, relu]	25	64	Adam	25.446552	0.090293	0.678093
Model 2	3	[64, 128, 64]	[relu, relu, relu]	25	64	Adam	40.763281	0.003548	1.504508

In [98]:
```
model_2_train_perf = model_performance_classification(model, x_train, y_train)
model_2_train_perf
```

```
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

Out[98]:

	Accuracy	Recall	Precision	F1 Score
0	1.0	1.0	1.0	1.0

In [99]:
```
model_2_valid_perf = model_performance_classification(model, x_val, y_val)
model_2_valid_perf
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

Out[99]:

	Accuracy	Recall	Precision	F1 Score
0	0.808	0.808	0.796422	0.801464

- In Model 2, where there are now 3 hidden layers, the time taken to run through all 25 epochs has increased slightly form the previous model.
- The recall score in model 2 is still overfitting, even more than in model 1, where the gap is much larger.
- The training loss score in this model has been the best out of all the models compiled so far. On the contrary for the validation loss score, it has been the highest/worst of all models.

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

In [100…
```
tf.keras.backend.clear_session()
```

In [101…
```
# Import the SMOTE class from imblearn.over_sampling
from imblearn.over_sampling import SMOTE

# Apply SMOTE to balance the training data
smote = SMOTE(random_state=42)
```

```
x_train_smote, y_train_smote = smote.fit_resample(x_train, y_train)

# Further split the balanced training set into training and validation sets
x_train_smote, x_val, y_train_smote, y_val = train_test_split(x_train_smote, y_train_smote, test_size=0.2, rand
```

In [102…

```
# Standardize the data
scaler = StandardScaler()
x_train_smote = scaler.fit_transform(x_train_smote)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)
```

In [103…

```
# Define the model
model = Sequential()

# Input layer
model.add(Dense(64, activation='relu', input_shape=(x_train_smote.shape[1],)))  # Number of features

# Hidden layers
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))

# Output layer
model.add(Dense(1, activation='sigmoid'))
```

In [104…

```
# Compile the model with SGD optimizer
model.compile(optimizer=SGD(learning_rate=0.01),
              loss='binary_crossentropy',
              metrics=[tf.keras.metrics.Recall(name='recall')])
```

In [105…

```
start = time.time()
history = model.fit(x_train_smote, y_train_smote, validation_data=(x_val, y_val), epochs=epochs, batch_size=bat
end=time.time()
```

```
Epoch 1/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.6099 - recall: 0.3944 - val_loss: 0.5860 - val_recall: 0.78
73
Epoch 2/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.5392 - recall: 0.7403 - val_loss: 0.5551 - val_recall: 0.788
9
Epoch 3/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.4819 - recall: 0.7433 - val_loss: 0.5373 - val_recall: 0.802
2
Epoch 4/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.4229 - recall: 0.7586 - val_loss: 0.5273 - val_recall: 0.805
3
Epoch 5/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.3705 - recall: 0.7773 - val_loss: 0.5273 - val_recall: 0.809
3
Epoch 6/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.3175 - recall: 0.7957 - val_loss: 0.5390 - val_recall: 0.812
4
Epoch 7/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.2720 - recall: 0.8173 - val_loss: 0.5629 - val_recall: 0.817
1
Epoch 8/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.2360 - recall: 0.8508 - val_loss: 0.5896 - val_recall: 0.814
8
Epoch 9/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2159 - recall: 0.8524 - val_loss: 0.6186 - val_recall: 0.821
0
Epoch 10/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.2108 - recall: 0.8598 - val_loss: 0.6475 - val_recall: 0.82
10
Epoch 11/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1923 - recall: 0.8726 - val_loss: 0.6722 - val_recall: 0.83
05
Epoch 12/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1834 - recall: 0.8829 - val_loss: 0.6917 - val_recall: 0.82
81
Epoch 13/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.1816 - recall: 0.8830 - val_loss: 0.7121 - val_recall: 0.833
6
Epoch 14/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.1859 - recall: 0.8769 - val_loss: 0.7277 - val_recall: 0.836
7
Epoch 15/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1765 - recall: 0.8805 - val_loss: 0.7454 - val_recall: 0.835
2
Epoch 16/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1701 - recall: 0.8893 - val_loss: 0.7620 - val_recall: 0.843
0
Epoch 17/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1752 - recall: 0.8929 - val_loss: 0.7753 - val_recall: 0.840
7
Epoch 18/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1647 - recall: 0.8959 - val_loss: 0.7829 - val_recall: 0.842
2
Epoch 19/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1619 - recall: 0.9000 - val_loss: 0.7944 - val_recall: 0.839
9
Epoch 20/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1589 - recall: 0.8960 - val_loss: 0.8038 - val_recall: 0.837
5
Epoch 21/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1553 - recall: 0.9001 - val_loss: 0.8179 - val_recall: 0.849
3
Epoch 22/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1569 - recall: 0.9015 - val_loss: 0.8196 - val_recall: 0.844
6
Epoch 23/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.1585 - recall: 0.8983 - val_loss: 0.8329 - val_recall: 0.843
0
Epoch 24/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1508 - recall: 0.9091 - val_loss: 0.8390 - val_recall: 0.836
7
Epoch 25/25
160/160 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1550 - recall: 0.9006 - val_loss: 0.8505 - val_recall: 0.839
1
```

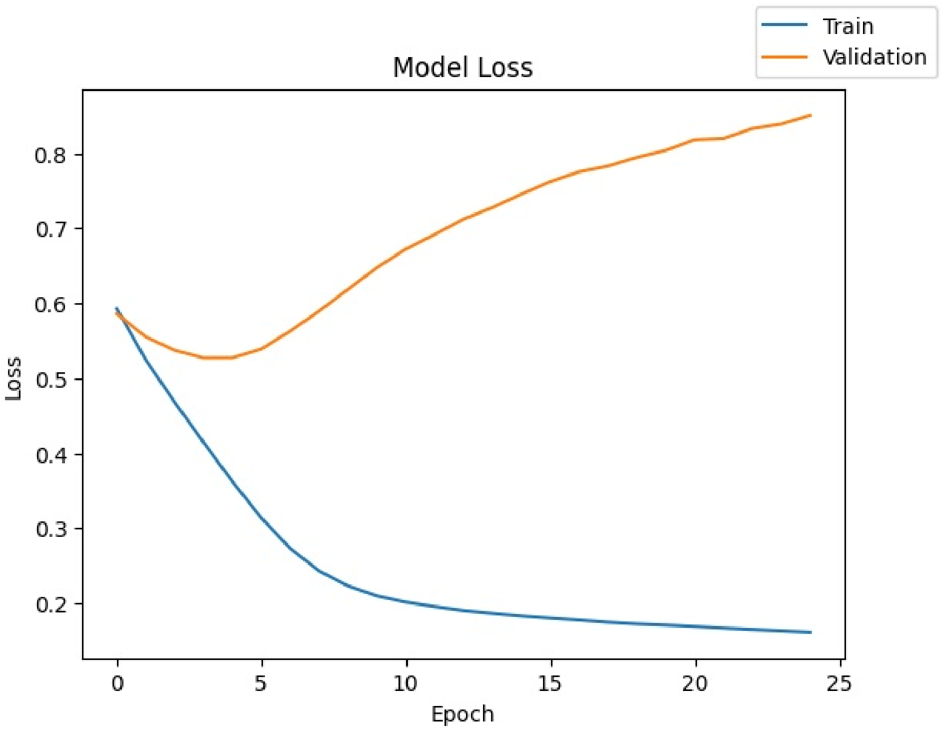
In [106…

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

```
Time taken in seconds  37.55686664581299
```

In [107…

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



In [108…

```
# Function for the chart

results.loc["Model 3"]=[3,[64,128,64],['relu','relu','relu'],epochs,batch_size,'SGD + SMOTE',(end-start),histor
```

In [109…

```
# Display the chart
results
```

Out[109]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	time(secs)	Train_loss	Valid_loss
Model 0	-	-	-	25	64	SGD	160.587798	0.136982	0.121964
Model 1	2	[14, 7]	[relu, relu]	25	64	Adam	25.446552	0.090293	0.678093
Model 2	3	[64, 128, 64]	[relu, relu, relu]	25	64	Adam	40.763281	0.003548	1.504508
Model 3	3	[64, 128, 64]	[relu, relu, relu]	25	64	SGD + SMOTE	37.556867	0.161548	0.850490

In [110…

```
model_3_train_perf = model_performance_classification(model, x_train_smote, y_train_smote)
model_3_train_perf
```

```
319/319 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

Out[110]:

	Accuracy	Recall	Precision	F1 Score
0	0.940612	0.940612	0.942162	0.94056

In [111…

```
model_3_valid_perf = model_performance_classification(model, x_val, y_val)
model_3_valid_perf
```

```
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

Out[111]:

	Accuracy	Recall	Precision	F1 Score
0	0.766066	0.766066	0.771958	0.76483

- Model 3, where the optimizers are now SGD & SMOTE, has run through the epochs slightly longer than the previous model.
- The training and validation recall scores still show overfitting as the training recall score is 0.94 and the validation is 0.77.
- The loss scores are worse in terms of the training data but have gotten much better from the validation compared to the previous model 2.

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

In [112…

```
tf.keras.backend.clear_session()
```

In [113…

```
# Split data into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE to balance the training data
```

```
smote = SMOTE(random_state=42)
x_train_smote, y_train_smote = smote.fit_resample(x_train, y_train)
```

In [114…

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Standardize the data
scaler = StandardScaler()
x_train_smote = scaler.fit_transform(x_train_smote)
x_val = scaler.transform(x_val)

# Define the model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(x_train_smote.shape[1],)))  # Input layer
model.add(Dense(128, activation='relu'))  # Hidden layer 1
model.add(Dense(64, activation='relu'))  # Hidden layer 2
model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification

# Compile the model with Adam optimizer and recall as a metric
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=[tf.keras.metrics.Recall(name='recall')])
```

In [115…

```
start = time.time()
history = model.fit(x_train_smote, y_train_smote, validation_data=(x_val, y_val), epochs=epochs, batch_size=bat
end=time.time()
```

```
Epoch 1/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 0.4553 - recall: 0.6847 - val_loss: 0.9000 - val_recall: 0.178
1
Epoch 2/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.2597 - recall: 0.7762 - val_loss: 0.9675 - val_recall: 0.29
26
Epoch 3/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.2220 - recall: 0.8396 - val_loss: 0.9845 - val_recall: 0.31
04
Epoch 4/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.1995 - recall: 0.8579 - val_loss: 1.0382 - val_recall: 0.282
4
Epoch 5/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.1760 - recall: 0.8706 - val_loss: 1.0837 - val_recall: 0.343
5
Epoch 6/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1645 - recall: 0.8827 - val_loss: 1.1163 - val_recall: 0.318
1
Epoch 7/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1497 - recall: 0.8898 - val_loss: 1.3353 - val_recall: 0.384
2
Epoch 8/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1392 - recall: 0.8962 - val_loss: 1.2558 - val_recall: 0.346
1
Epoch 9/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1303 - recall: 0.8982 - val_loss: 1.5180 - val_recall: 0.409
7
Epoch 10/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1145 - recall: 0.9139 - val_loss: 1.3861 - val_recall: 0.363
9
Epoch 11/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0955 - recall: 0.9312 - val_loss: 1.4466 - val_recall: 0.412
2
Epoch 12/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.0834 - recall: 0.9450 - val_loss: 1.7693 - val_recall: 0.40
46
Epoch 13/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.0671 - recall: 0.9595 - val_loss: 1.9841 - val_recall: 0.427
5
Epoch 14/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0539 - recall: 0.9695 - val_loss: 2.0981 - val_recall: 0.424
9
Epoch 15/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0390 - recall: 0.9819 - val_loss: 2.3223 - val_recall: 0.419
8
Epoch 16/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0307 - recall: 0.9855 - val_loss: 2.4590 - val_recall: 0.432
6
Epoch 17/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0208 - recall: 0.9906 - val_loss: 2.6391 - val_recall: 0.452
9
Epoch 18/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0166 - recall: 0.9938 - val_loss: 2.9496 - val_recall: 0.442
7
Epoch 19/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0122 - recall: 0.9950 - val_loss: 3.1670 - val_recall: 0.473
3
Epoch 20/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0150 - recall: 0.9963 - val_loss: 3.2079 - val_recall: 0.435
1
Epoch 21/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0095 - recall: 0.9952 - val_loss: 3.3593 - val_recall: 0.440
2
Epoch 22/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0112 - recall: 0.9965 - val_loss: 3.0912 - val_recall: 0.455
5
Epoch 23/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0056 - recall: 0.9981 - val_loss: 3.2425 - val_recall: 0.458
0
Epoch 24/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0054 - recall: 0.9985 - val_loss: 3.1998 - val_recall: 0.430
0
Epoch 25/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0045 - recall: 0.9979 - val_loss: 3.6766 - val_recall: 0.508
9
```

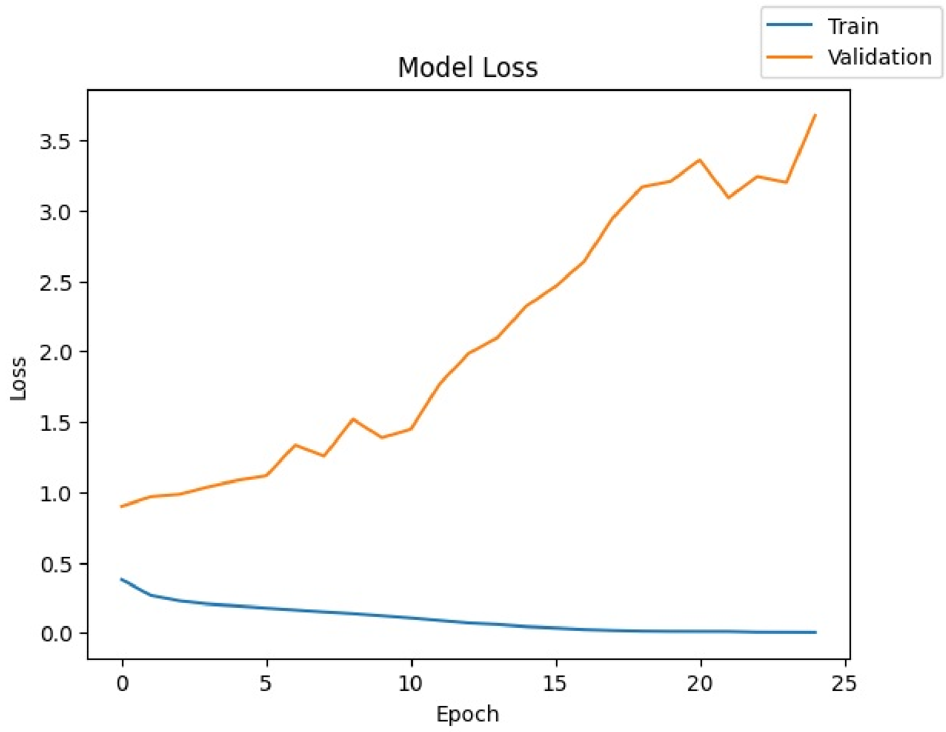
In [116…

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

```
Time taken in seconds  40.7591872215271
```

In [117…

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



In [118…
```
# Function for the chart

results.loc["Model 4"]=[3,[64,128,64],['relu','relu','relu'],epochs,batch_size,'Adam + SMOTE',(end-start),histo
```

In [119…
```
# Displaying the results
results
```

Out[119]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	time(secs)	Train_loss	Valid_loss
Model 0	-	-	-	25	64	SGD	160.587798	0.136982	0.121964
Model 1	2	[14, 7]	[relu, relu]	25	64	Adam	25.446552	0.090293	0.678093
Model 2	3	[64, 128, 64]	[relu, relu, relu]	25	64	Adam	40.763281	0.003548	1.504508
Model 3	3	[64, 128, 64]	[relu, relu, relu]	25	64	SGD + SMOTE	37.556867	0.161548	0.850490
Model 4	3	[64, 128, 64]	[relu, relu, relu]	25	64	Adam + SMOTE	40.759187	0.005454	3.676630

In [120…
```
model_4_train_perf = model_performance_classification(model, x_train_smote, y_train_smote)
model_4_train_perf
```

398/398 ━━━━━━━━━━━━━━━━━━━━ **1s** 2ms/step

Out[120]:

	Accuracy	Recall	Precision	F1 Score
0	0.998584	0.998584	0.998585	0.998584

In [121…
```
model_4_valid_perf = model_performance_classification(model, x_val, y_val)
model_4_valid_perf
```

63/63 ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step

Out[121]:

	Accuracy	Recall	Precision	F1 Score
0	0.662	0.662	0.743291	0.690787

- In Model 4, with the optimizers being Adam & SMOTE this time, the time taken has increased significantly since Model 0, by approximately 20 seconds compared to Model 3.
- The recall scores in this model are still overfitting, where the training score is 0.99 and the validation score is 0.72.
- The loss training score resulted as the best out of all models, although the same cannot be said for the validation loss score, it is a drastically high amount (3.06), which is the most of all models.

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

In [122…
```
tf.keras.backend.clear_session()
```

In [123…
```
from imblearn.over_sampling import SMOTE
```

In [123…
```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Applying SMOTE to balance the data
smote = SMOTE()
x_smote, y_smote = smote.fit_resample(x_train, y_train)

# Split the SMOTE balanced data into training and validation sets
x_train_smote, x_val, y_train_smote, y_val = train_test_split(x_smote, y_smote, test_size=0.2, random_state=42)
```

In [124…
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Initialize the neural network
model = Sequential()

# Input layer
model.add(Dense(64, activation='relu', input_dim=x_train_smote.shape[1]))

# Hidden layer 1 with Dropout
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # 50% dropout rate

# Hidden layer 2 with Dropout
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))  # 50% dropout rate

# Output layer for binary classification
model.add(Dense(1, activation='sigmoid'))

# Compile the model with Adam optimizer and Recall metric
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['Recall'])
```

In [125…
```
start = time.time()
history = model.fit(x_train_smote, y_train_smote, validation_data=(x_val, y_val), epochs=epochs, batch_size=bat
end=time.time()
```

```
Epoch 1/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - Recall: 0.5545 - loss: 342.2238 - val_Recall: 0.4298 - val_loss: 1.
2335
Epoch 2/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - Recall: 0.5327 - loss: 8.9537 - val_Recall: 0.7008 - val_loss: 0.697
0
Epoch 3/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - Recall: 0.4163 - loss: 3.0170 - val_Recall: 0.0105 - val_loss: 0.696
1
Epoch 4/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - Recall: 0.0148 - loss: 0.7071 - val_Recall: 0.0129 - val_loss: 0.694
2
Epoch 5/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 0.0024 - loss: 0.6933 - val_Recall: 0.0000e+00 - val_loss: 0
.6929
Epoch 6/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - Recall: 2.7224e-05 - loss: 0.6933 - val_Recall: 0.0000e+00 - val_los
s: 0.6931
Epoch 7/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 0.1783 - loss: 0.6931 - val_Recall: 0.9984 - val_loss: 0.693
1
Epoch 8/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 0.9329 - loss: 0.6931 - val_Recall: 0.9976 - val_loss: 0.693
1
Epoch 9/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - Recall: 0.9983 - loss: 0.6932 - val_Recall: 0.9984 - val_loss: 0.693
1
Epoch 10/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - Recall: 0.9990 - loss: 0.6931 - val_Recall: 0.9984 - val_loss: 0.69
30
Epoch 11/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - Recall: 0.9979 - loss: 0.6931 - val_Recall: 0.9992 - val_loss: 0.69
29
Epoch 12/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - Recall: 0.9977 - loss: 0.6931 - val_Recall: 0.9992 - val_loss: 0.692
6
Epoch 13/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - Recall: 0.9972 - loss: 0.6949 - val_Recall: 1.0000 - val_loss: 0.692
9
Epoch 14/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 0.9884 - loss: 0.6972 - val_Recall: 1.0000 - val_loss: 0.692
6
Epoch 15/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 1.0000 - loss: 0.6930 - val_Recall: 1.0000 - val_loss: 0.692
5
Epoch 16/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 0.9745 - loss: 0.6925 - val_Recall: 1.0000 - val_loss: 0.692
4
Epoch 17/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - Recall: 0.9998 - loss: 0.6926 - val_Recall: 1.0000 - val_loss: 0.692
3
Epoch 18/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 0.9993 - loss: 0.6928 - val_Recall: 1.0000 - val_loss: 0.692
2
Epoch 19/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 0.9990 - loss: 0.6929 - val_Recall: 1.0000 - val_loss: 0.692
3
Epoch 20/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - Recall: 0.9993 - loss: 0.6927 - val_Recall: 1.0000 - val_loss: 0.692
2
Epoch 21/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - Recall: 0.9997 - loss: 0.6929 - val_Recall: 0.9984 - val_loss: 0.69
18
Epoch 22/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - Recall: 0.9854 - loss: 0.7353 - val_Recall: 0.9702 - val_loss: 0.691
9
Epoch 23/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - Recall: 0.9968 - loss: 0.6939 - val_Recall: 0.9992 - val_loss: 0.692
0
Epoch 24/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - Recall: 0.9994 - loss: 0.6924 - val_Recall: 0.9992 - val_loss: 0.692
0
Epoch 25/25
159/159 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - Recall: 0.9996 - loss: 0.6924 - val_Recall: 0.9992 - val_loss: 0.692
0
```

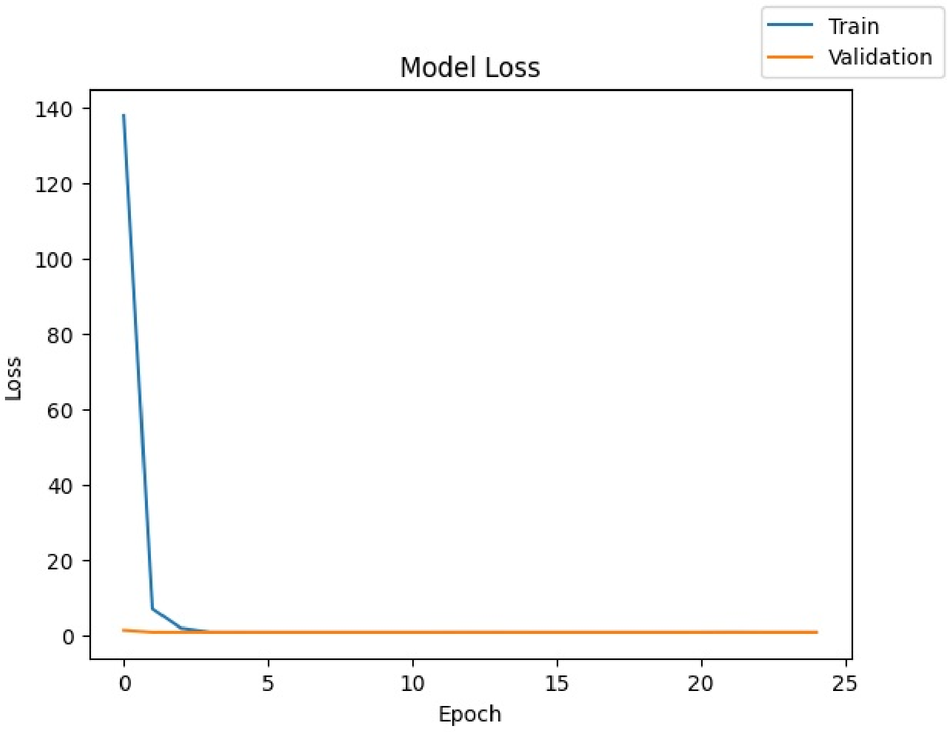
In [126…

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

```
Time taken in seconds  37.589741945266724
```

In [127…

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



In [128…

```
# Function for the chart

results.loc["Model 5"]=[2,[128,64],['relu','relu'],epochs,batch_size,'Adam + SMOTE',(end-start),history.history
```

In [129…

```
# Displaying the results
results
```

Out[129]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	time(secs)	Train_loss	Valid_loss
Model 0	-	-	-	25	64	SGD	160.587798	0.136982	0.121964
Model 1	2	[14, 7]	[relu, relu]	25	64	Adam	25.446552	0.090293	0.678093
Model 2	3	[64, 128, 64]	[relu, relu, relu]	25	64	Adam	40.763281	0.003548	1.504508
Model 3	3	[64, 128, 64]	[relu, relu, relu]	25	64	SGD + SMOTE	37.556867	0.161548	0.850490
Model 4	3	[64, 128, 64]	[relu, relu, relu]	25	64	Adam + SMOTE	40.759187	0.005454	3.676630
Model 5	2	[128, 64]	[relu, relu]	25	64	Adam + SMOTE	37.589742	0.692641	0.691960

In [130…

```
model_5_train_perf = model_performance_classification(model, x_train_smote, y_train_smote)
model_5_train_perf
```

318/318 ━━━━━━━━━━━━━━━━━━━━ **1s** 4ms/step

Out[130]:

	Accuracy	Recall	Precision	F1 Score
0	0.503688	0.503688	0.576246	0.339307

In [131…

```
model_5_valid_perf = model_performance_classification(model, x_val, y_val)
model_5_valid_perf
```

80/80 ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step

Out[131]:

	Accuracy	Recall	Precision	F1 Score
0	0.490366	0.490366	0.693874	0.326414

- In the final Model 5, it can be observed that the recall score stands out over all the other models. For it is the lowest overall for both training and validation data sets. However, it is generalized enough compared to the others which are not closely generalized at all.
- The time taken to run through all the epochs in this model is one of the lowest.
- From the recall scores being generalized, the training and validation loss scores are also balanced as well.

Model Performance Comparison and Final Model Selection

In [132…

```
# Displaying the reults for all the models
```

```
results
```

Out[132]:

	# hidden layers	# neurons - hidden layer	activation function - hidden layer	# epochs	batch size	optimizer	time(secs)	Train_loss	Valid_loss
Model 0	-	-	-	25	64	SGD	160.587798	0.136982	0.121964
Model 1	2	[14, 7]	[relu, relu]	25	64	Adam	25.446552	0.090293	0.678093
Model 2	3	[64, 128, 64]	[relu, relu, relu]	25	64	Adam	40.763281	0.003548	1.504508
Model 3	3	[64, 128, 64]	[relu, relu, relu]	25	64	SGD + SMOTE	37.556867	0.161548	0.850490
Model 4	3	[64, 128, 64]	[relu, relu, relu]	25	64	Adam + SMOTE	40.759187	0.005454	3.676630
Model 5	2	[128, 64]	[relu, relu]	25	64	Adam + SMOTE	37.589742	0.692641	0.691960

In [133…

```
# Display all the models training recall scores

models_train_comp_df = pd.concat(
    [
        model_0_train_perf.T,
        model_1_train_perf.T,
        model_2_train_perf.T,
        model_3_train_perf.T,
        model_4_train_perf.T,
        model_5_train_perf.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Neural Network with SGD",
    "Neural Network with Adam",
    "Neural Network with Adam & Dropout",
    "Neural Network with Balanced data, SMOTE, and SGD",
    "Neural Network with Balanced data, SMOTE, and Adam",
    "Neural Network with Balanced data, SMOTE, Adam, and Dropout",
]
```

In [134…

```
# Display all the models validation recall scores

models_valid_comp_df = pd.concat(
    [
        model_0_valid_perf.T,
        model_1_valid_perf.T,
        model_2_valid_perf.T,
        model_3_valid_perf.T,
        model_4_valid_perf.T,
        model_5_valid_perf.T,
    ],
    axis=1,
)
models_valid_comp_df.columns = [
    "Neural Network with SGD",
    "Neural Network with Adam",
    "Neural Network with Adam & Dropout",
    "Neural Network with Balanced data, SMOTE, and SGD",
    "Neural Network with Balanced data, SMOTE, and Adam",
    "Neural Network with Balanced data, SMOTE, Adam, and Dropout",
]
```

In [135…

```
# Training scores for all models
models_train_comp_df
```

Out[135]:

	Neural Network with SGD	Neural Network with Adam	Neural Network with Adam & Dropout	Neural Network with Balanced data, SMOTE, and SGD	Neural Network with Balanced data, SMOTE, and Adam	Neural Network with Balanced data, SMOTE, Adam, and Dropout
Accuracy	0.807625	0.972875	1.0	0.940612	0.998584	0.503688
Recall	0.807625	0.972875	1.0	0.940612	0.998584	0.503688
Precision	0.779839	0.972731	1.0	0.942162	0.998585	0.576246
F1 Score	0.748994	0.972518	1.0	0.940560	0.998584	0.339307

In [136…

```
# Validation scores for all models
models_valid_comp_df
```

Out[136]:

	Neural Network with SGD	Neural Network with Adam	Neural Network with Adam & Dropout	Neural Network with Balanced data, SMOTE, and SGD	Neural Network with Balanced data, SMOTE, and Adam	Neural Network with Balanced data, SMOTE, Adam, and Dropout
Accuracy	0.834000	0.82100	0.808000	0.766066	0.662000	0.490366
Recall	0.834000	0.82100	0.808000	0.766066	0.662000	0.490366
Precision	0.815012	0.80432	0.796422	0.771958	0.743291	0.693874
F1 Score	0.782367	0.81052	0.801464	0.764830	0.690787	0.326414

- Final Model: Model 0 (SGD)
- Reasoning: Model 0 is the best-performing model, with consistent recall scores for training and validation data sets. It has a good balance between performance and generalization, as indicated in the small gap between the training and validation recall scores and the lowest validation loss.

Final Model

In [137…

```
# clears the current Keras session, resetting all layers and models previously created, freeing up memory and r
tf.keras.backend.clear_session()
```

In [138…

```
# Importing necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Assuming you already have your dataset loaded into 'X' (features) and 'y' (labels)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the data
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

In [139…

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	188,736
dense_1 (Dense)	(None, 128)	8,320
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Total params: 616,133 (2.35 MB)
Trainable params: 205,377 (802.25 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 410,756 (1.57 MB)

In [140…

```
# Build the model
model = Sequential()

# Input layer and first hidden layer
model.add(Dense(14, activation='relu', input_dim=x_train_scaled.shape[1]))  # Adjust input_dim based on the num

# Second hidden layer
model.add(Dense(7, activation='relu'))

# Output layer
model.add(Dense(1, activation='sigmoid'))

# Compile the model using SGD optimizer
model.compile(optimizer=tf.keras.optimizers.SGD(),  # Stochastic Gradient Descent
              loss='binary_crossentropy',           # For binary classification
              metrics=[tf.keras.metrics.Recall(name="recall")])  # Track recall as a metric
```

In [141…

```
start = time.time()
```

```
history = model.fit(x_train, y_train, validation_data=(x_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()
```

```
Epoch 1/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - loss: 86685264.0000 - recall: 0.0423 - val_loss: 0.7015 - val_recal
l: 0.0000e+00
Epoch 2/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.5997 - recall: 0.0000e+00 - val_loss: 0.7234 - val_recall: 0
.0000e+00
Epoch 3/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5594 - recall: 0.0000e+00 - val_loss: 0.7480 - val_recall: 0
.0000e+00
Epoch 4/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5432 - recall: 0.0000e+00 - val_loss: 0.7714 - val_recall: 0
.0000e+00
Epoch 5/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5258 - recall: 0.0000e+00 - val_loss: 0.7918 - val_recall: 0
.0000e+00
Epoch 6/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5198 - recall: 0.0000e+00 - val_loss: 0.8092 - val_recall: 0
.0000e+00
Epoch 7/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5152 - recall: 0.0000e+00 - val_loss: 0.8236 - val_recall: 0
.0000e+00
Epoch 8/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5140 - recall: 0.0000e+00 - val_loss: 0.8356 - val_recall: 0
.0000e+00
Epoch 9/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5007 - recall: 0.0000e+00 - val_loss: 0.8453 - val_recall: 0
.0000e+00
Epoch 10/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5045 - recall: 0.0000e+00 - val_loss: 0.8533 - val_recall: 0
.0000e+00
Epoch 11/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5186 - recall: 0.0000e+00 - val_loss: 0.8601 - val_recall: 0
.0000e+00
Epoch 12/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5110 - recall: 0.0000e+00 - val_loss: 0.8655 - val_recall: 0
.0000e+00
Epoch 13/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5070 - recall: 0.0000e+00 - val_loss: 0.8699 - val_recall: 0
.0000e+00
Epoch 14/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5052 - recall: 0.0000e+00 - val_loss: 0.8734 - val_recall: 0
.0000e+00
Epoch 15/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5103 - recall: 0.0000e+00 - val_loss: 0.8765 - val_recall: 0
.0000e+00
Epoch 16/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5040 - recall: 0.0000e+00 - val_loss: 0.8788 - val_recall: 0
.0000e+00
Epoch 17/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.5098 - recall: 0.0000e+00 - val_loss: 0.8808 - val_recall: 0
.0000e+00
Epoch 18/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.5131 - recall: 0.0000e+00 - val_loss: 0.8824 - val_recall: 0
.0000e+00
Epoch 19/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.5026 - recall: 0.0000e+00 - val_loss: 0.8837 - val_recall: 0
.0000e+00
Epoch 20/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.5157 - recall: 0.0000e+00 - val_loss: 0.8848 - val_recall:
0.0000e+00
Epoch 21/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.5063 - recall: 0.0000e+00 - val_loss: 0.8856 - val_recall: 0
.0000e+00
Epoch 22/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.5086 - recall: 0.0000e+00 - val_loss: 0.8864 - val_recall: 0
.0000e+00
Epoch 23/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.5057 - recall: 0.0000e+00 - val_loss: 0.8869 - val_recall: 0
.0000e+00
Epoch 24/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.5129 - recall: 0.0000e+00 - val_loss: 0.8875 - val_recall: 0
.0000e+00
Epoch 25/25
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.4996 - recall: 0.0000e+00 - val_loss: 0.8877 - val_recall: 0
.0000e+00
```

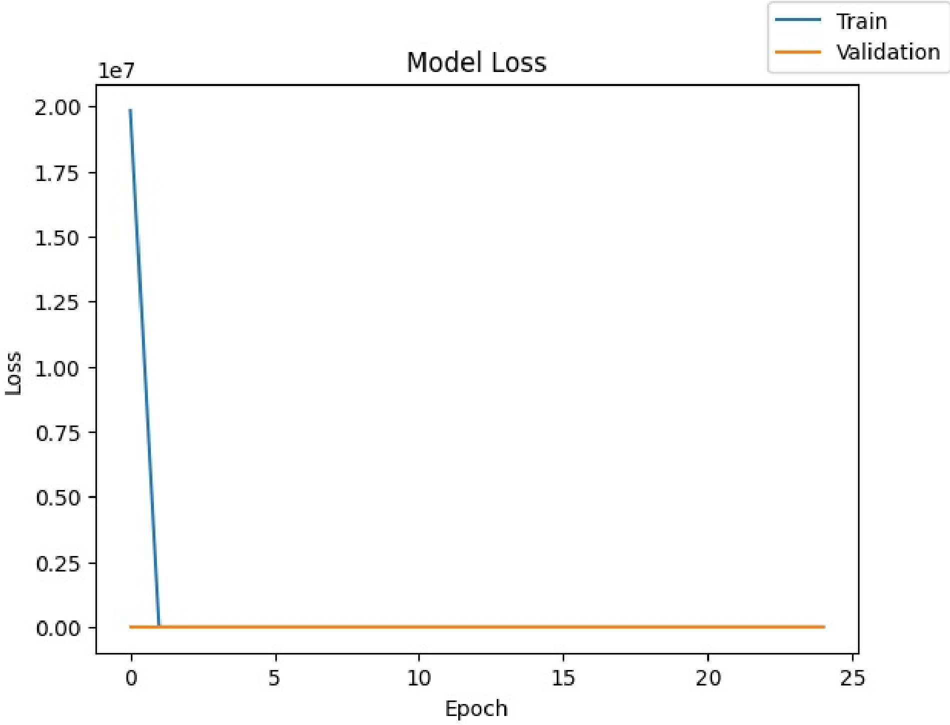
In [142…

```
print("Time Taken (Seconds): ",end-start)
```

```
Time Taken (Seconds):  26.248335599899292
```

In [143…

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



In [144…
```
model_0_final_train_perf = model_performance_classification(model, x_train, y_train)
model_0_final_train_perf
```

```
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
```

Out[144]:

	Accuracy	Recall	Precision	F1 Score
0	0.7945	0.7945	0.63123	0.703517

In [145…
```
model_0_final_test_perf = model_performance_classification(model, x_test, y_test)
model_0_final_test_perf
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

Out[145]:

	Accuracy	Recall	Precision	F1 Score
0	0.8035	0.8035	0.645612	0.715955

In [146…
```
y_train_pred = model.predict(x_train)
y_valid_pred = model.predict(x_val)
y_test_pred = model.predict(x_test)
```

```
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

In [147…
```
print("Classification Report - Train data",end="\n\n")
cr = classification_report(y_train,y_train_pred>0.5)
print(cr)
```

```
Classification Report - Train data

              precision    recall  f1-score   support

           0       0.79      1.00      0.89      6356
           1       0.00      0.00      0.00      1644

    accuracy                           0.79      8000
   macro avg       0.40      0.50      0.44      8000
weighted avg       0.63      0.79      0.70      8000
```

In [148…
```
print("Classification Report - Validation data",end="\n\n")
cr = classification_report(y_val,y_valid_pred > 0.5)
print(cr)
```

```
Classification Report - Validation data

              precision    recall  f1-score   support

           0       0.51      1.00      0.68      1303
           1       0.00      0.00      0.00      1240

    accuracy                           0.51      2543
   macro avg       0.26      0.50      0.34      2543
weighted avg       0.26      0.51      0.35      2543
```

In [149…

```
print("Classification Report - Test data",end="\n\n")
cr = classification_report(y_test,y_test_pred>0.5)
print(cr)
```

```
Classification Report - Test data

              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1607
           1       0.00      0.00      0.00       393

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.45      2000
weighted avg       0.65      0.80      0.72      2000
```

- Training Data Report
  - Class 0 (Did not exit):
    - Precision: 0.79, meaning that 79% of predicted class 0 instances were actually class 0.
    - Recall: 1.00, meaning the model predicted all actual class 0 instances correctly.
    - F1-score: 0.89, which is a high score due to perfect recall.
  - Class 1 (Exited):
    - Precision, Recall, F1-score: All are 0, indicating the model did not predict any instances of class 1 correctly. This suggests a complete failure to classify the minority class.

- Validation Data Report
  - Class 0 (Did not exit):
  - Precision: 0.51, meaning 51% of predicted class 0 instances were correct.
  - Recall: 1.00, meaning all actual class 0 instances were predicted correctly.
  - F1-score: 0.68, moderately high due to perfect recall.
  - Class 1 (Exited):
    - Precision, Recall, F1-score: All are 0, meaning the model again failed to identify any class 1 instances. Accuracy: 0.51, which suggests that the model is just slightly better than random guessing, driven entirely by predicting class 0.

- Test Data Report
  - Class 0 (Did not exit):
    - Precision: 0.80, meaning 80% of predicted class 0 instances were correct.
    - Recall: 1.00, meaning the model correctly predicted all class 0 instances.
    - F1-score: 0.89.
  - Class 1 (Exited):
    - Precision, Recall, F1-score: Again, all are 0, showing that the model completely fails to predict the minority class.

Conclusion

- The model is overfitting to the majority class (class 0) and completely ignoring the minority class (class 1), which is common in imbalanced datasets.
- Recall for class 1 is 0 across training, validation, and test datasets, meaning the model is not detecting any of the customers who exited.
- F1-scores for class 1 are also 0, indicating poor overall performance for class 1.

Actionable Insights and Business Recommendations

Actionable Insights

- Churn Prediction Model Performance
  - The model generally performed well in predicting customers who stayed (class 0), but struggled significantly to identify customers who churned (class 1), especially on validation and test data.
  - The best-performing model based on recall and other metrics was a Neural Network with Adam optimizer, but even this model showed biases, particularly in its ability to identify churning customers.
- Model Insights and Shortcomings
  - Imbalanced Data Issue: The main reason behind the model's poor performance in predicting customer churn was due to the imbalance between the two classes (most customers stayed, very few churned).
  - Despite applying SMOTE (Synthetic Minority Over-sampling Technique) to balance the data, the models still tended to perform better in classifying non-churning customers.
  - Overfitting: Some models showed signs of overfitting, as their training recall was very high (near 100%), but validation recall dropped significantly, indicating that these models performed well on the training data but struggled to generalize to unseen data.
  - Lack of Generalization: Even with various optimizers (SGD, Adam), dropout regularization, and data balancing techniques, the

model consistently failed to achieve satisfactory recall on class 1, meaning it often missed predicting customers who were likely to churn.
- Key Features Influencing Customer Churn Based on exploratory data analysis and the input features used in the model, the following factors appeared to significantly influence customer churn:
  - Credit Score and Age: Older customers with higher credit scores tended to stay longer with the bank.
  - Geography and Gender: Location and gender also seemed to play a role, with some regions showing higher churn rates.
  - Balance and NumOfProducts: Customers with a higher balance and more purchased products were less likely to churn. This suggests that offering more personalized services or increasing customer engagement through new products could help in retention.
  - IsActiveMember: Active members were less likely to leave, showing the importance of promoting engagement and regular usage of bank services.

Business Recommendations

- Improve Customer Engagement Strategies: Customers who actively use the bank's services are less likely to churn. Thus, increasing engagement through rewards, personalized offers, or financial advice could help improve customer retention.
- Target At-Risk Customers: Even though the model struggled with recall, the features identified (e.g., low balance, few products) provide clues on which types of customers are at a higher risk of leaving. Banks can focus on these customers with targeted campaigns or improved service offerings.
- Address Regional and Demographic Differences: Certain regions and demographics may be more prone to churn. Understanding the specific needs of these customer groups and tailoring services accordingly may reduce churn rates.
- Enhance Data Collection and Balance: Improving the collection of customer feedback, transaction patterns, and product usage data might provide a clearer picture of the reasons behind customer churn. Additionally, further exploration of advanced balancing techniques beyond SMOTE might improve prediction accuracy.

Future Directions

- Experiment with More Advanced Models: The current models, based on neural networks, could be improved by exploring other techniques like gradient boosting machines (GBMs), XGBoost, or ensemble methods to achieve better predictions for the minority class (churning customers).
- Adjust Class Weights: In future iterations, adjusting the class weights in the model's loss function could help the model focus more on the minority class (class 1), which could improve recall for customers who churn.

Power Ahead
