Problem Statement

Business Context

Renewable energy sources play an increasingly important role in the global energy mix, as the effort to reduce the environmental impact of energy production increases.

Out of all the renewable energy alternatives, wind energy is one of the most developed technologies worldwide. The U.S Department of Energy has put together a guide to achieving operational efficiency using predictive maintenance practices.

Predictive maintenance uses sensor information and analysis methods to measure and predict degradation and future component capability. The idea behind predictive maintenance is that failure patterns are predictable and if component failure can be predicted accurately and the component is replaced before it fails, the costs of operation and maintenance will be much lower.

The sensors fitted across different machines involved in the process of energy generation collect data related to various environmental factors (temperature, humidity, wind speed, etc.) and additional features related to various parts of the wind turbine (gearbox, tower, blades, break, etc.).

Objective

“ReneWind” is a company working on improving the machinery/processes involved in the production of wind energy using machine learning and has collected data of generator failure of wind turbines using sensors. They have shared a ciphered version of the data, as the data collected through sensors is confidential (the type of data collected varies with companies). Data has 40 predictors, 20000 observations in the training set and 5000 in the test set.

The objective is to build various classification models, tune them, and find the best one that will help identify failures so that the generators could be repaired before failing/breaking to reduce the overall maintenance cost. The nature of predictions made by the classification model will translate as follows:

- True positives (TP) are failures correctly predicted by the model. These will result in repairing costs.
- False negatives (FN) are real failures where there is no detection by the model. These will result in replacement costs.
- False positives (FP) are detections where there is no failure. These will result in inspection costs.

It is given that the cost of repairing a generator is much less than the cost of replacing it, and the cost of inspection is less than the cost of repair.

“1” in the target variables should be considered as “failure” and “0” represents “No failure”.

Data Description

The data provided is a transformed version of the original data which was collected using sensors.

- Train.csv - To be used for training and tuning of models.
- Test.csv - To be used only for testing the performance of the final best model.

Both the datasets consist of 40 predictor variables and 1 target variable.

Installing and Importing the necessary libraries

In [4]:

```
# Installing the libraries with the specified version
!pip install --no-deps tensorflow==2.18.0 scikit-learn==1.3.2 matplotlib===3.8.3 seaborn==0.13.2 numpy==1.26.4 pa
ndas==2.2.2 -q --user --no-warn-script-location
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 2.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.9/10.9 MB 83.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 100.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 81.0 MB/s eta 0:00:00
```

In [5]:

```
# 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.
from sklearn.metrics import confusion_matrix
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, classificat
ion_report
#Imports metrics from
from sklearn import metrics

#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
from tensorflow.keras.layers import Dropout

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

Note:

- After running the above cell, kindly restart the runtime (for Google Colab) or notebook kernel (for Jupyter Notebook), and run all cells sequentially from the next cell.
- On executing the above line of code, you might see a warning regarding package dependencies. This error message can be ignored as the above code ensures that all necessary libraries and their dependencies are maintained to successfully execute the code in ***this notebook***.

Loading the Data

In [6]:

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

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

In [7]:

```
train_path = '/content/drive/My Drive/python/Train_neural_network.csv'
test_path = '/content/drive/My Drive/python/Test_neural_network.csv'
df_orig_train = pd.read_csv(train_path)
df_orig_test = pd.read_csv(test_path)
df_train = df_orig_train.copy()
df_test = df_orig_test.copy()
```

Data Overview

In [8]:

```
df_train.head()
```

Out[8]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V32	V33	
0	-4.464606	-4.679129	3.101546	0.506130	-0.221083	-2.032511	-2.910870	0.050714	-1.522351	3.761892	...	3.059700	-1.690440	2
1	3.365912	3.653381	0.909671	-1.367528	0.332016	2.358938	0.732600	-4.332135	0.565695	-0.101080	...	-1.795474	3.032780	-2
2	-3.831843	-5.824444	0.634031	-2.418815	-1.773827	1.016824	-2.098941	-3.173204	-2.081860	5.392621	...	-0.257101	0.803550	4
3	1.618098	1.888342	7.046143	-1.147285	0.083080	-1.529780	0.207309	-2.493629	0.344926	2.118578	...	-3.584425	-2.577474	1
4	-0.111440	3.872488	-3.758361	-2.982897	3.792714	0.544960	0.205433	4.848994	-1.854920	-6.220023	...	8.265896	6.629213	-10

5 rows × 41 columns

In [9]:

```
df_test.head()
```

Out[9]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V32	V33
0	-0.613489	-3.819640	2.202302	1.300420	-1.184929	-4.495964	-1.835817	4.722989	1.206140	-0.341909	...	2.291204	-5.411388
1	0.389608	-0.512341	0.527053	-2.576776	-1.016766	2.235112	-0.441301	-4.405744	-0.332869	1.966794	...	-2.474936	2.493582
2	-0.874861	-0.640632	4.084202	-1.590454	0.525855	-1.957592	-0.695367	1.347309	-1.732348	0.466500	...	-1.318888	-2.997464
3	0.238384	1.458607	4.014528	2.534478	1.196987	-3.117330	-0.924035	0.269493	1.322436	0.702345	...	3.517918	-3.074085
4	5.828225	2.768260	-1.234530	2.809264	-1.641648	-1.406698	0.568643	0.965043	1.918379	-2.774855	...	1.773841	-1.501573

5 rows × 41 columns

In [10]:

```
df_train.shape
```

Out[10]:

```
(20000, 41)
```

In [11]:

```
df_test.shape
```

Out[11]:

```
(5000, 41)
```

In [12]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 41 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   V1      19982 non-null  float64
 1   V2      19982 non-null  float64
 2   V3      20000 non-null  float64
 3   V4      20000 non-null  float64
 4   V5      20000 non-null  float64
 5   V6      20000 non-null  float64
 6   V7      20000 non-null  float64
 7   V8      20000 non-null  float64
 8   V9      20000 non-null  float64
 9   V10     20000 non-null  float64
 10  V11     20000 non-null  float64
 11  V12     20000 non-null  float64
 12  V13     20000 non-null  float64
 13  V14     20000 non-null  float64
 14  V15     20000 non-null  float64
 15  V16     20000 non-null  float64
 16  V17     20000 non-null  float64
 17  V18     20000 non-null  float64
 18  V19     20000 non-null  float64
 19  V20     20000 non-null  float64
 20  V21     20000 non-null  float64
 21  V22     20000 non-null  float64
 22  V23     20000 non-null  float64
 23  V24     20000 non-null  float64
 24  V25     20000 non-null  float64
 25  V26     20000 non-null  float64
 26  V27     20000 non-null  float64
 27  V28     20000 non-null  float64
 28  V29     20000 non-null  float64
 29  V30     20000 non-null  float64
 30  V31     20000 non-null  float64
 31  V32     20000 non-null  float64
 32  V33     20000 non-null  float64
 33  V34     20000 non-null  float64
 34  V35     20000 non-null  float64
 35  V36     20000 non-null  float64
 36  V37     20000 non-null  float64
 37  V38     20000 non-null  float64
 38  V39     20000 non-null  float64
 39  V40     20000 non-null  float64
 40  Target  20000 non-null  int64  
dtypes: float64(40), int64(1)
memory usage: 6.3 MB
```

In [13]:

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 41 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   V1      4995 non-null   float64
 1   V2      4994 non-null   float64
 2   V3      5000 non-null   float64
 3   V4      5000 non-null   float64
 4   V5      5000 non-null   float64
 5   V6      5000 non-null   float64
 6   V7      5000 non-null   float64
 7   V8      5000 non-null   float64
 8   V9      5000 non-null   float64
 9   V10     5000 non-null   float64
 10  V11     5000 non-null   float64
 11  V12     5000 non-null   float64
 12  V13     5000 non-null   float64
 13  V14     5000 non-null   float64
 14  V15     5000 non-null   float64
 15  V16     5000 non-null   float64
 16  V17     5000 non-null   float64
 17  V18     5000 non-null   float64
 18  V19     5000 non-null   float64
 19  V20     5000 non-null   float64
 20  V21     5000 non-null   float64
 21  V22     5000 non-null   float64
 22  V23     5000 non-null   float64
 23  V24     5000 non-null   float64
 24  V25     5000 non-null   float64
 25  V26     5000 non-null   float64
 26  V27     5000 non-null   float64
 27  V28     5000 non-null   float64
 28  V29     5000 non-null   float64
 29  V30     5000 non-null   float64
 30  V31     5000 non-null   float64
 31  V32     5000 non-null   float64
 32  V33     5000 non-null   float64
 33  V34     5000 non-null   float64
 34  V35     5000 non-null   float64
 35  V36     5000 non-null   float64
 36  V37     5000 non-null   float64
 37  V38     5000 non-null   float64
 38  V39     5000 non-null   float64
 39  V40     5000 non-null   float64
 40  Target  5000 non-null   int64  
dtypes: float64(40), int64(1)
memory usage: 1.6 MB
```

In [14]:

```
train_summary = df_train.describe(include="all");
test_summary = df_test.describe(include="all");
print(train_summary)
print(test_summary)
```

```
                 V1            V2            V3            V4            V5  \
count  19982.000000  19982.000000  20000.000000  20000.000000  20000.000000   
mean      -0.271996      0.440430      2.484699     -0.083152     -0.053752   
std        3.441625      3.150784      3.388963      3.431595      2.104801   
min      -11.876451    -12.319951    -10.708139    -15.082052     -8.603361   
25%       -2.737146     -1.640674      0.206860     -2.347660     -1.535607   
50%       -0.747917      0.471536      2.255786     -0.135241     -0.101952   
75%        1.840112      2.543967      4.566165      2.130615      1.340480   
max       15.493002     13.089269     17.090919     13.236381      8.133797   

                 V6            V7            V8            V9           V10  \
count  20000.000000  20000.000000  20000.000000  20000.000000  20000.000000   
mean      -0.995443     -0.879325     -0.548195     -0.016808     -0.012998   
std        2.040970      1.761626      3.295756      2.160568      2.193201   
min      -10.227147     -7.949681    -15.657561     -8.596313     -9.853957   
25%       -2.347238     -2.030926     -2.642665     -1.494973     -1.411212   
50%       -1.000515     -0.917179     -0.389085     -0.067597      0.100973   
75%        0.380330      0.223695      1.722965      1.409203      1.477045   
max        6.975847      8.006091     11.679495      8.137580      8.108472   

       ...           V32           V33           V34           V35  \
count  ...  20000.000000  20000.000000  20000.000000  20000.000000   
mean   ...      0.303799      0.049825     -0.462702      2.229620   
```

```
std    ...      5.500400      3.575285      3.183841      2.937102
min    ...    -19.876502    -16.898353    -17.985094    -15.349803
25%    ...     -3.420469     -2.242857     -2.136984      0.336191
50%    ...      0.052073     -0.066249     -0.255008      2.098633
75%    ...      3.761722      2.255134      1.436935      4.064358
max    ...     23.633187     16.692486     14.358213     15.291065

                V36           V37           V38           V39           V40  \
count  20000.000000  20000.000000  20000.000000  20000.000000  20000.000000
mean       1.514809      0.011316     -0.344025      0.890653     -0.875630
std        3.800860      1.788165      3.948147      1.753054      3.012155
min      -14.833178     -5.478350    -17.375002     -6.438880    -11.023935
25%       -0.943809     -1.255819     -2.987638     -0.272250     -2.940193
50%        1.566526     -0.128435     -0.316849      0.919261     -0.920806
75%        3.983939      1.175533      2.279399      2.057540      1.119897
max       19.329576      7.467006     15.289923      7.759877     10.654265

             Target
count  20000.000000
mean       0.055500
std        0.228959
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000

[8 rows x 41 columns]
                V1            V2            V3            V4            V5  \
count  4995.000000  4994.000000  5000.000000  5000.000000  5000.000000
mean     -0.277622     0.397928     2.551787    -0.048943    -0.080120
std       3.466280     3.139562     3.326607     3.413937     2.110870
min     -12.381696   -10.716179    -9.237940   -14.682446    -7.711569
25%      -2.743691    -1.649211     0.314931    -2.292694    -1.615238
50%      -0.764767     0.427369     2.260428    -0.145753    -0.131890
75%       1.831313     2.444486     4.587000     2.166468     1.341197
max      13.504352    14.079073    15.314503    12.140157     7.672835

                V6            V7            V8            V9           V10  ...  \
count  5000.000000  5000.000000  5000.000000  5000.000000  5000.000000  ...
mean     -1.042138    -0.907922    -0.574592     0.030121     0.018524  ...
std       2.005444     1.769017     3.331911     2.174139     2.145437  ...
min      -8.924196    -8.124230   -12.252731    -6.785495    -8.170956  ...
25%      -2.368853    -2.054259    -2.642088    -1.455712    -1.353320  ...
50%      -1.048571    -0.939695    -0.357943    -0.079891     0.166292  ...
75%       0.307555     0.212228     1.712896     1.449548     1.511248  ...
max       5.067685     7.616182    10.414722     8.850720     6.598728  ...

               V32           V33           V34           V35           V36  \
count  5000.000000  5000.000000  5000.000000  5000.000000  5000.000000
mean      0.232567    -0.080115    -0.392663     2.211205     1.594845
std       5.585628     3.538624     3.166101     2.948426     3.774970
min     -17.244168   -14.903781   -14.699725   -12.260591   -12.735567
25%      -3.556267    -2.348121    -2.009604     0.321818    -0.866066
50%      -0.076694    -0.159713    -0.171745     2.111750     1.702964
75%       3.751857     2.099160     1.465402     4.031639     4.104409
max      26.539391    13.323517    12.146302    13.489237    17.116122

               V37           V38           V39           V40       Target
count  5000.000000  5000.000000  5000.000000  5000.000000  5000.000000
mean      0.022931    -0.405659     0.938800    -0.932406     0.056400
std       1.785320     3.968936     1.716502     2.978193     0.230716
min      -5.079070   -15.334533    -5.451050   -10.076234     0.000000
25%      -1.240526    -2.984480    -0.208024    -2.986587     0.000000
50%      -0.110415    -0.381162     0.959152    -1.002764     0.000000
75%       1.237522     2.287998     2.130769     1.079738     0.000000
max       6.809938    13.064950     7.182237     8.698460     1.000000

[8 rows x 41 columns]
```

In [15]:

```
#the Columns V1 and V2 and 18 missing values. Lets fix them.
# Create an imputer object with the mean strategy
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Fit and transform the specified columns
df_train[['V1', 'V2']] = imputer.fit_transform(df_train[['V1', 'V2']])
df_test[['V1', 'V2']] = imputer.fit_transform(df_test[['V1', 'V2']])
```

In [16]:

```
# lets recheck the columns now

print(df_train.info())
print(df_test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 41 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   V1      20000 non-null  float64
 1   V2      20000 non-null  float64
 2   V3      20000 non-null  float64
 3   V4      20000 non-null  float64
 4   V5      20000 non-null  float64
 5   V6      20000 non-null  float64
 6   V7      20000 non-null  float64
 7   V8      20000 non-null  float64
 8   V9      20000 non-null  float64
 9   V10     20000 non-null  float64
 10  V11     20000 non-null  float64
 11  V12     20000 non-null  float64
 12  V13     20000 non-null  float64
 13  V14     20000 non-null  float64
 14  V15     20000 non-null  float64
 15  V16     20000 non-null  float64
 16  V17     20000 non-null  float64
 17  V18     20000 non-null  float64
 18  V19     20000 non-null  float64
 19  V20     20000 non-null  float64
 20  V21     20000 non-null  float64
 21  V22     20000 non-null  float64
 22  V23     20000 non-null  float64
 23  V24     20000 non-null  float64
 24  V25     20000 non-null  float64
 25  V26     20000 non-null  float64
 26  V27     20000 non-null  float64
 27  V28     20000 non-null  float64
 28  V29     20000 non-null  float64
 29  V30     20000 non-null  float64
 30  V31     20000 non-null  float64
 31  V32     20000 non-null  float64
 32  V33     20000 non-null  float64
 33  V34     20000 non-null  float64
 34  V35     20000 non-null  float64
 35  V36     20000 non-null  float64
 36  V37     20000 non-null  float64
 37  V38     20000 non-null  float64
 38  V39     20000 non-null  float64
 39  V40     20000 non-null  float64
 40  Target  20000 non-null  int64  
dtypes: float64(40), int64(1)
memory usage: 6.3 MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 41 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   V1      5000 non-null   float64
 1   V2      5000 non-null   float64
 2   V3      5000 non-null   float64
 3   V4      5000 non-null   float64
 4   V5      5000 non-null   float64
 5   V6      5000 non-null   float64
 6   V7      5000 non-null   float64
 7   V8      5000 non-null   float64
 8   V9      5000 non-null   float64
 9   V10     5000 non-null   float64
 10  V11     5000 non-null   float64
 11  V12     5000 non-null   float64
 12  V13     5000 non-null   float64
 13  V14     5000 non-null   float64
 14  V15     5000 non-null   float64
 15  V16     5000 non-null   float64
 16  V17     5000 non-null   float64
 17  V18     5000 non-null   float64
 18  V19     5000 non-null   float64
 19  V20     5000 non-null   float64
 20  V21     5000 non-null   float64
 21  V22     5000 non-null   float64
```

```
 22  V23     5000 non-null   float64
 23  V24     5000 non-null   float64
 24  V25     5000 non-null   float64
 25  V26     5000 non-null   float64
 26  V27     5000 non-null   float64
 27  V28     5000 non-null   float64
 28  V29     5000 non-null   float64
 29  V30     5000 non-null   float64
 30  V31     5000 non-null   float64
 31  V32     5000 non-null   float64
 32  V33     5000 non-null   float64
 33  V34     5000 non-null   float64
 34  V35     5000 non-null   float64
 35  V36     5000 non-null   float64
 36  V37     5000 non-null   float64
 37  V38     5000 non-null   float64
 38  V39     5000 non-null   float64
 39  V40     5000 non-null   float64
 40  Target  5000 non-null   int64
dtypes: float64(40), int64(1)
memory usage: 1.6 MB
None
```

Exploratory Data Analysis

Univariate analysis

In [17]:

```
#Lets use the funcitons from the last project to make things easy
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a triangle will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins
    ) 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
```

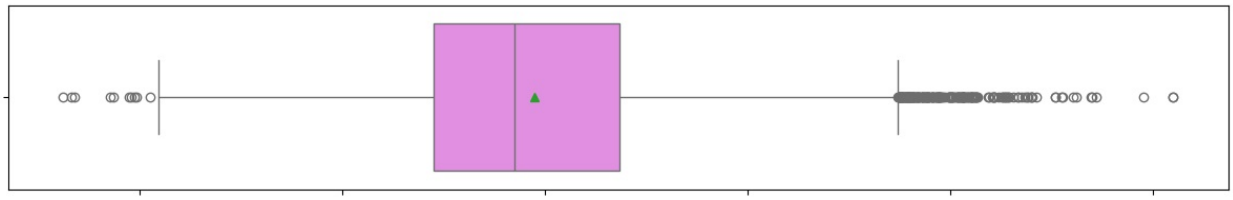
In [18]:

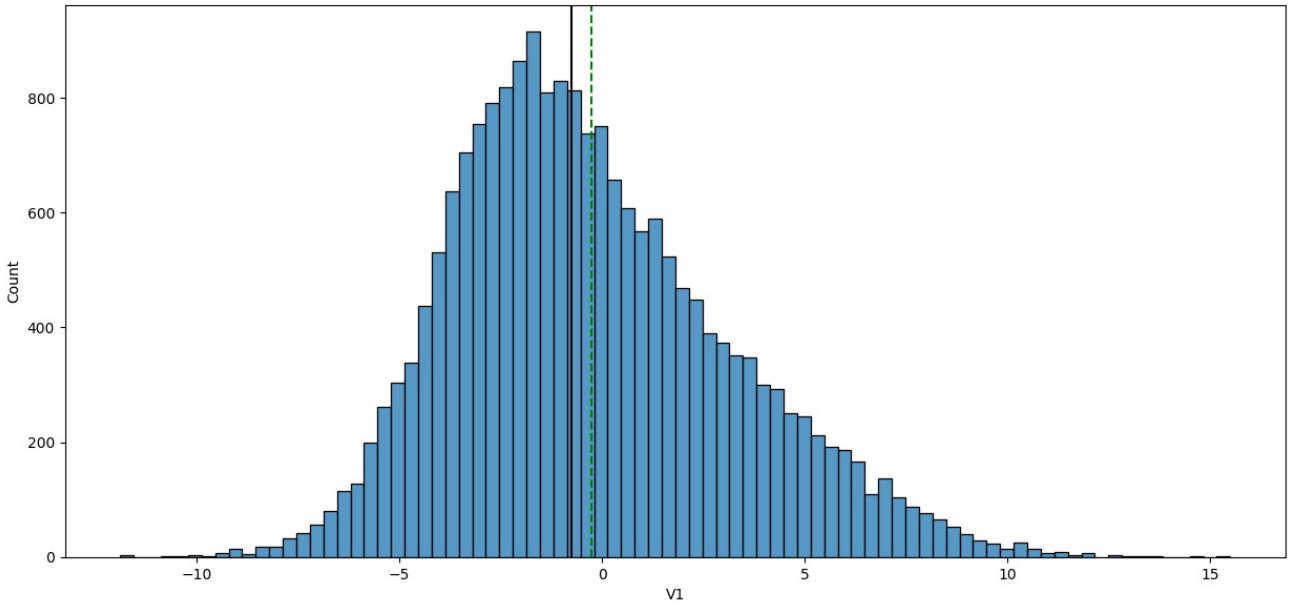
```
# function to create labeled barplots


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
```

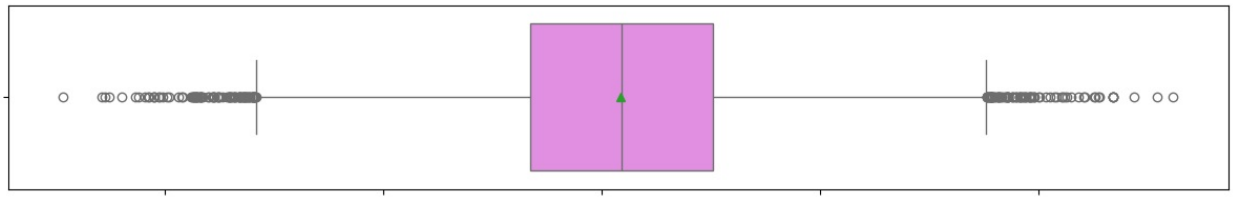
In [19]:

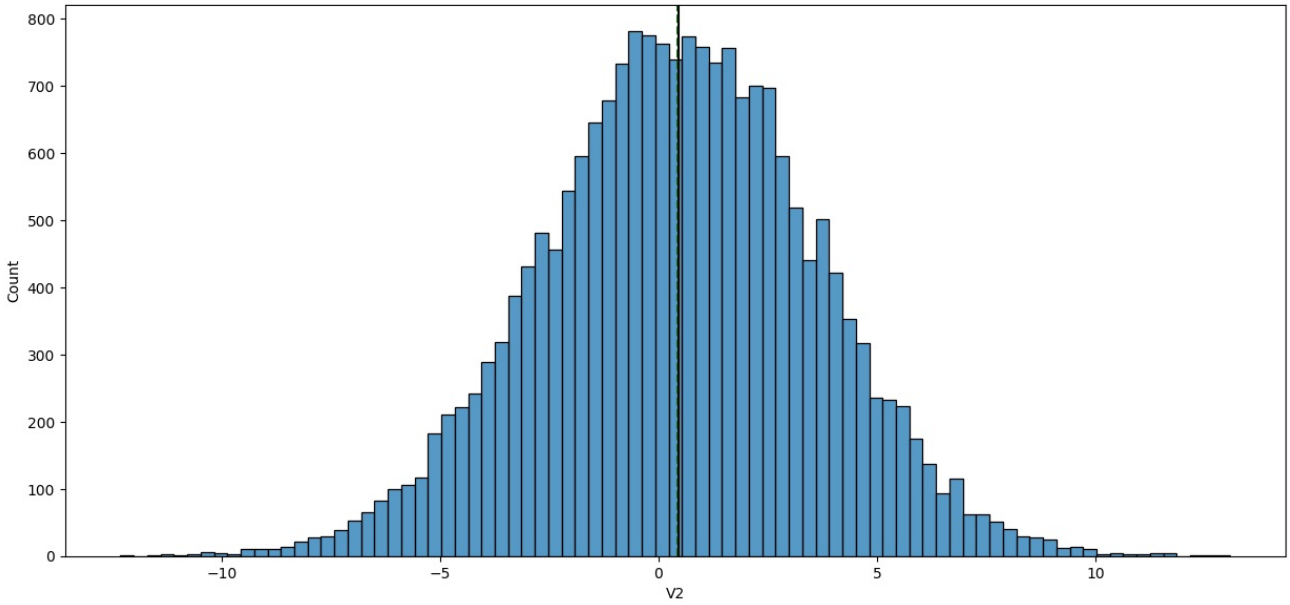
```
df_train_univariate = df_train.drop(columns=["Target"], axis=1)

for column in df_train_univariate.columns:
      histogram_boxplot(df_train_univariate, column, figsize=(15, 10), kde=False, bins=None)
      print(f"Processing column: {column}")
```

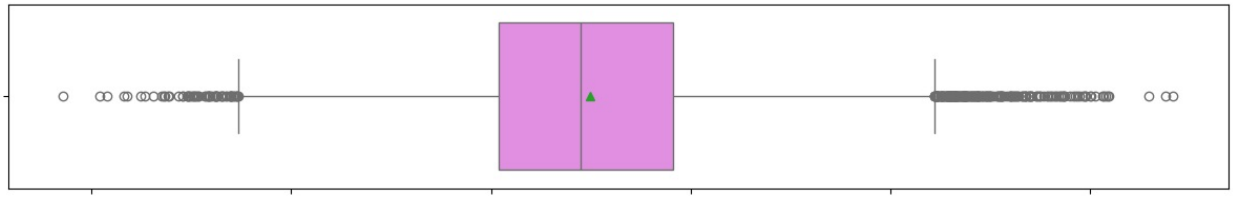
```
Processing column: V1
Processing column: V2
Processing column: V3
Processing column: V4
Processing column: V5
Processing column: V6
Processing column: V7
Processing column: V8
Processing column: V9
Processing column: V10
Processing column: V11
Processing column: V12
Processing column: V13
Processing column: V14
Processing column: V15
Processing column: V16
Processing column: V17
Processing column: V18
Processing column: V19
Processing column: V20
Processing column: V21
Processing column: V22
Processing column: V23
Processing column: V24
Processing column: V25
Processing column: V26
Processing column: V27
Processing column: V28
Processing column: V29
Processing column: V30
Processing column: V31
Processing column: V32
Processing column: V33
Processing column: V34
Processing column: V35
Processing column: V36
Processing column: V37
Processing column: V38
Processing column: V39
Processing column: V40
```

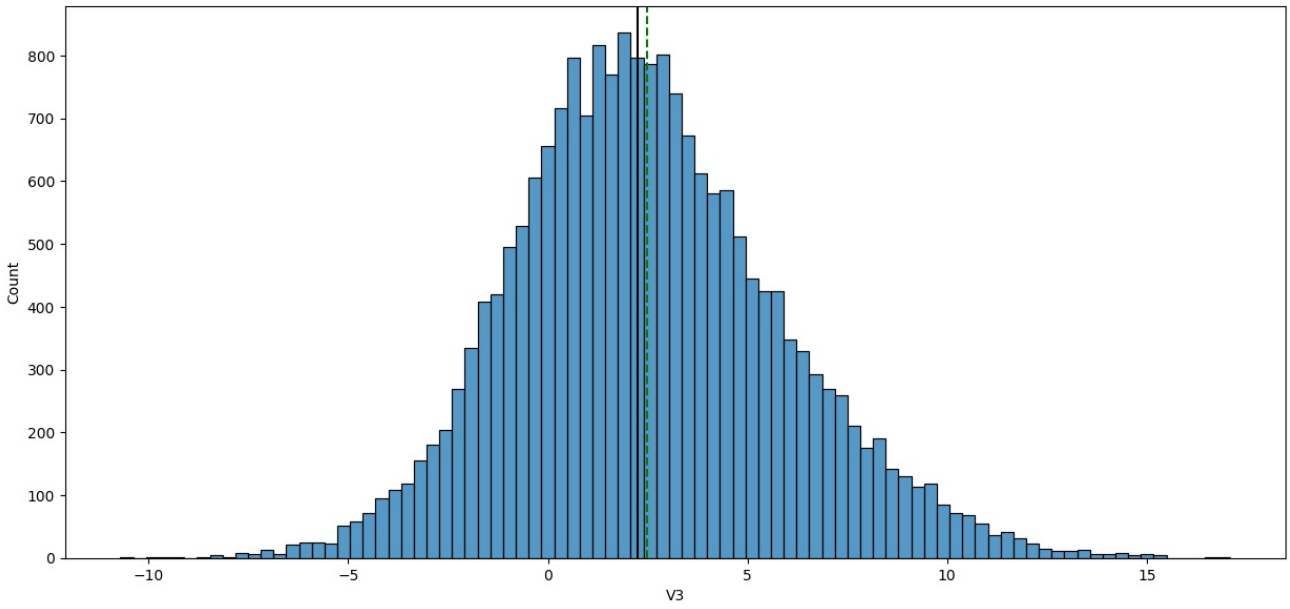


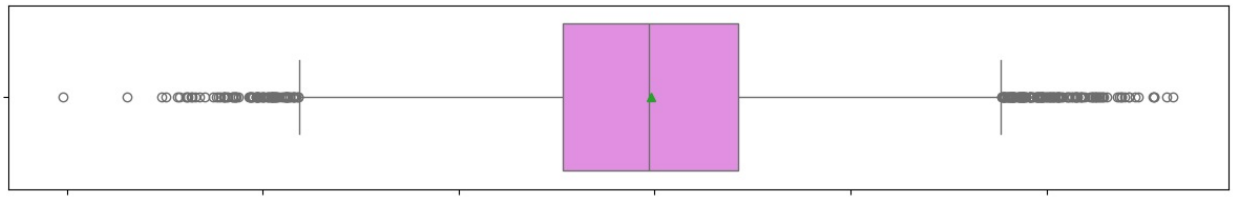


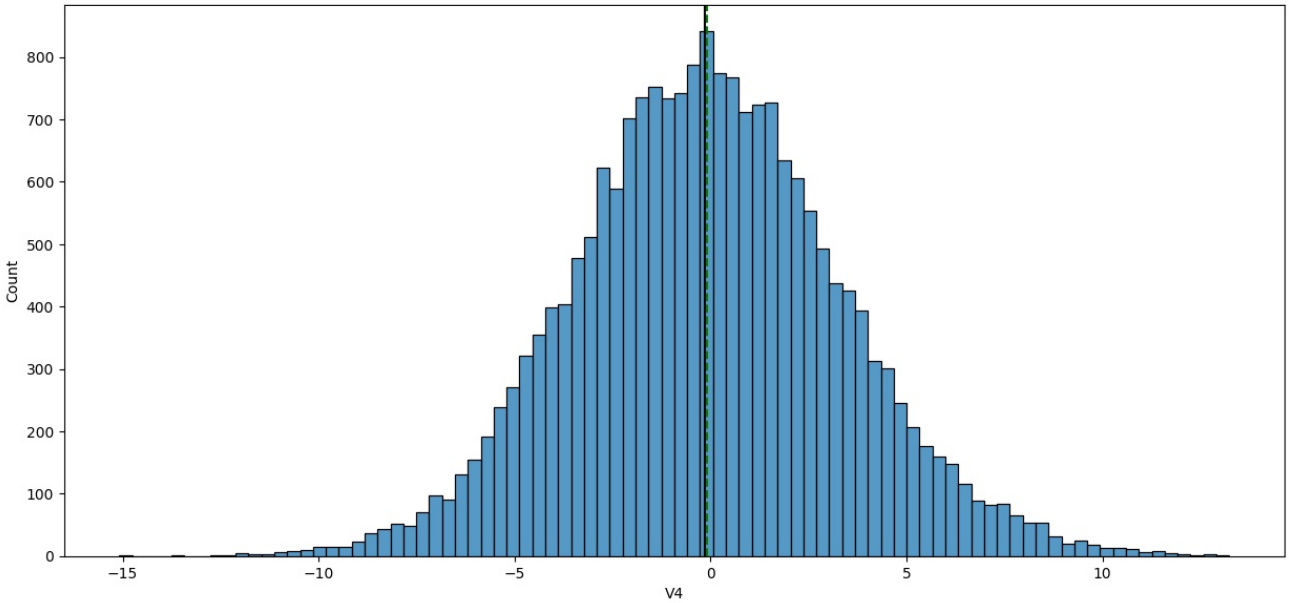


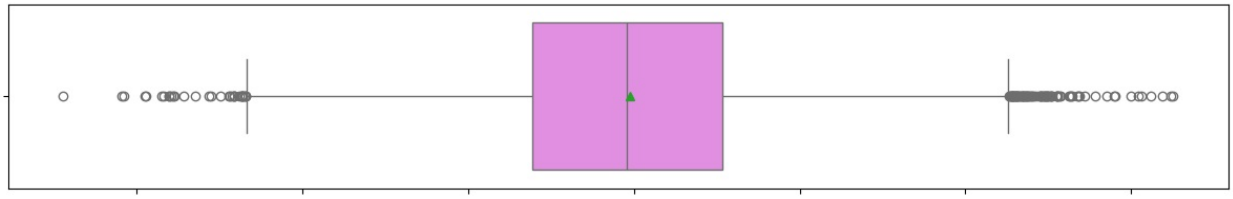

Count
V2

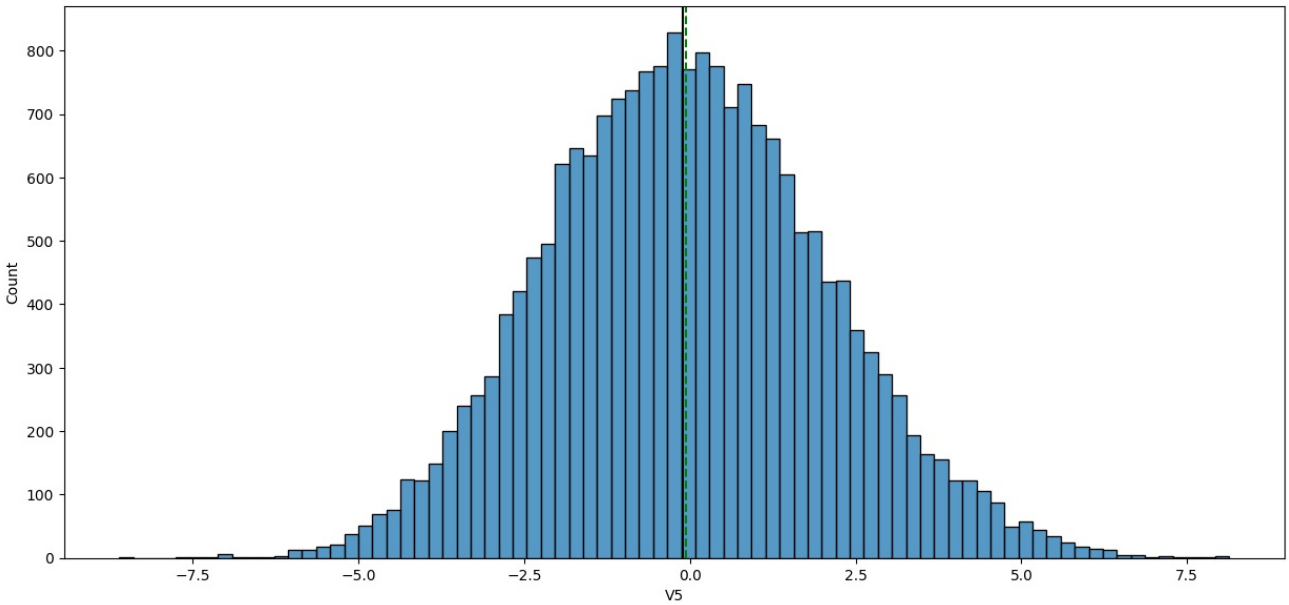


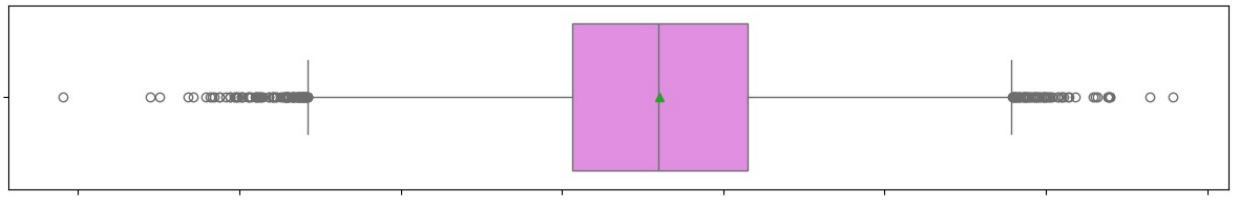

Count
V3

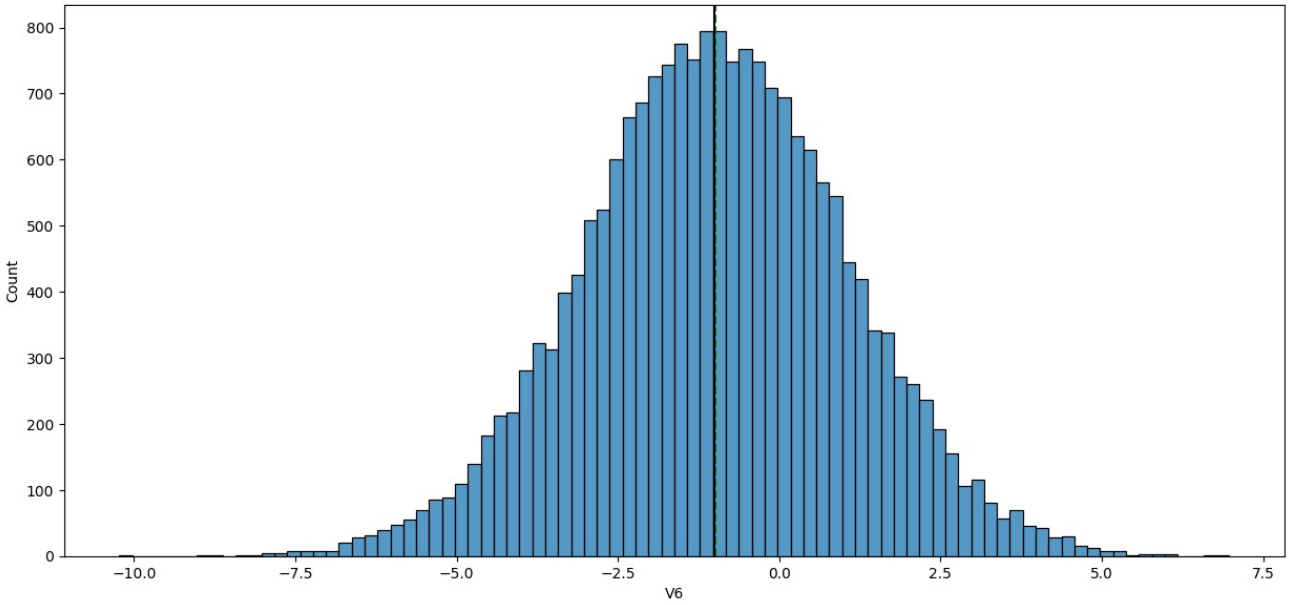


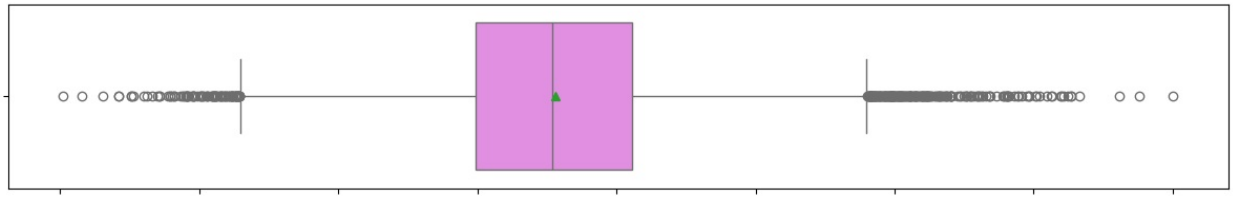

Count
V4

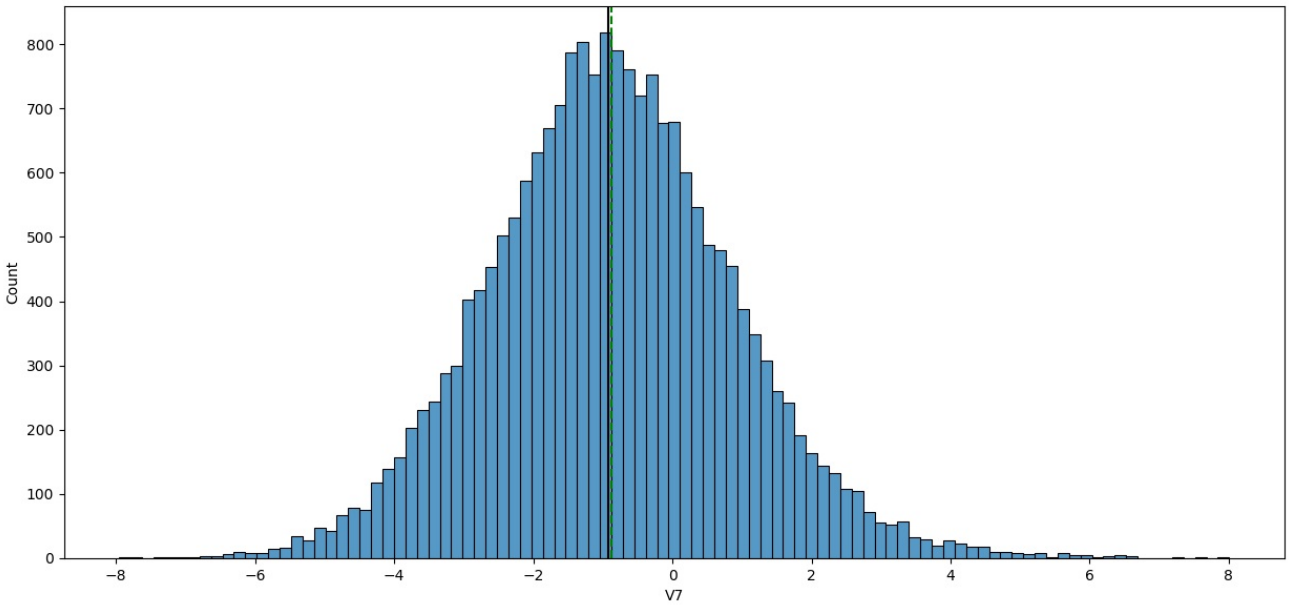


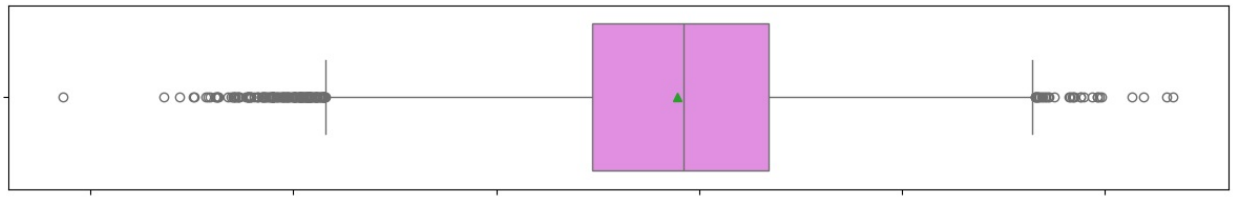

Count
V5

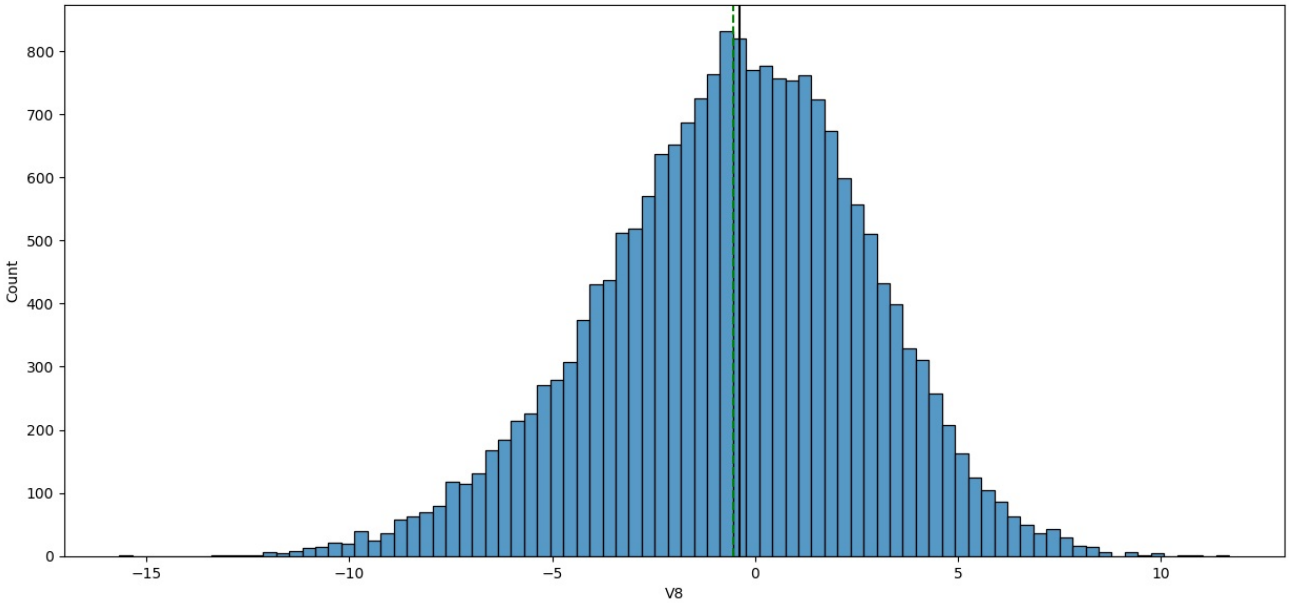


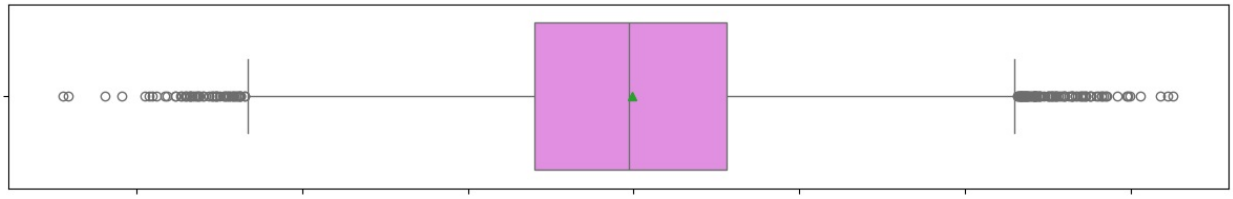

Count
800
700
600
500
400
300
200
100
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V6

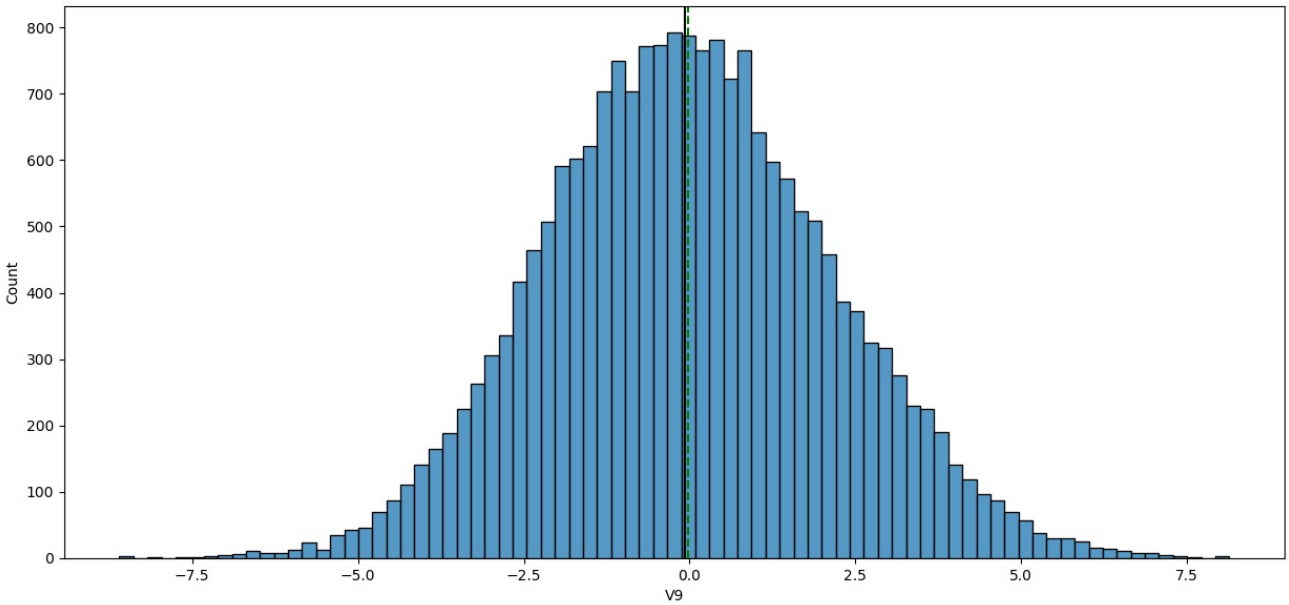


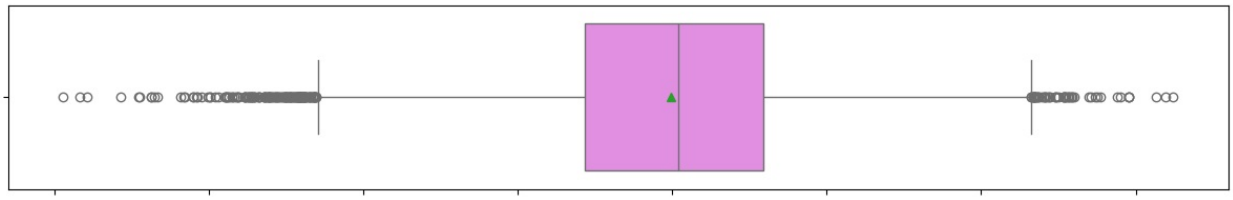

Count
800
700
600
500
400
300
200
100
0
−8
−6
−4
−2
0
2
4
6
8
V7

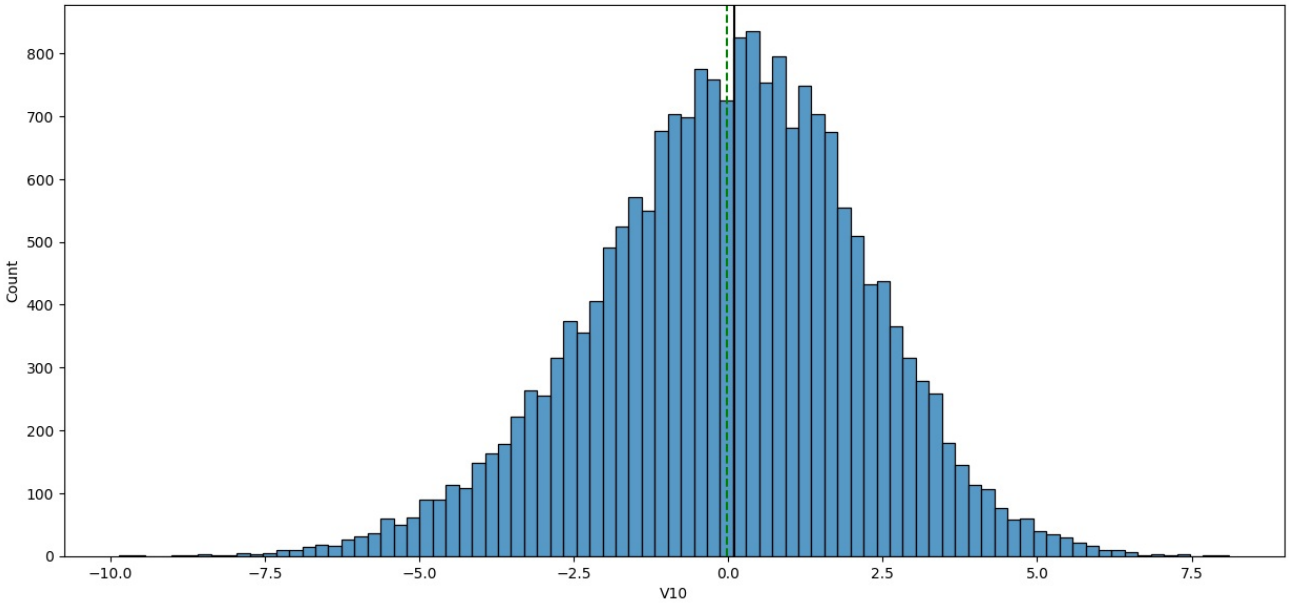


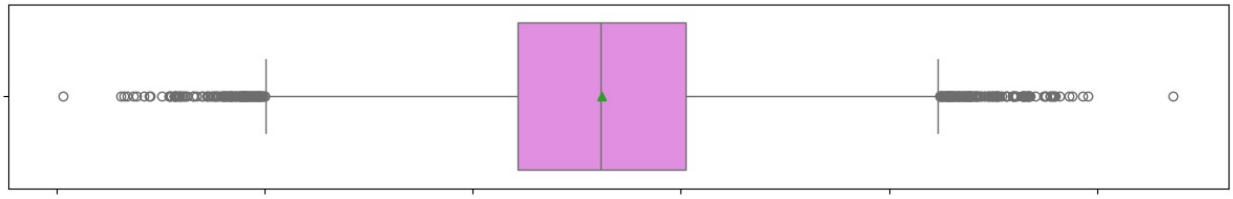

Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V8

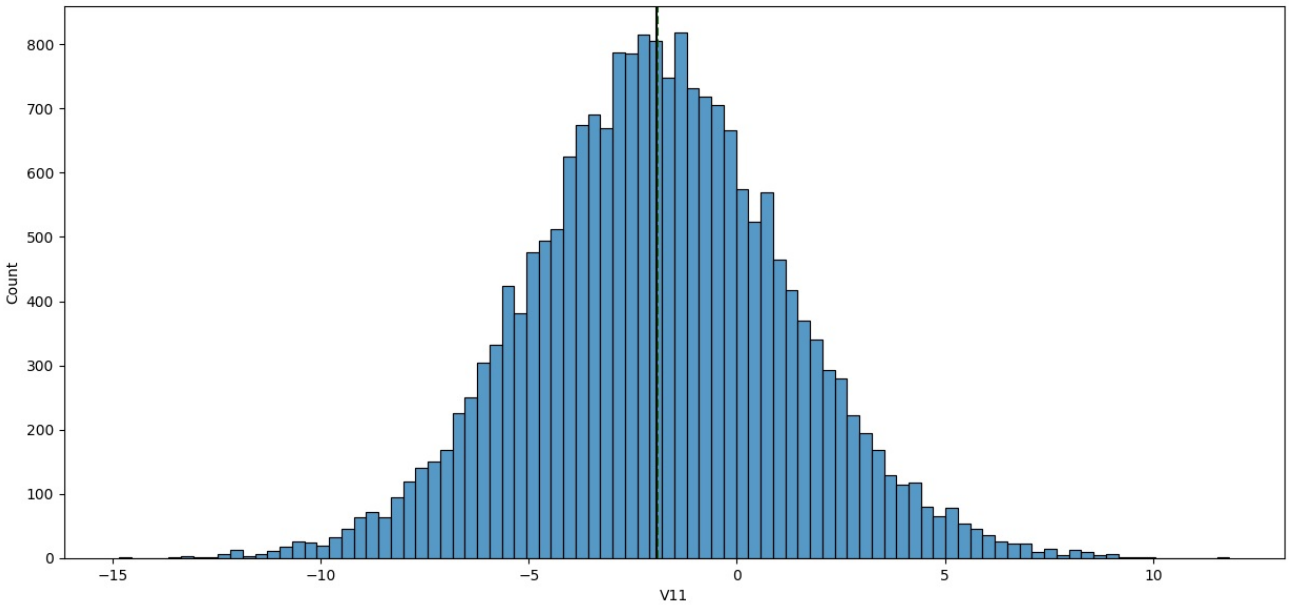


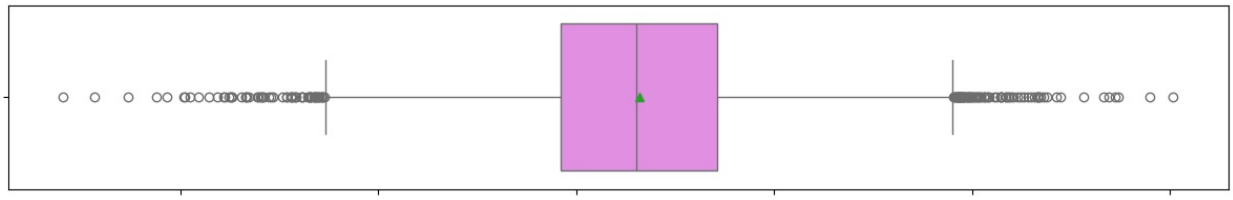

Count
800
700
600
500
400
300
200
100
0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V9

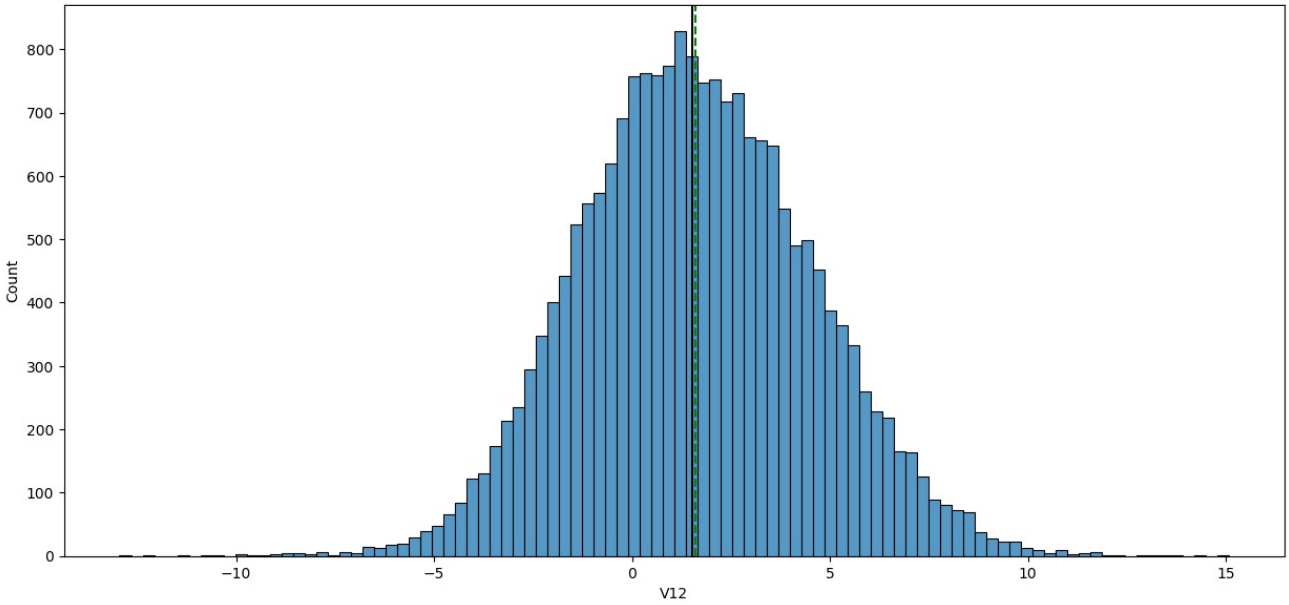


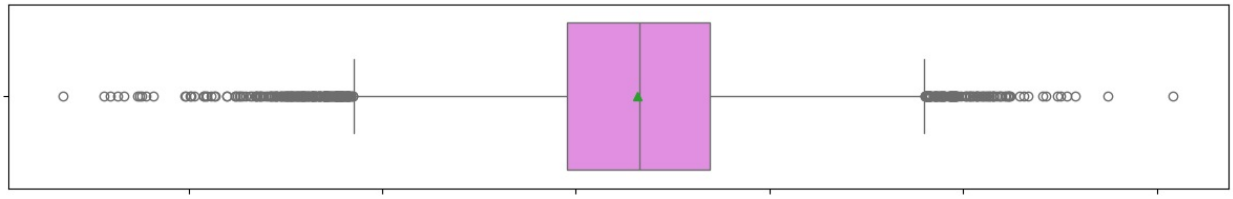

Count
800
700
600
500
400
300
200
100
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V10

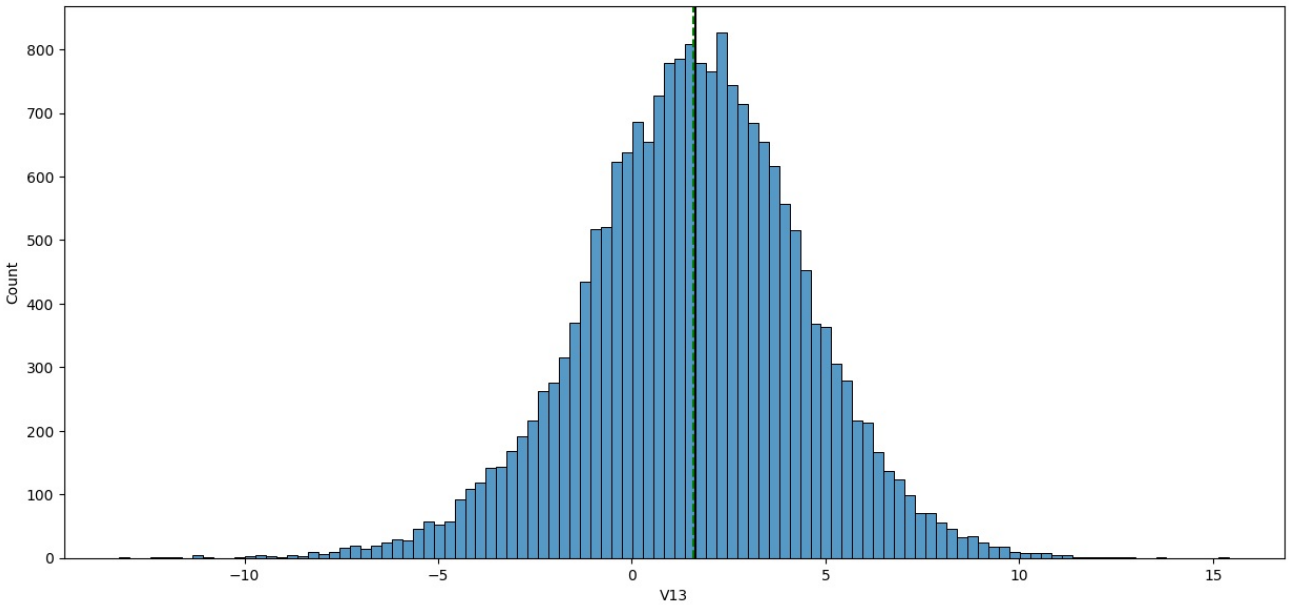


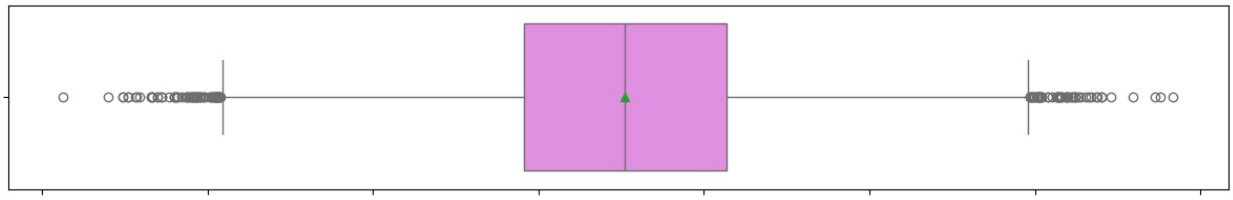

Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V11

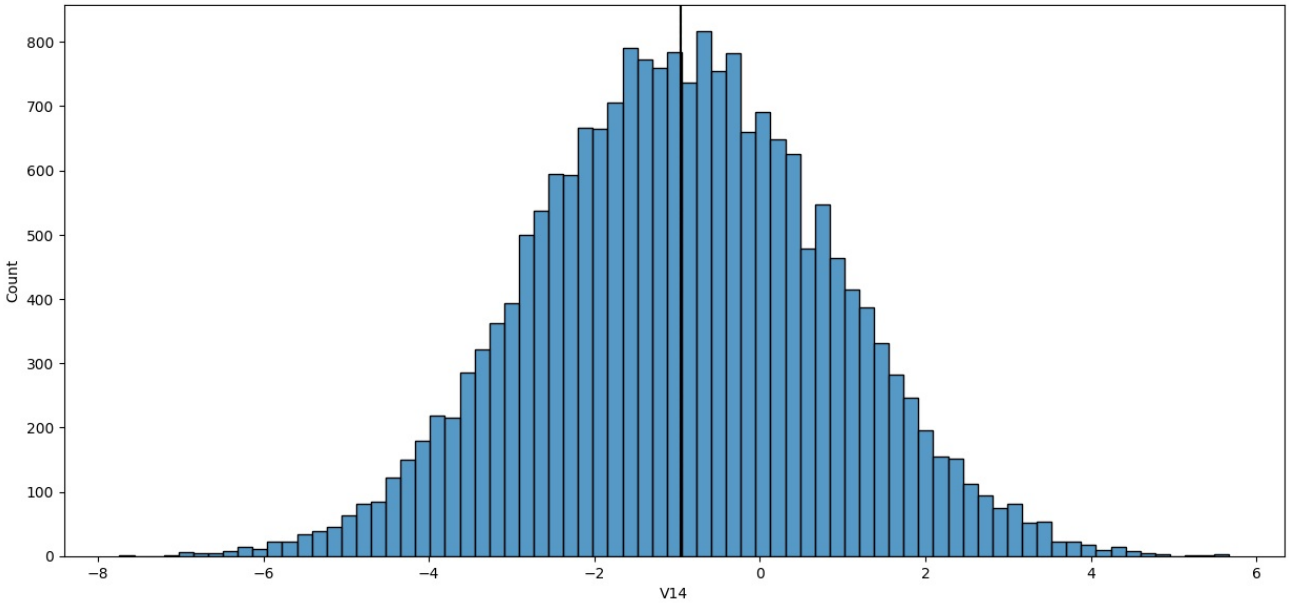


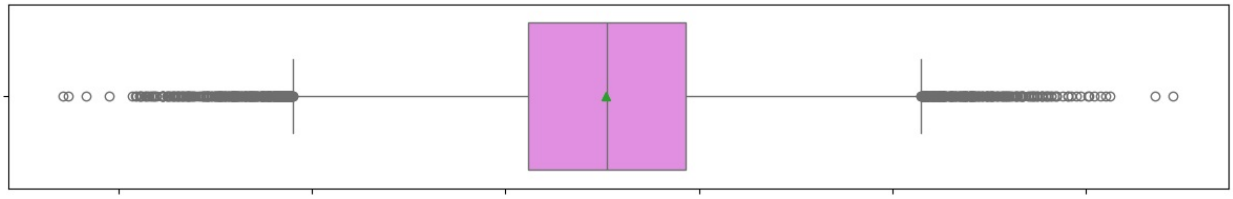

Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
15
V12

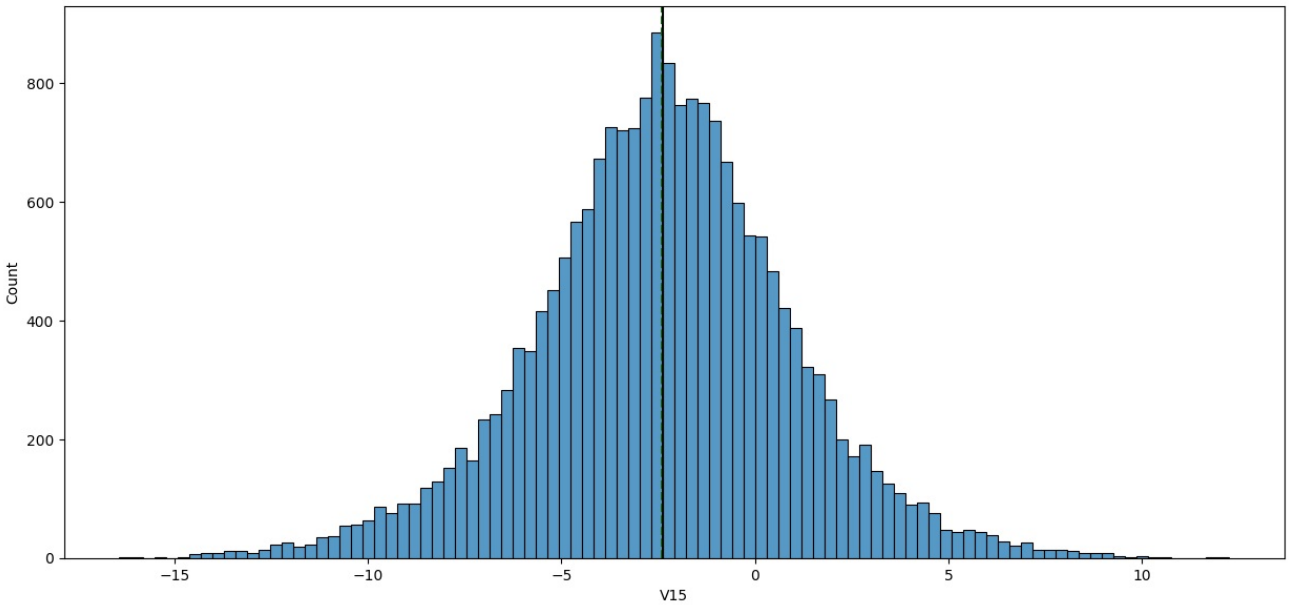


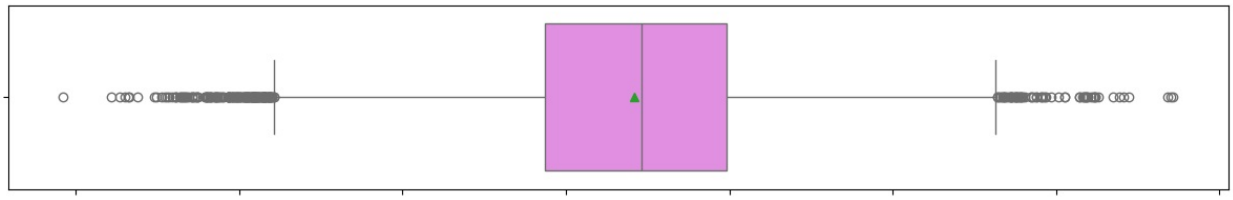

Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
15
V13

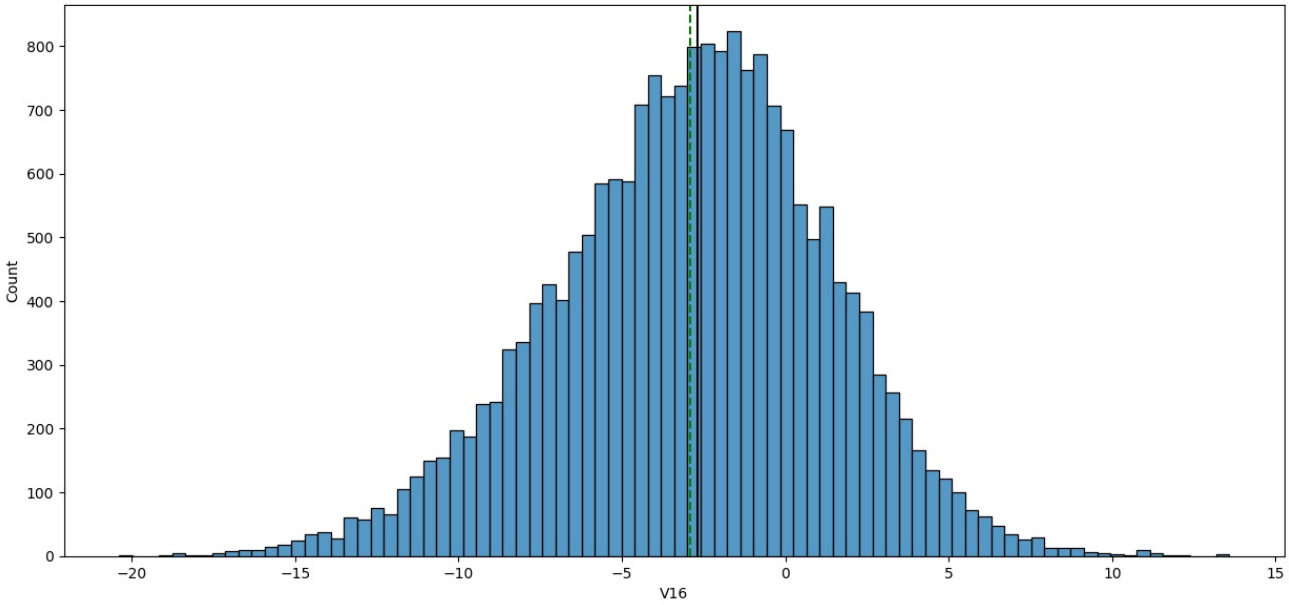


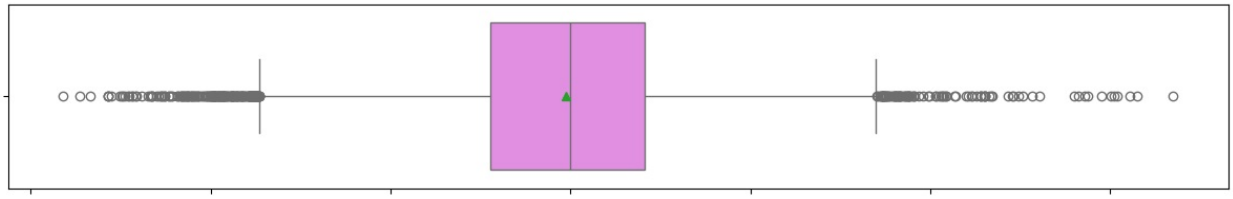

Count
V14

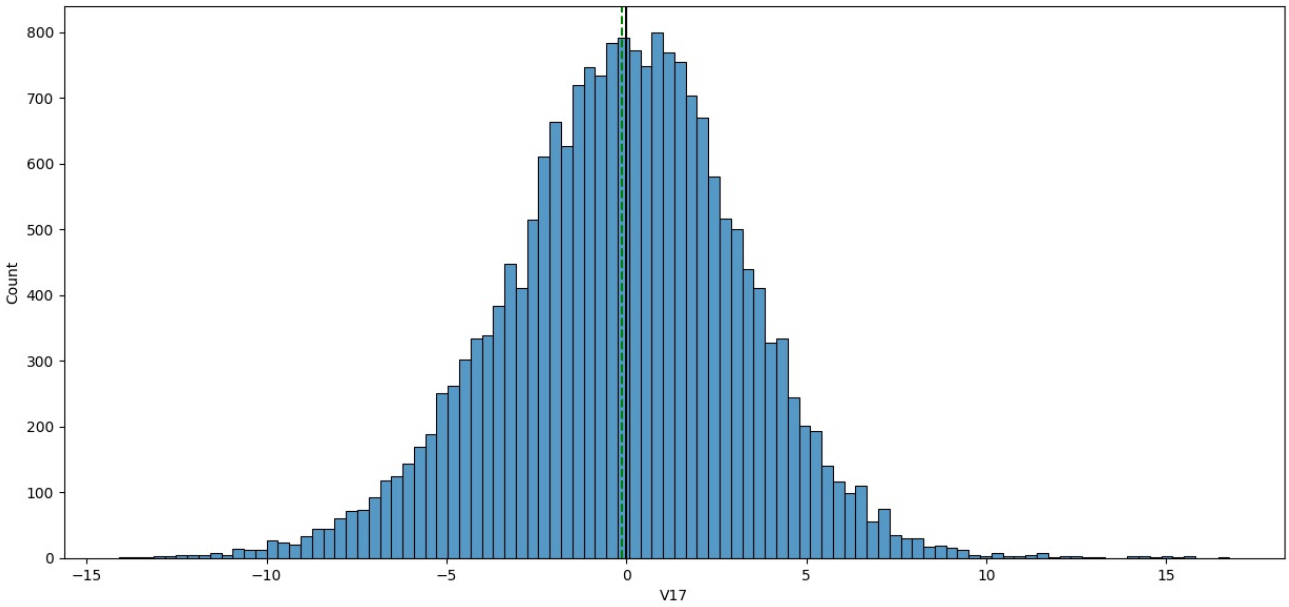


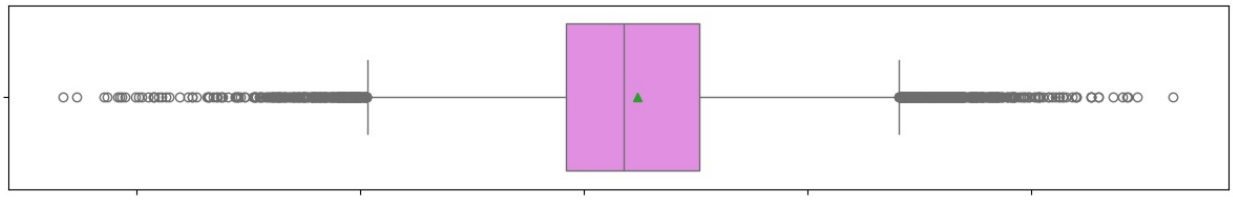

Count
V15

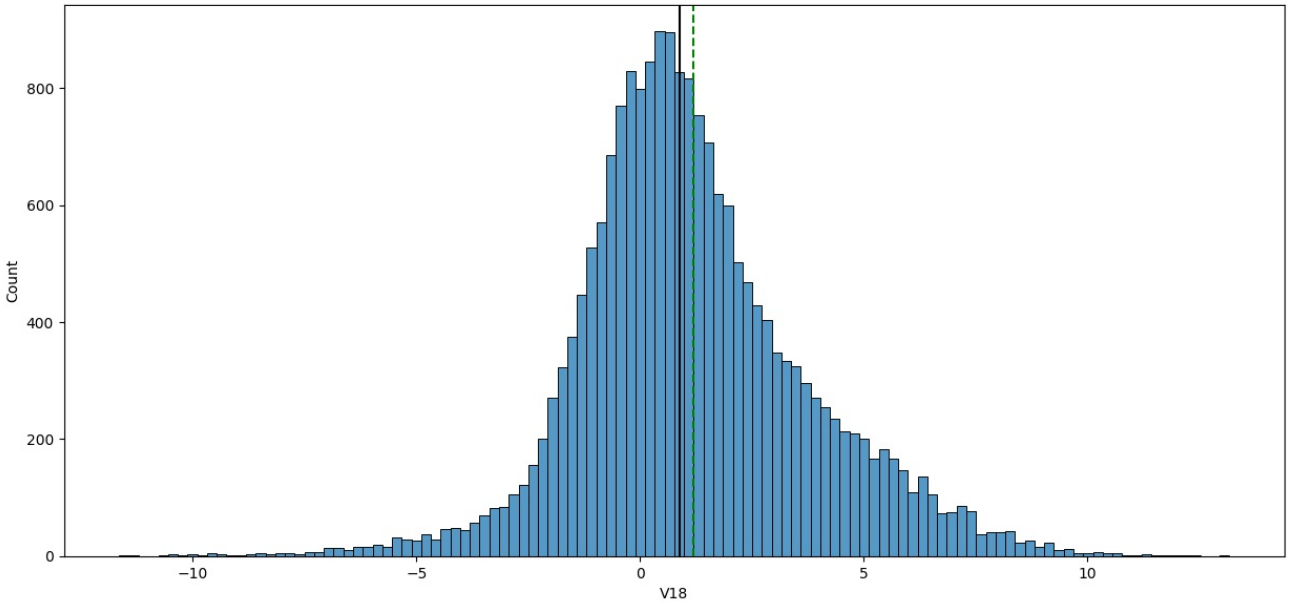


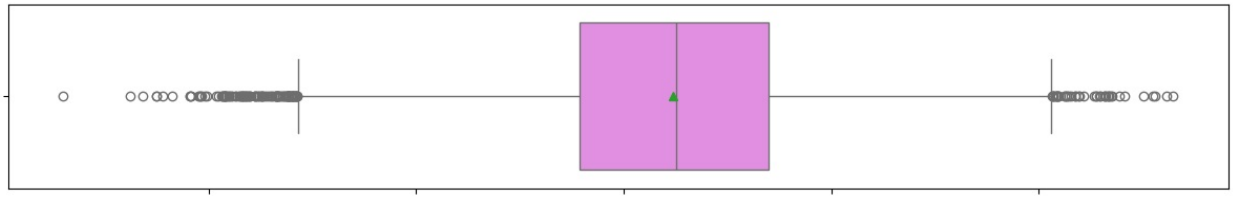

Count
V16

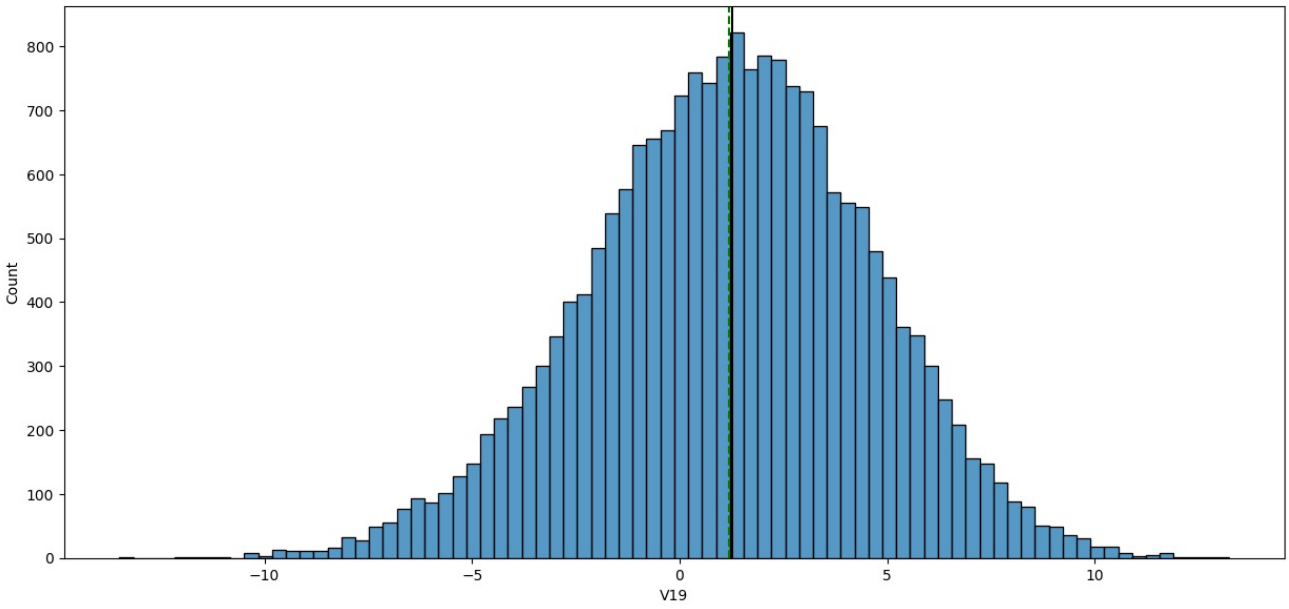


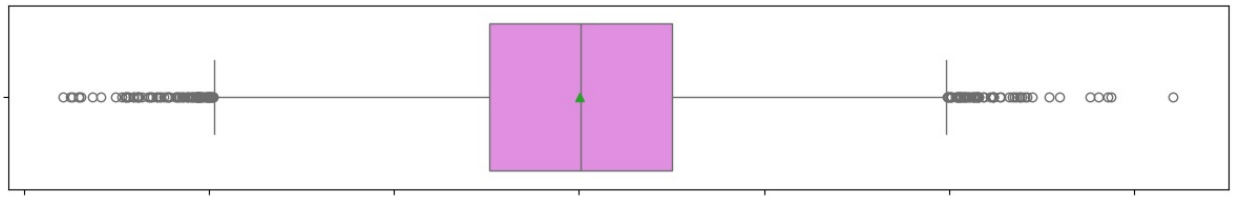

Count
V17

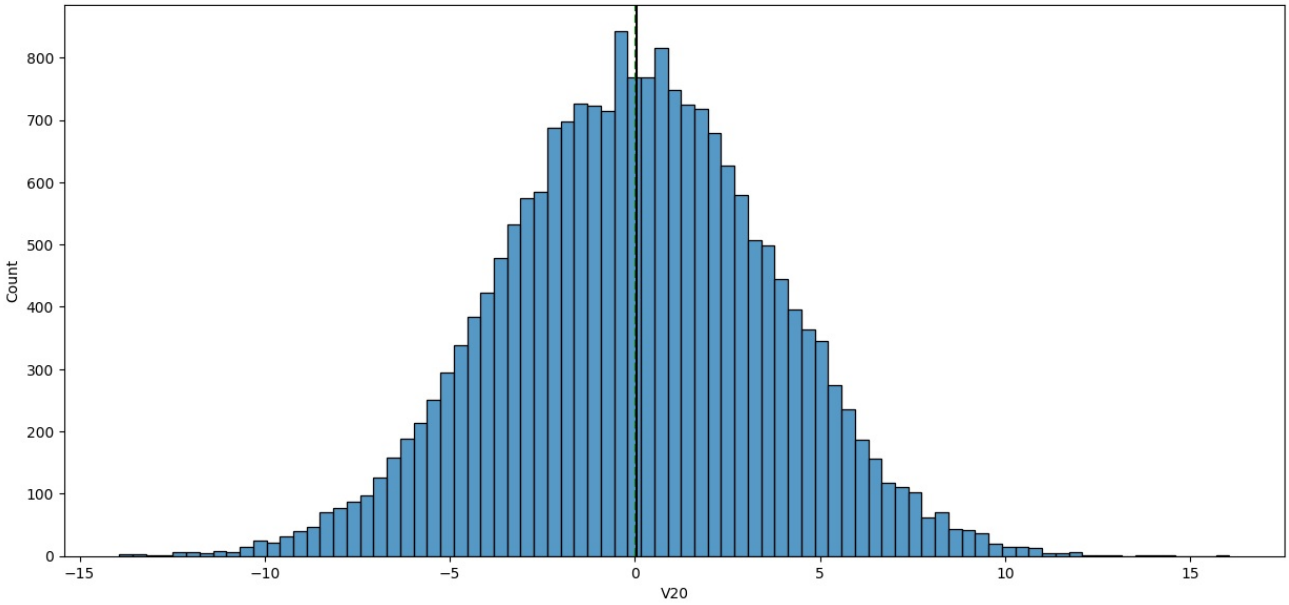


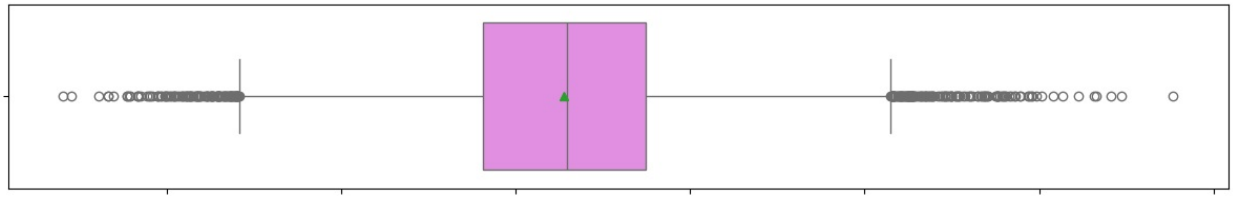

Count
800
600
400
200
0
−10
−5
0
5
10
V18

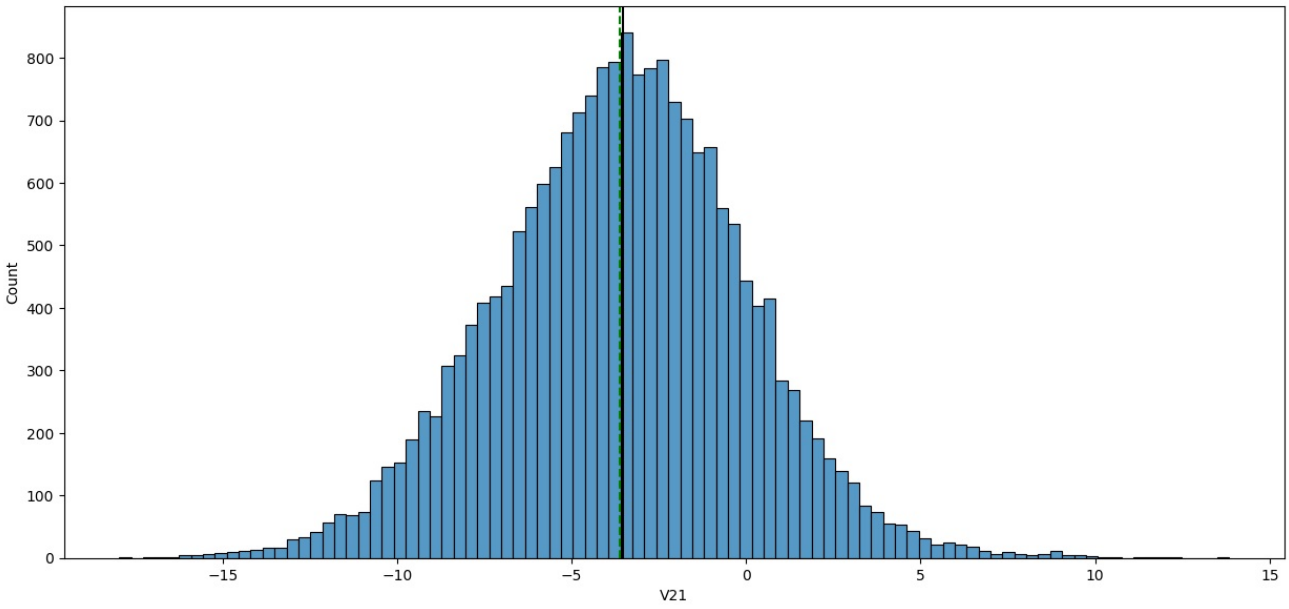


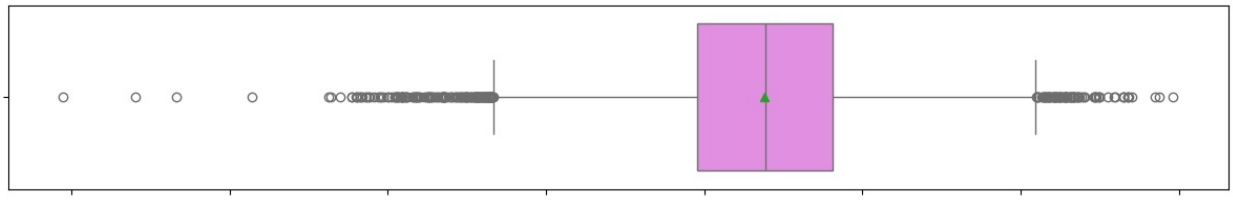

Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
V19

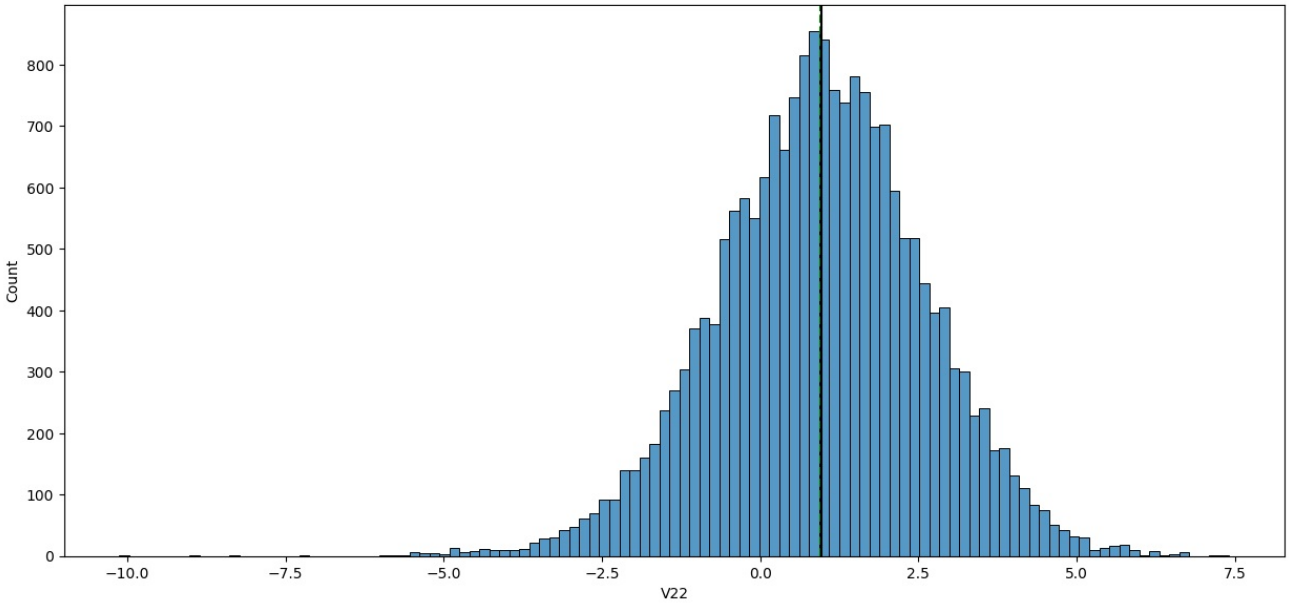


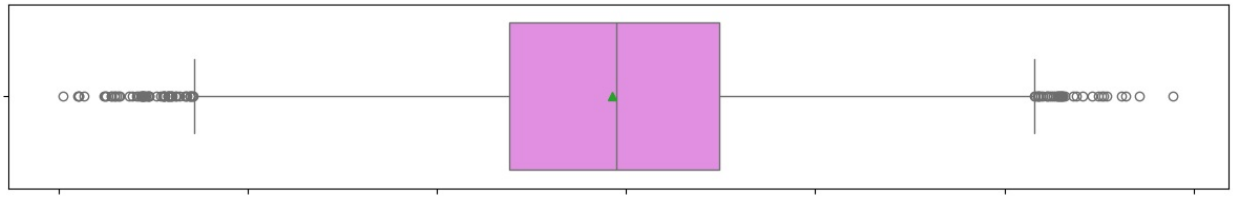

Count
V20

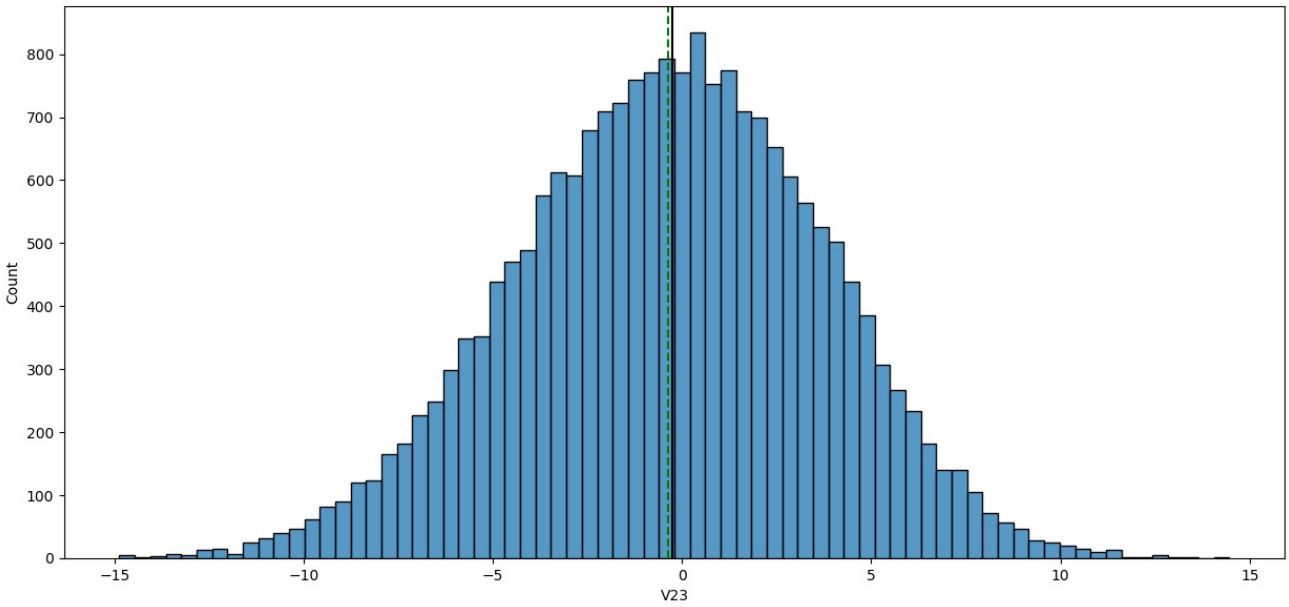


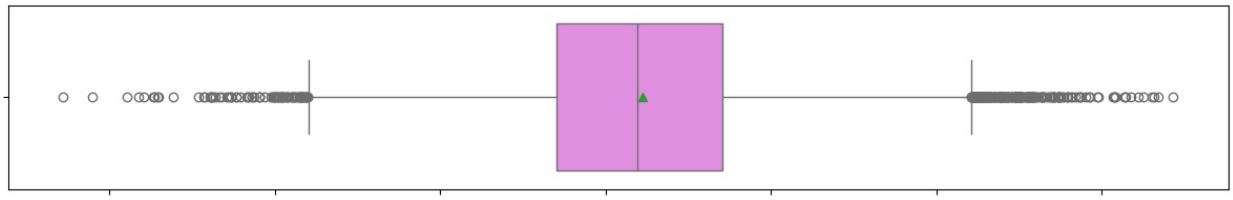

Count
V21

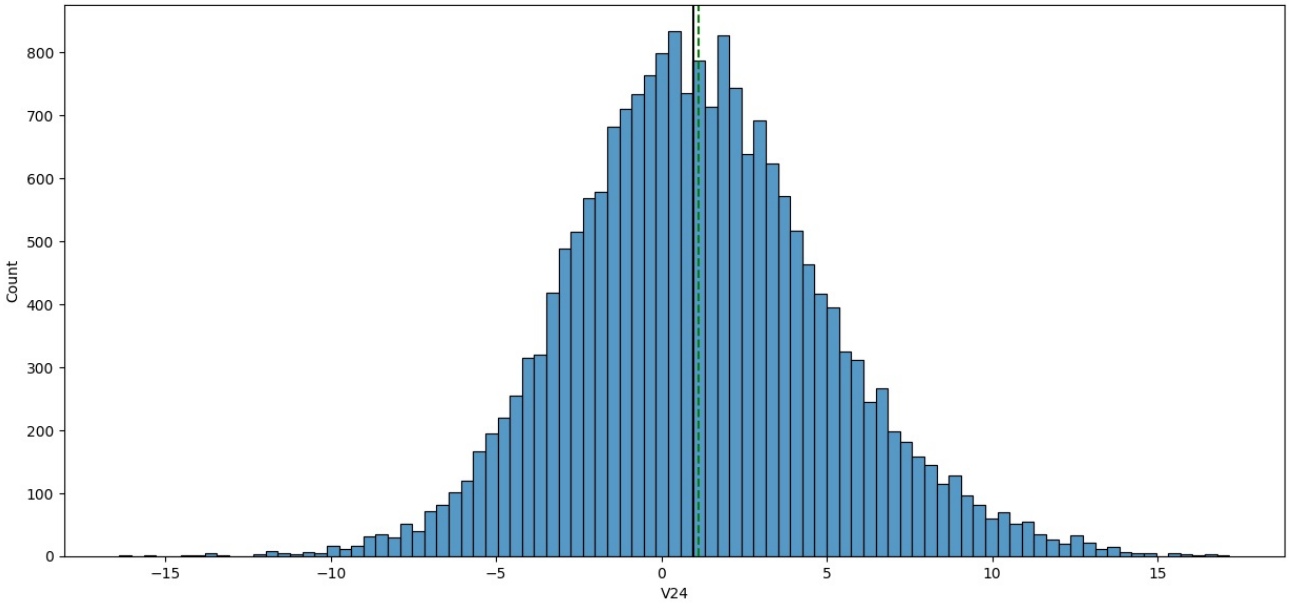


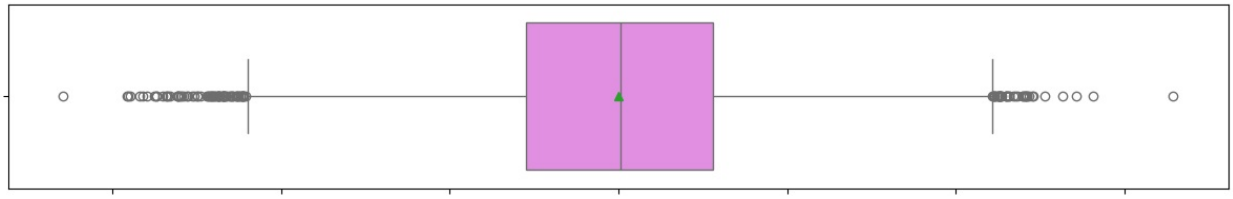

Count
800
700
600
500
400
300
200
100
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V22

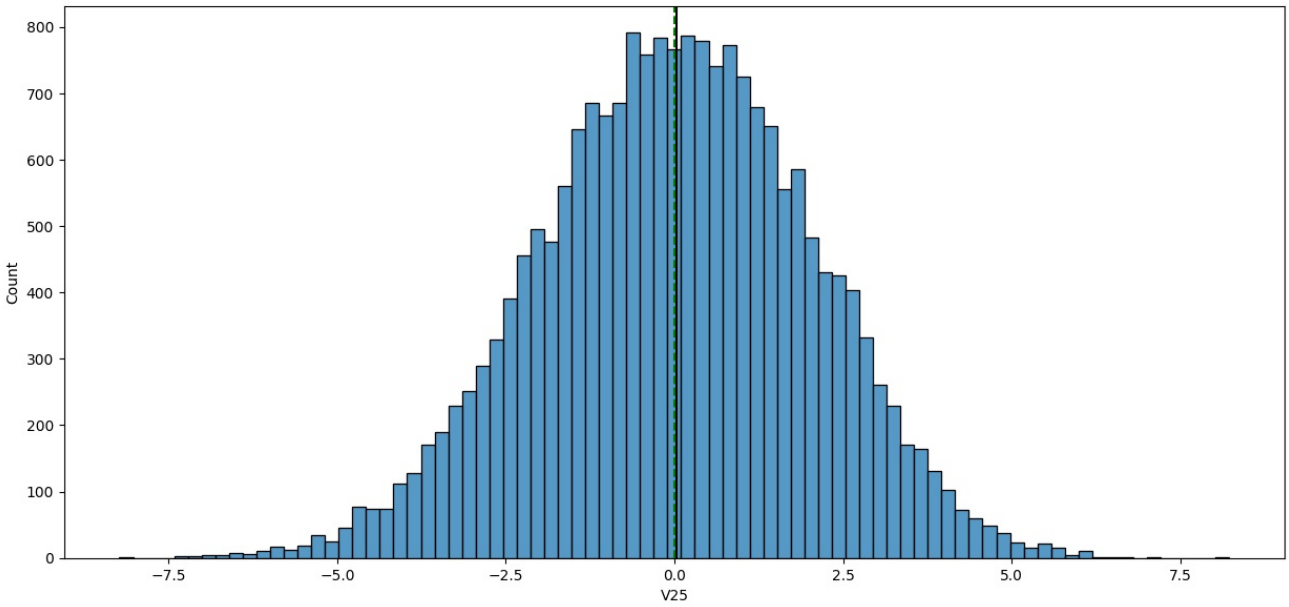


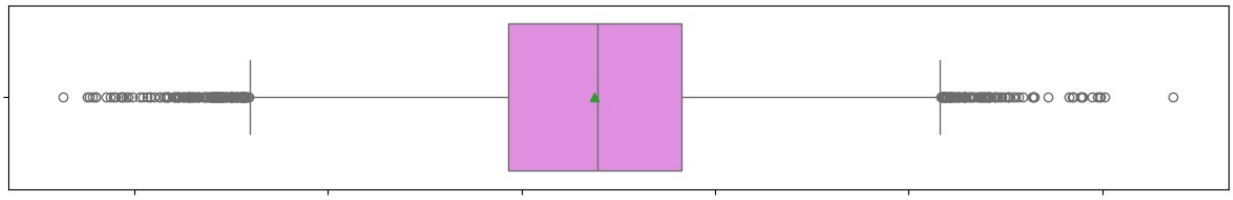

Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V23

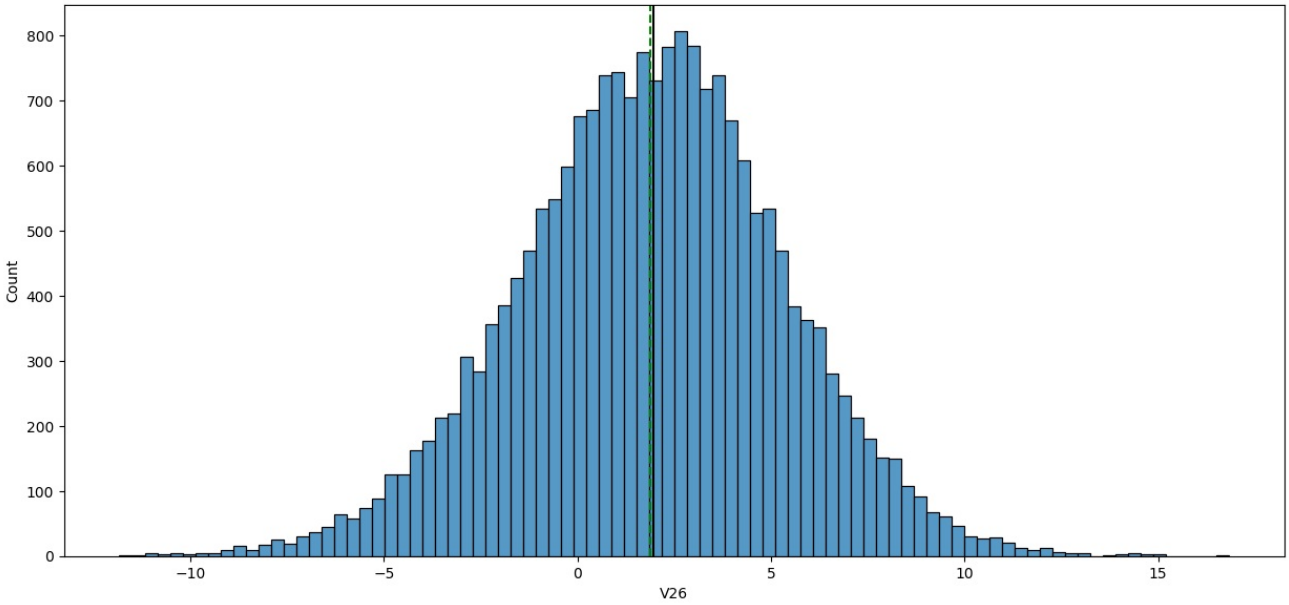


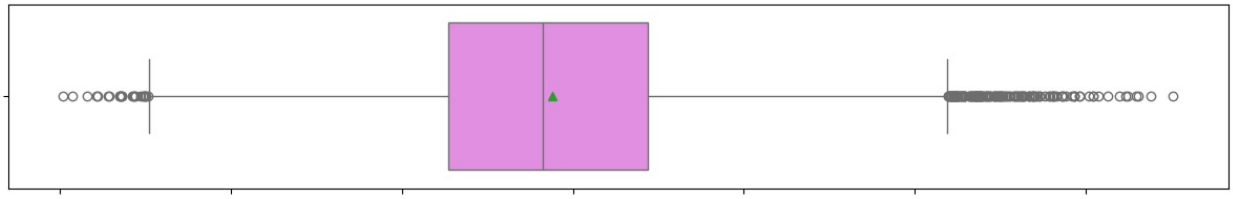

Count
V24

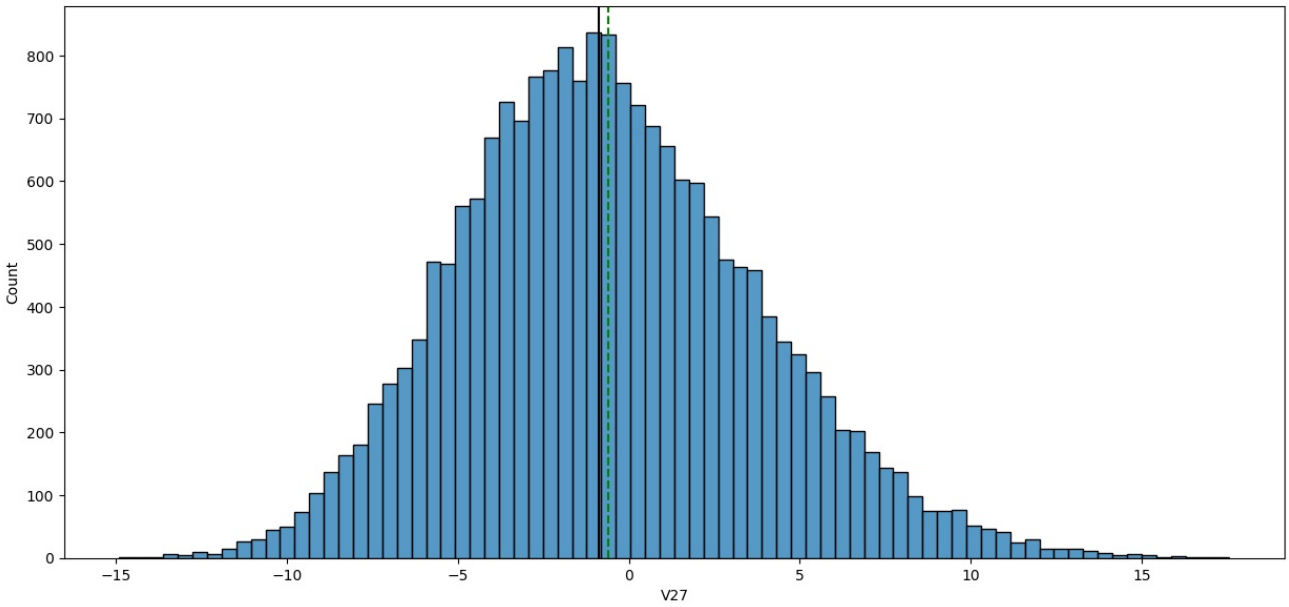


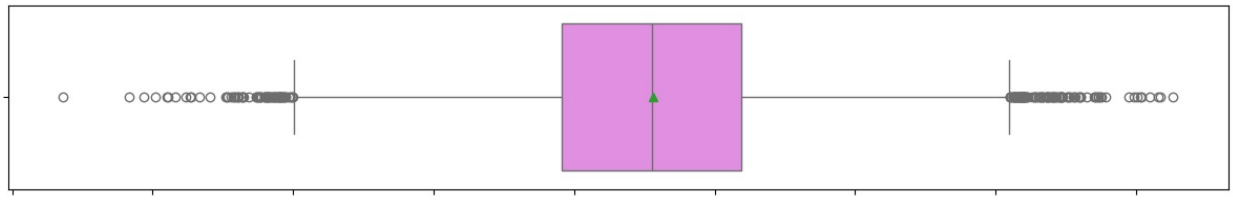

Count
V25

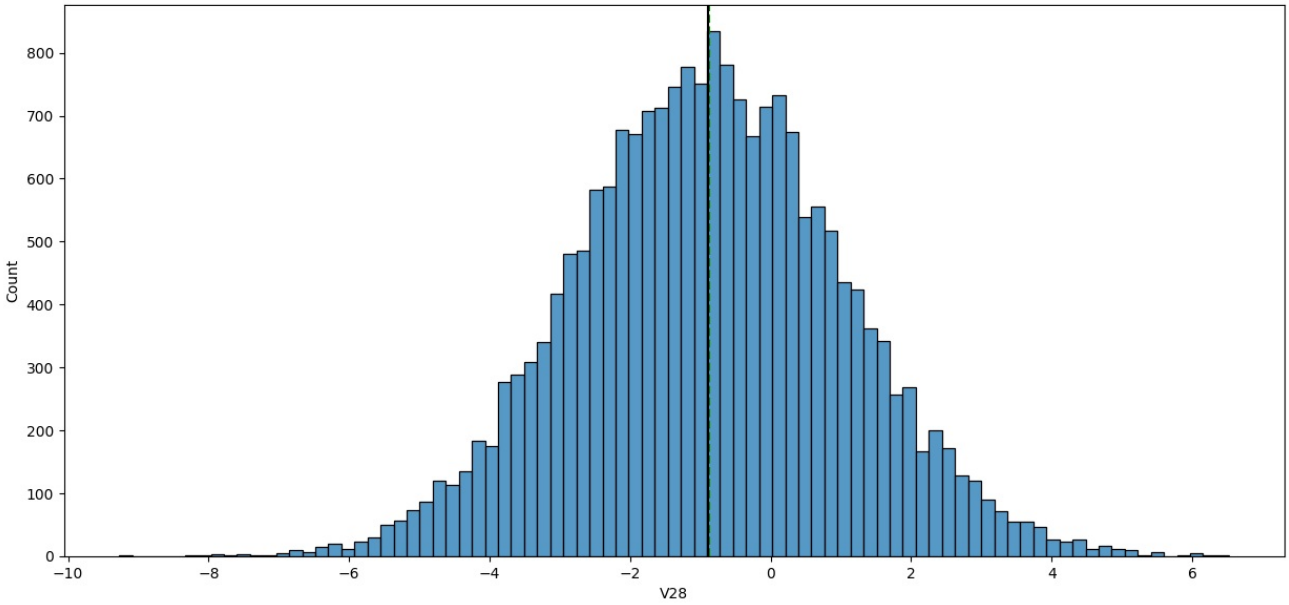


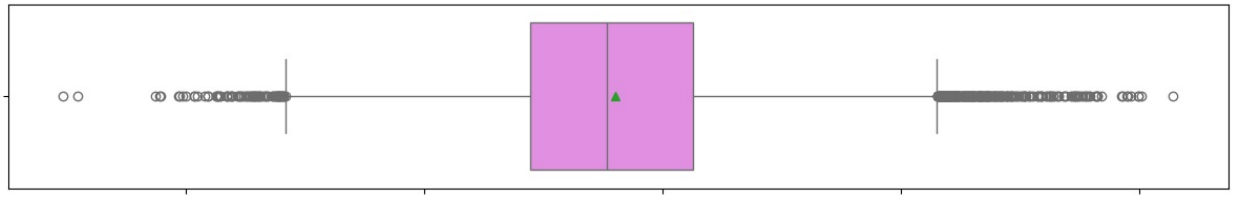

Count
V26

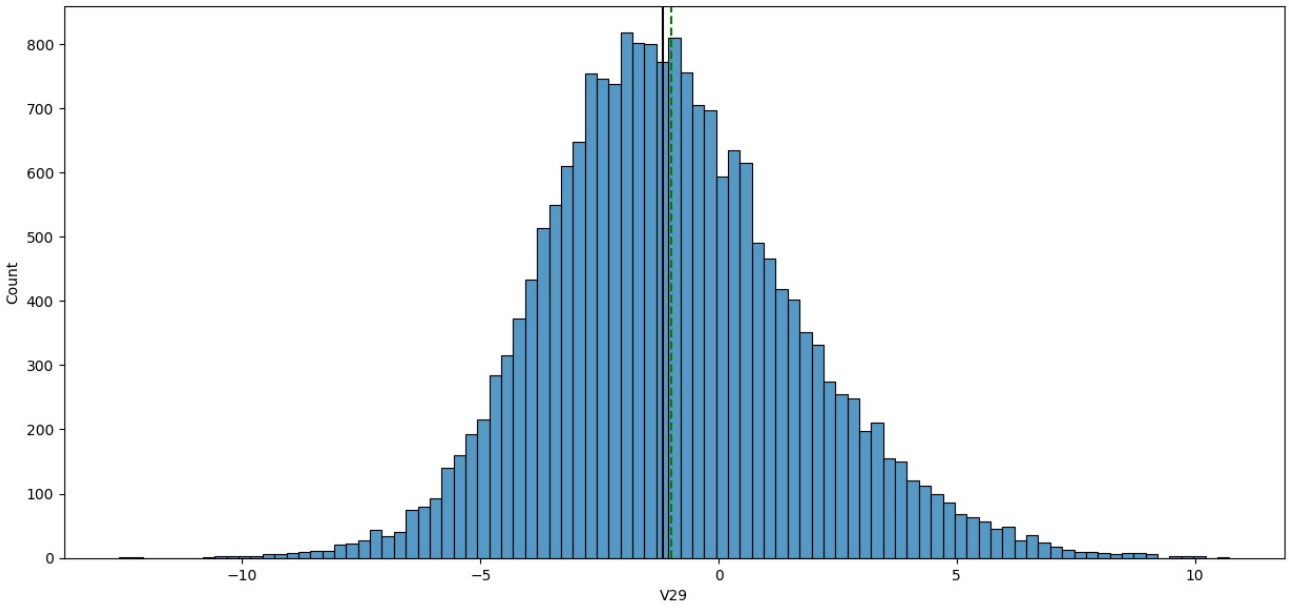


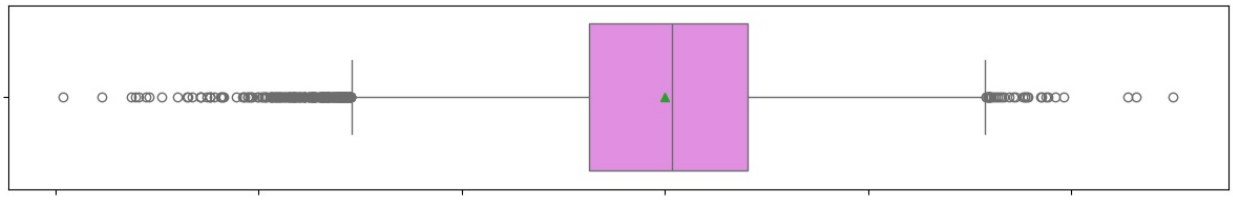

Count
V27

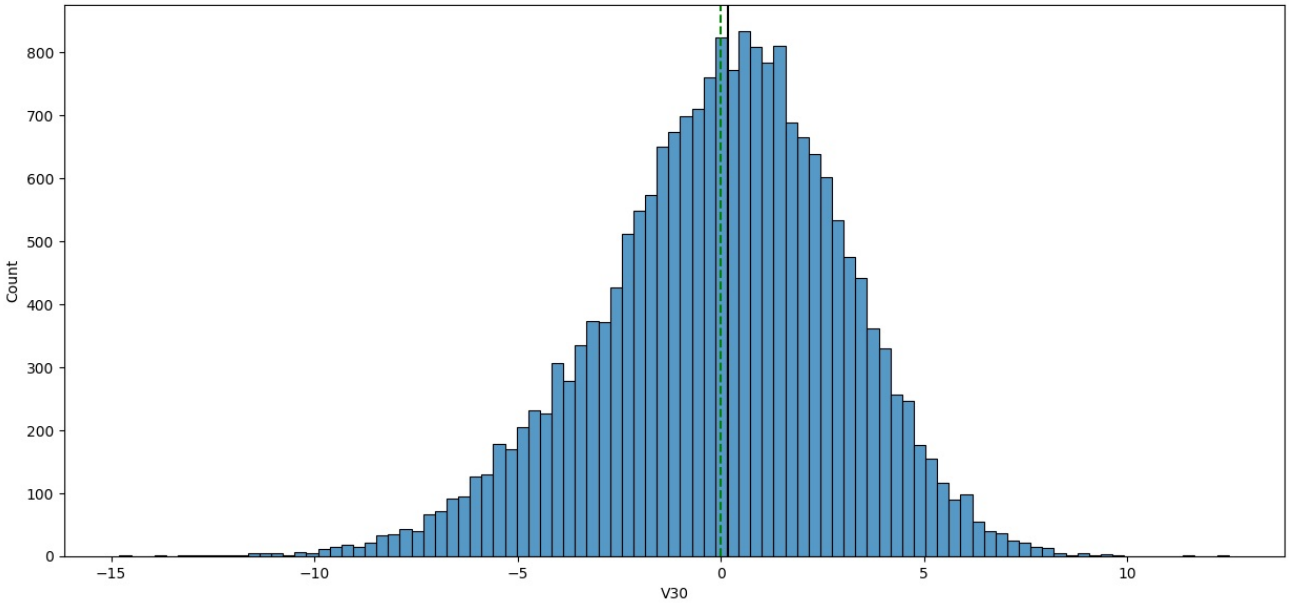


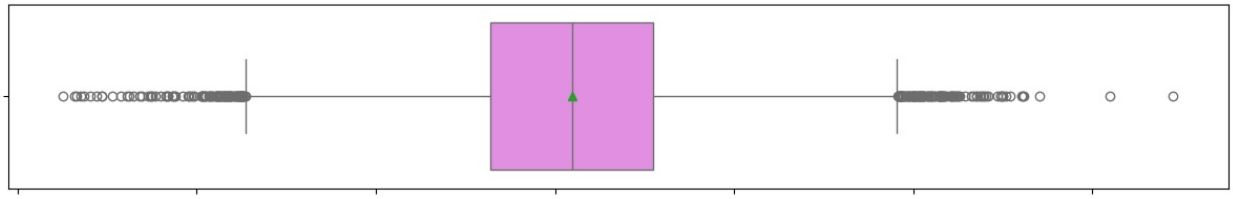

Count
V28

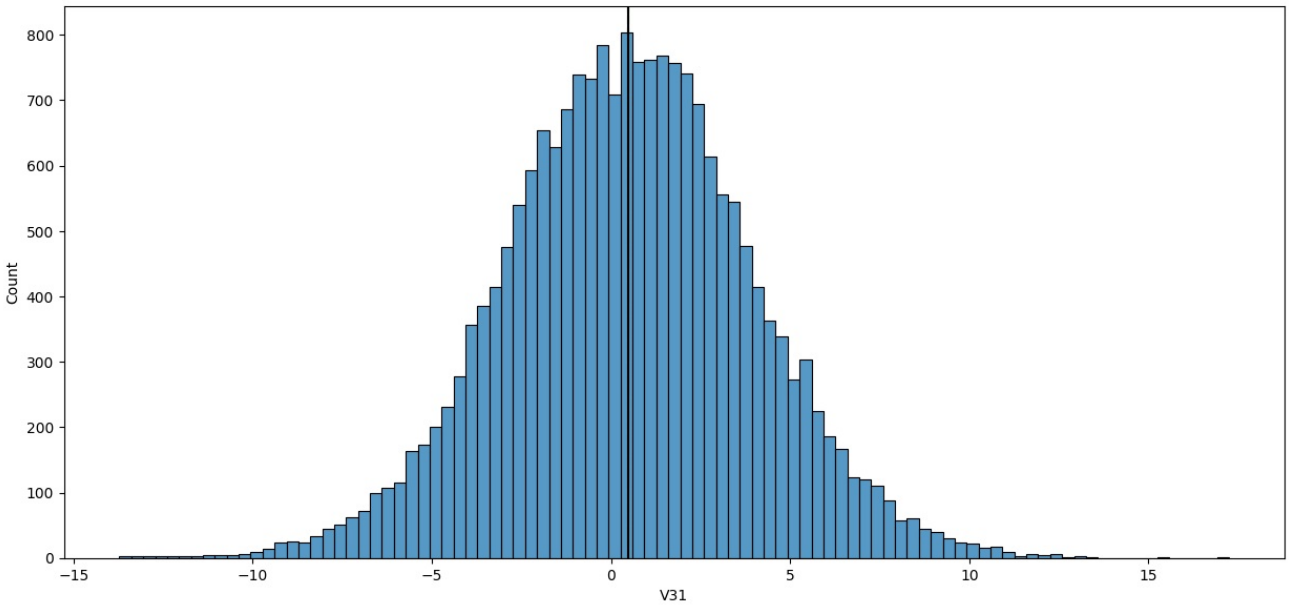


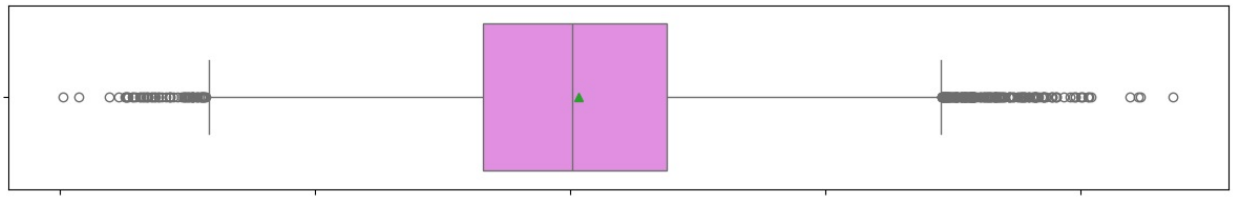

Count
V29

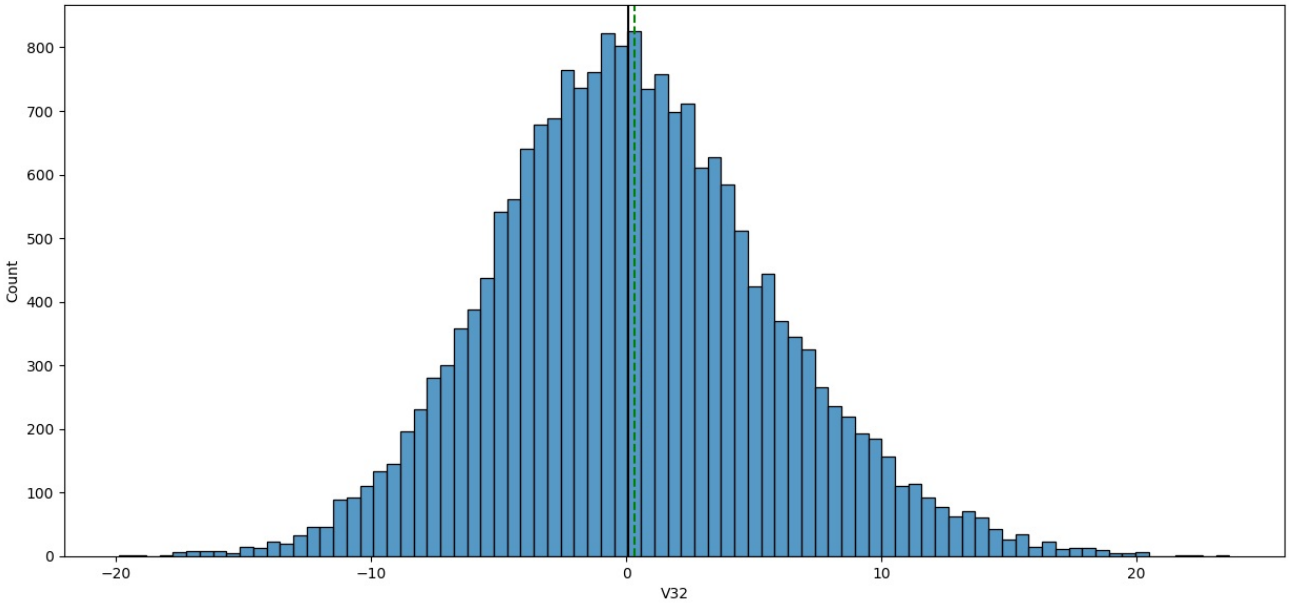


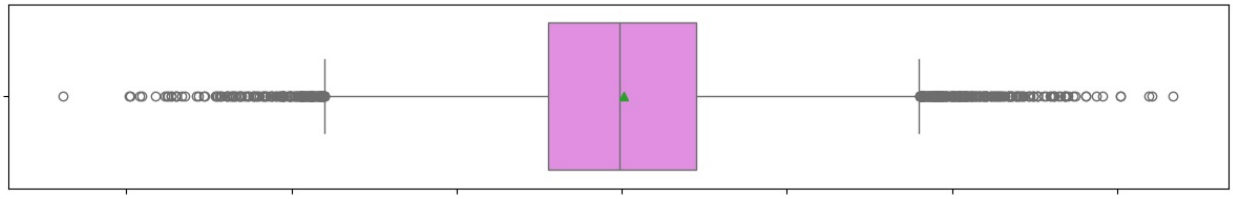

Count
V30

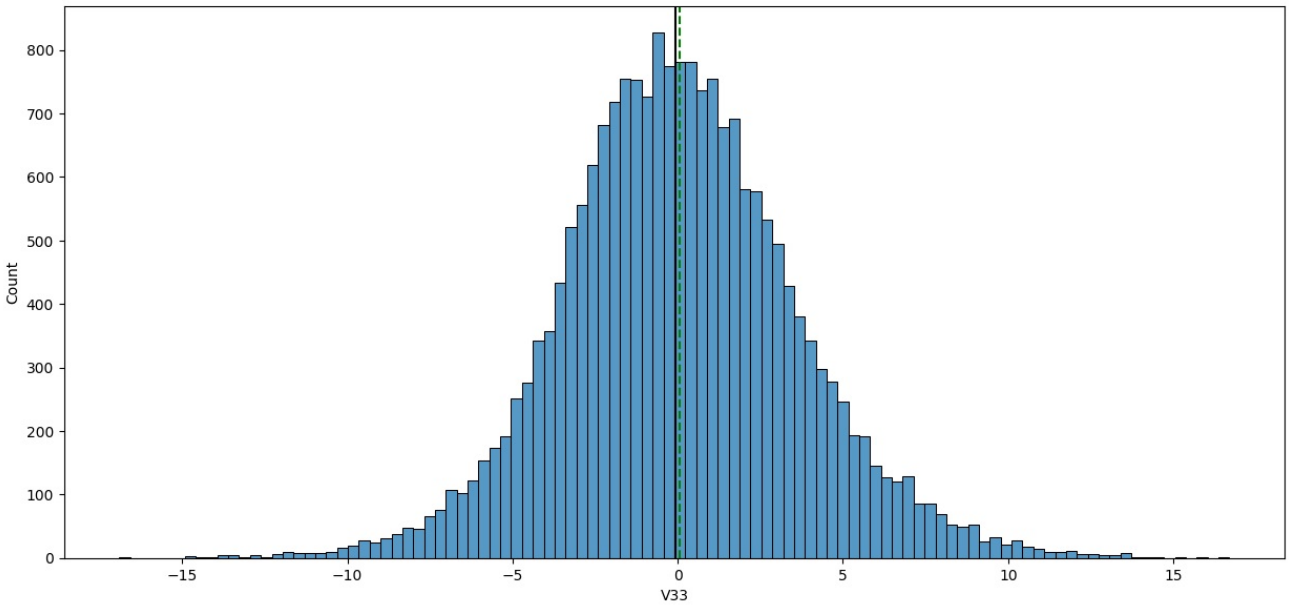


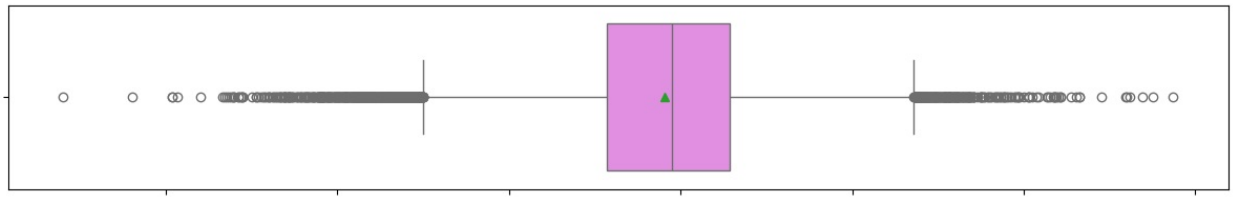

Count
V31

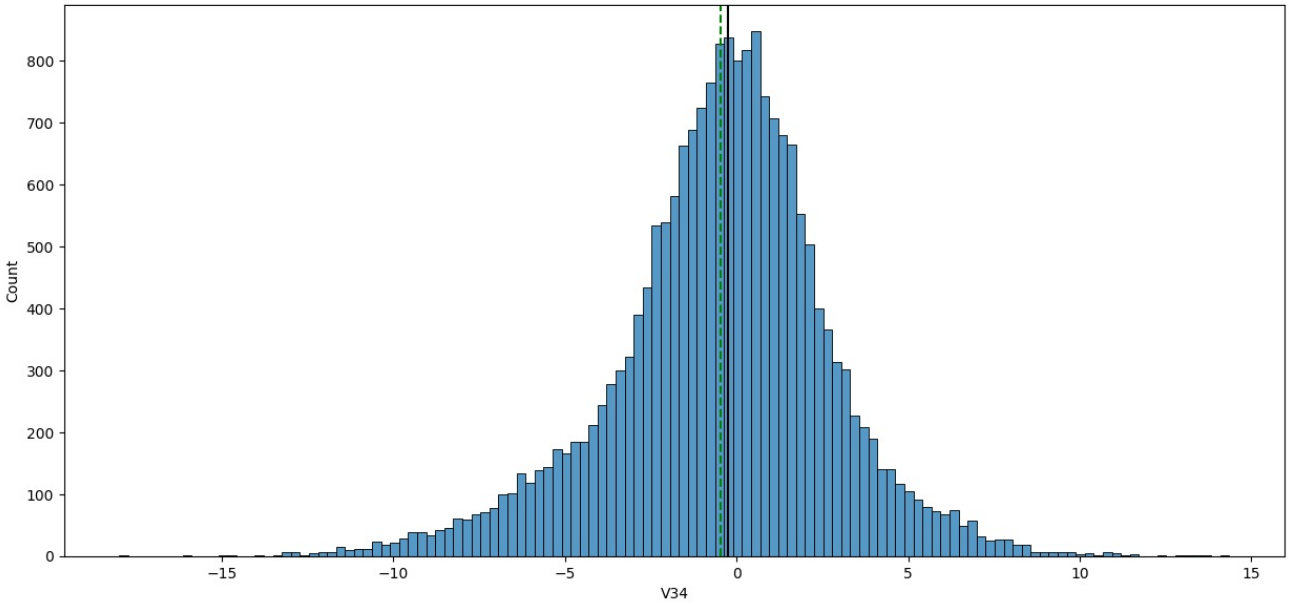


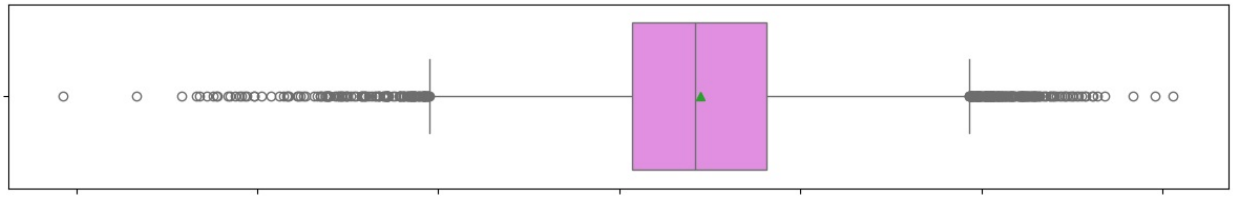

Count
V32

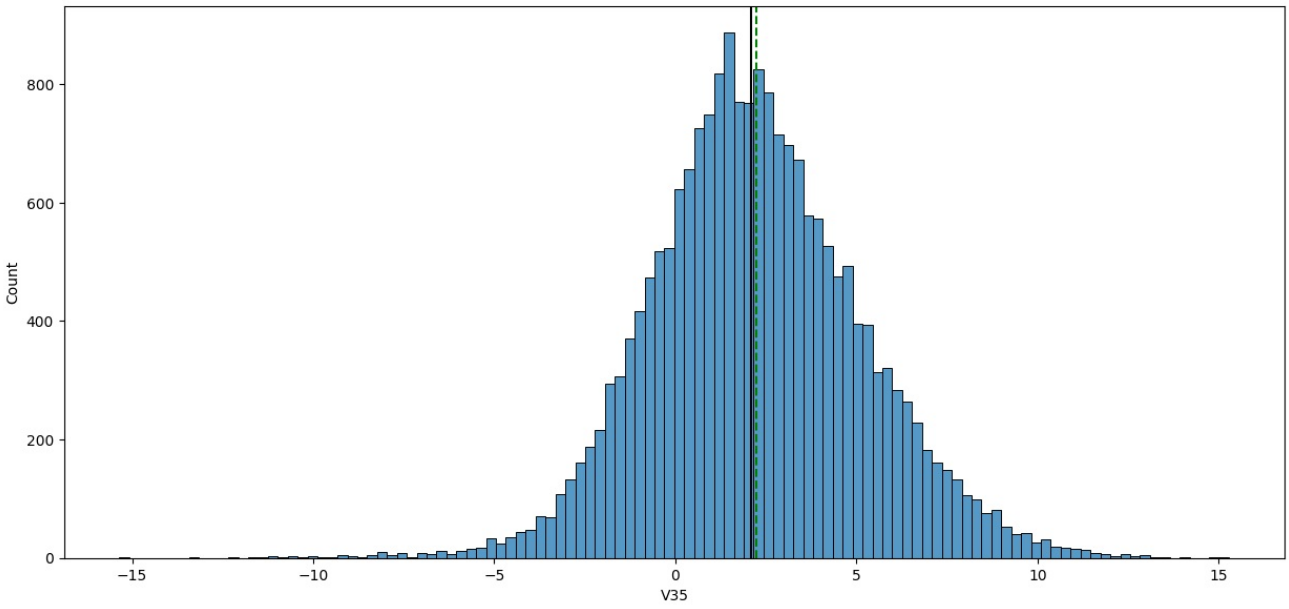


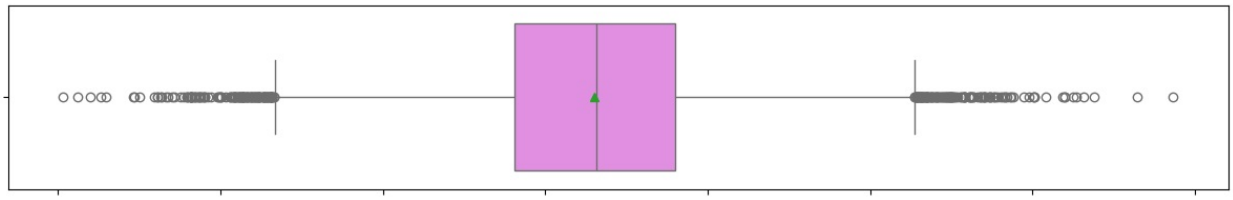

Count
V33

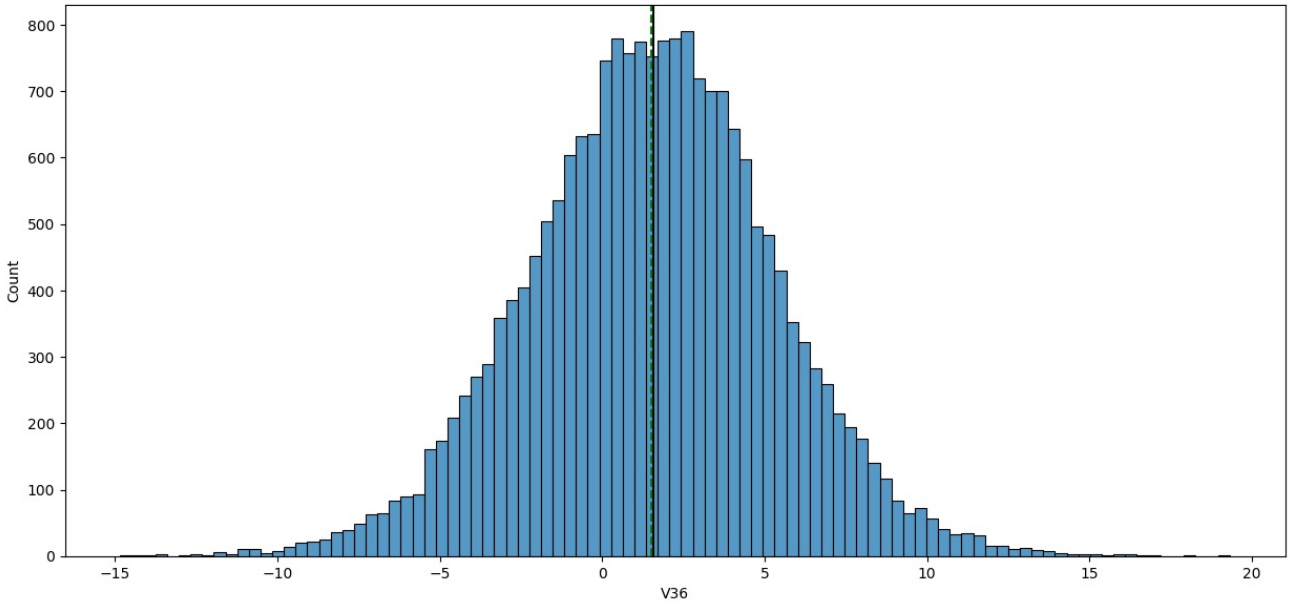


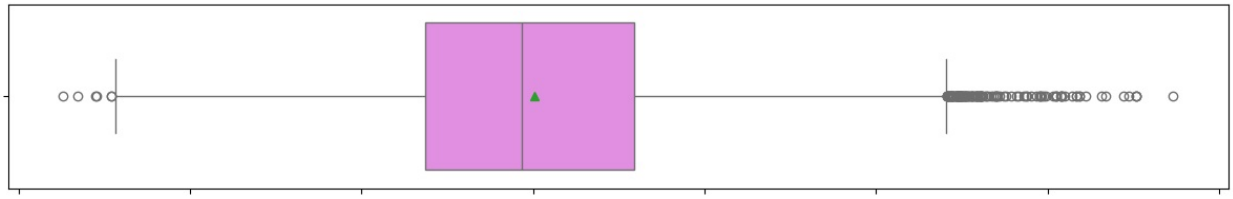

Count
V34

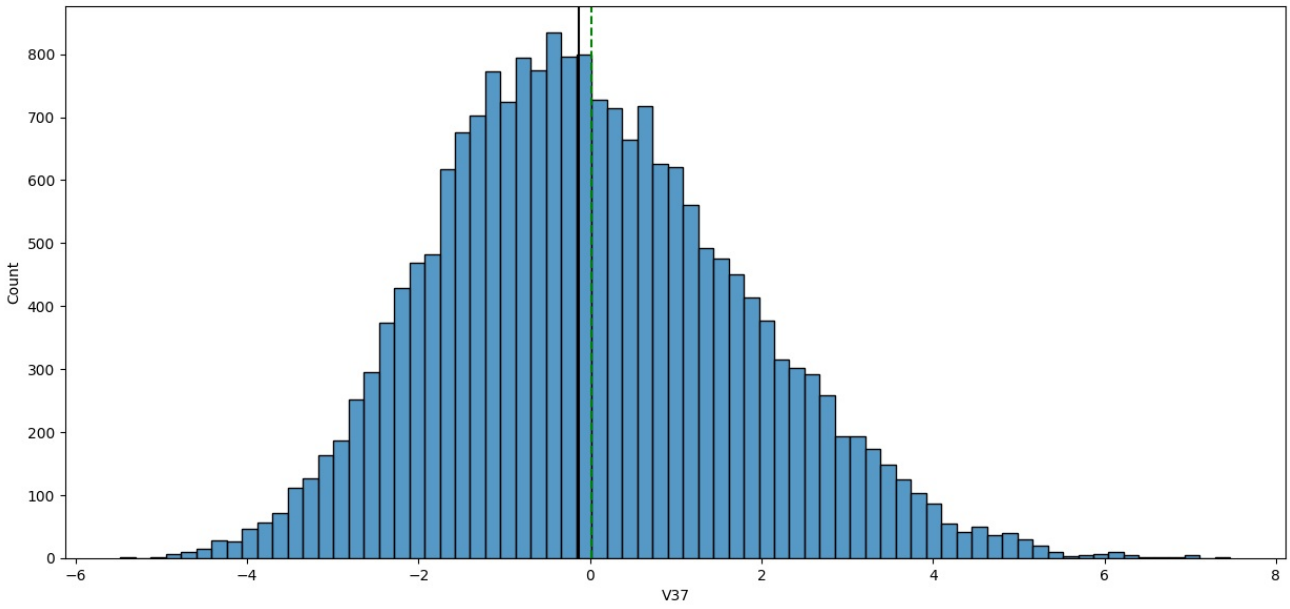


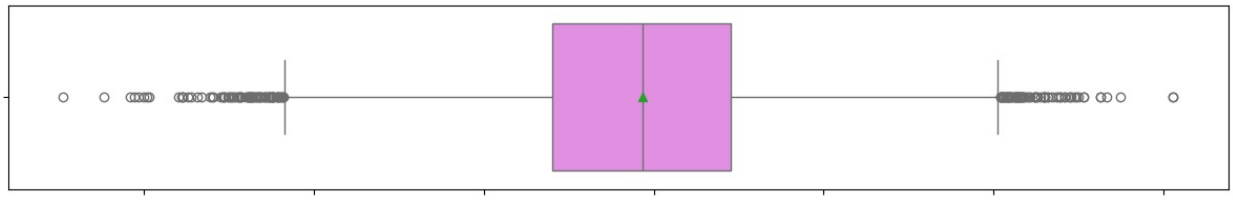

Count
V35

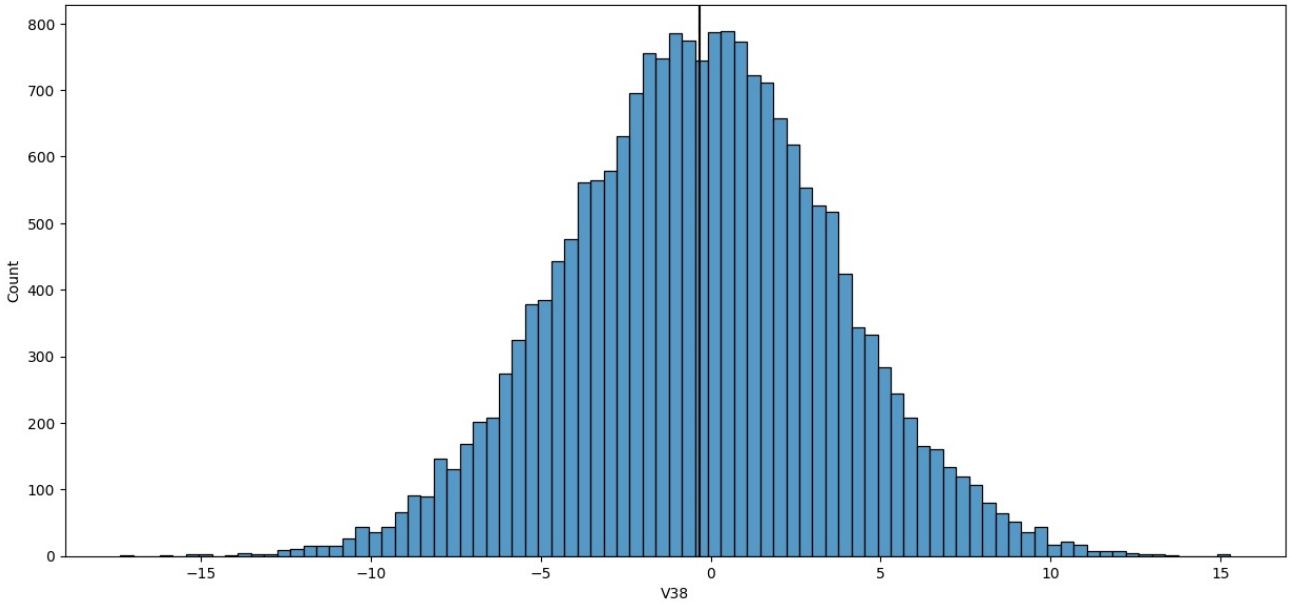


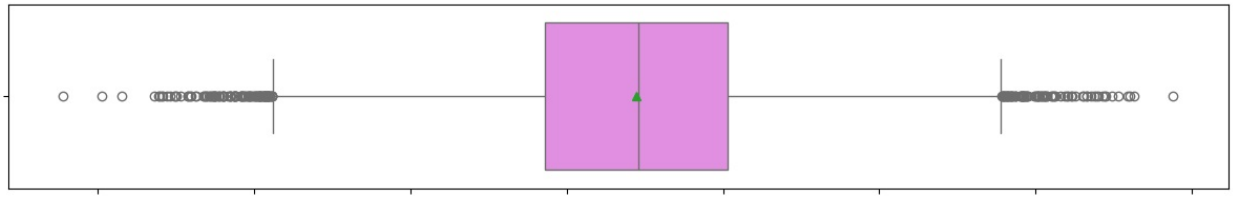

Count
V36

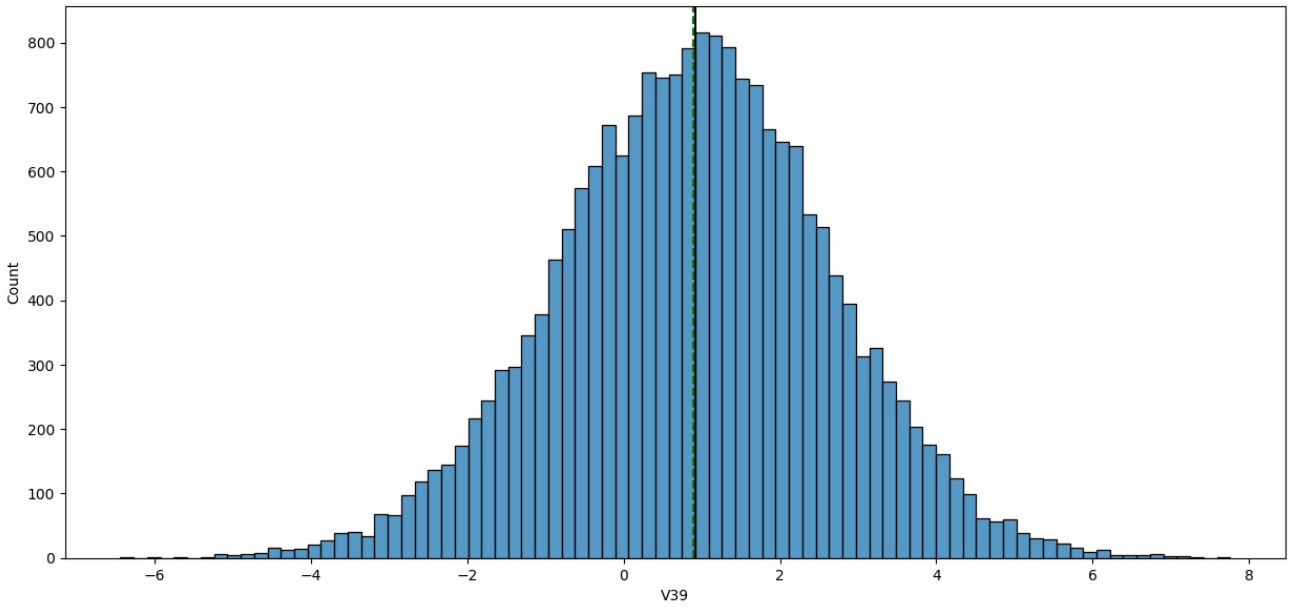


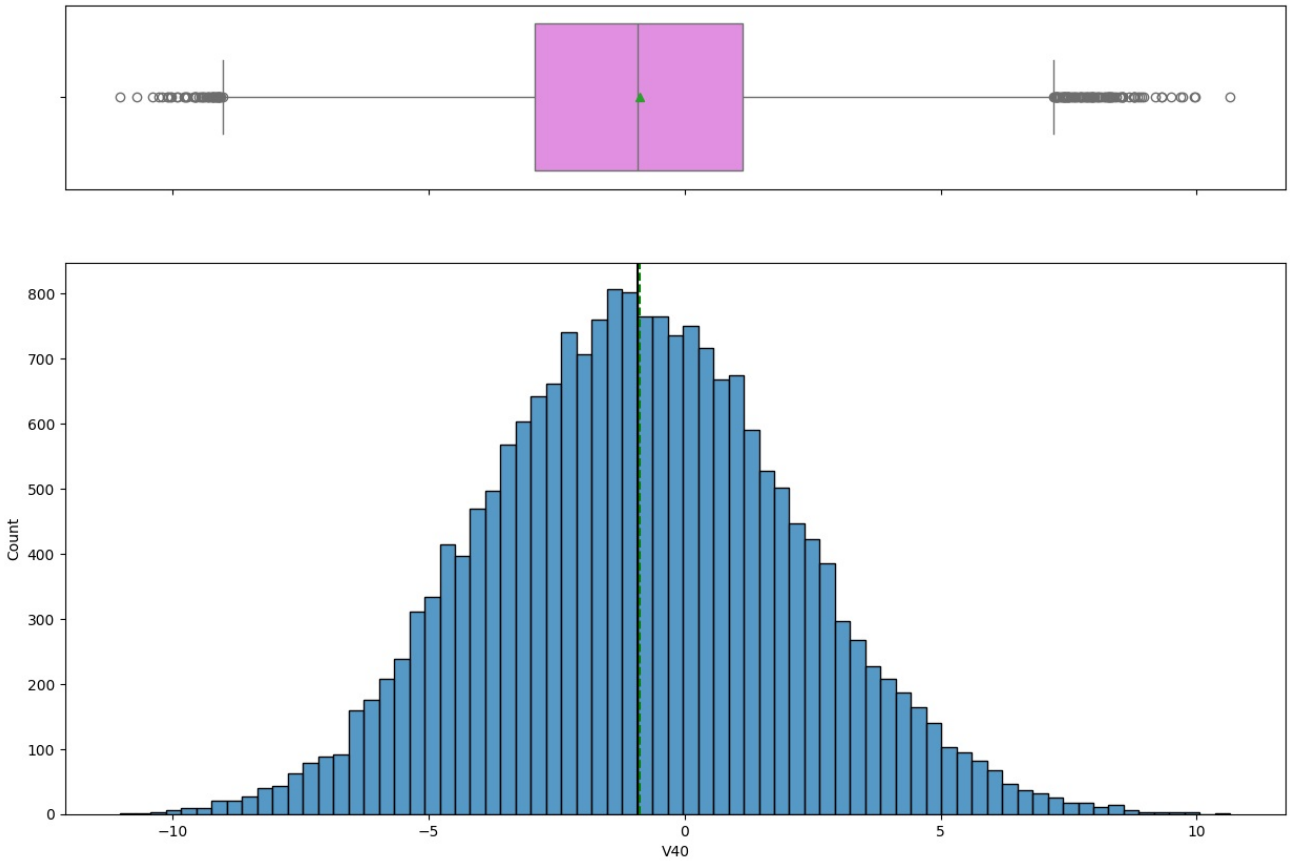

Count
V37




Count
V38




Count
V39

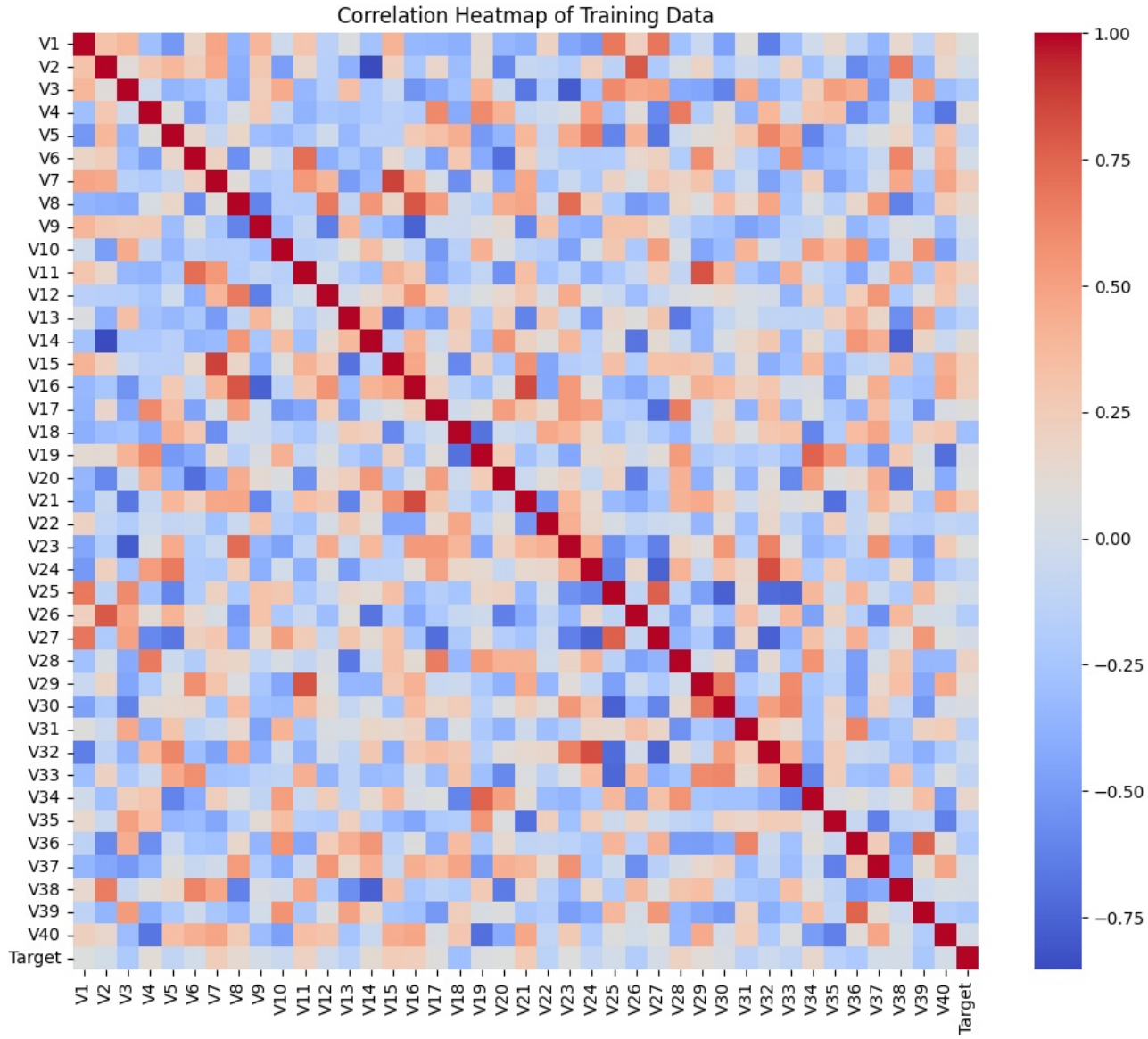


Univariate Analysis - All the features V1 through V40 range between minus 10(~) to plus 10(~). The count goes to 800 for the sensor values between close to 0.

Bivariate Analysis

In [22]:

```
# As a part of the bi-variate lets do a heat map.
plt.figure(figsize=(12, 10))
sns.heatmap(df_train.corr(), annot=False, cmap='coolwarm')
plt.title('Correlation Heatmap of Training Data')
plt.show()
```



Some features show moderate to strong corelations amongst themselves with the darkblue or dark red spots. But none of them have a strong coorelation with Target.

In [23]:

```
plt.figure(figsize=(6, 4))
sns.countplot(data=df_train, x='Target', palette='viridis')
plt.title('Distribution of Target Variable in Training Data')
plt.xlabel('Target (0: No Failure, 1: Failure)')
plt.ylabel('Count')
plt.show()
```

Data Preprocessing

In [24]:

```
df_train['Target'] = df_train['Target'].astype(float)
```

In [25]:

```
df_test['Target'] = df_test['Target'].astype(float)
```

In [26]:

```
df_train.isnull().sum()
df_test.isnull().sum()
```

Out[26]:

	0
V1	0
V2	0
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0
V10	0
V11	0
V12	0
V13	0
V14	0
V15	0
V16	0
V17	0
V18	0
V19	0
V20	0
V21	0
V22	0
V23	0
V24	0
V25	0
V26	0
V27	0
V28	0
V29	0
V30	0
V31	0
V32	0
V33	0
V34	0
V35	0
V36	0
V37	0
V38	0
V39	0
V40	0
Target	0

dtype: int64

In [27]:

```
# Dividing train data into X and y
X = df_train.drop(columns = ["Target"] , axis=1)
y = df_train["Target"]
```

In [28]:

```
# Splitting data into training and validation set:

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.8, random_state=42, stratify=y # Complete the code to define the test size
)
```

In [29]:

```
# Checking the number of rows and columns in the X_train data
X_train.shape
```

Out[29]:

```
(4000, 40)
```

In [30]:

```
# Checking the number of rows and columns in the X_val data
X_val.shape
```

Out[30]:

```
(16000, 40)
```

In [31]:

```
# Dividing test data into X_test and y_test
X_test = df_test.drop(columns = ['Target'] , axis= 1) # Complete the code to remove the target column
y_test = df_test["Target"] # Complete the code to select the target column
```

In [32]:

```
# Checking the number of rows and columns in the X_test data
X_test.shape
```

Out[32]:

```
(5000, 40)
```

In [33]:

```
# Checking that no column has missing values in train or test sets
print(X_train.isna().sum())
print("-" * 30)
print(X_val.isna().sum())
print("-" * 30)
print(X_test.isna().sum())
```

```
V1     0
V2     0
V3     0
V4     0
V5     0
V6     0
V7     0
V8     0
V9     0
V10    0
V11    0
V12    0
V13    0
V14    0
V15    0
V16    0
V17    0
V18    0
V19    0
V20    0
V21    0
V22    0
V23    0
V24    0
V25    0
V26    0
V27    0
V28    0
V29    0
V30    0
V31    0
V32    0
V33    0
V34    0
```

```
V35    0
V36    0
V37    0
V38    0
V39    0
V40    0
dtype: int64
------------------------------
V1     0
V2     0
V3     0
V4     0
V5     0
V6     0
V7     0
V8     0
V9     0
V10    0
V11    0
V12    0
V13    0
V14    0
V15    0
V16    0
V17    0
V18    0
V19    0
V20    0
V21    0
V22    0
V23    0
V24    0
V25    0
V26    0
V27    0
V28    0
V29    0
V30    0
V31    0
V32    0
V33    0
V34    0
V35    0
V36    0
V37    0
V38    0
V39    0
V40    0
dtype: int64
------------------------------
V1     0
V2     0
V3     0
V4     0
V5     0
V6     0
V7     0
V8     0
V9     0
V10    0
V11    0
V12    0
V13    0
V14    0
V15    0
V16    0
V17    0
V18    0
V19    0
V20    0
V21    0
V22    0
V23    0
V24    0
V25    0
V26    0
V27    0
V28    0
V29    0
V30    0
V31    0
V32    0
V33    0
```

```
V34    0
V35    0
V36    0
V37    0
V38    0
V39    0
V40    0
dtype: int64
```

In [34]:

```
y_train = y_train.to_numpy()
y_val = y_val.to_numpy()
y_test = y_test.to_numpy()
```

In [35]:

```
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 positio
n of the legend.
```

In [36]:

```
# defining a function to compute different metrics to check performance of a classification model built using sta
tsmodels
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='macro')  # to compute Recall
    precision = precision_score(target, pred, average='macro')  # to compute Precision
    f1 = f1_score(target, pred, average='macro')  # 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 model evaluation criterion with rationale

Initial Model Building (Model 0)

- Let's start with a neural network consisting of
  - just one hidden layer
  - activation function of ReLU
  - SGD as the optimizer

In [37]:

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

In [38]:

```
#Initializing the neural network.

model_0 = Sequential()
model_0.add(Dense(128,activation="relu",input_dim=X_train.shape[1]))
model_0.add(Dense(units=1,activation="sigmoid"))

# defining the batch size and # epochs upfront
epochs = 50
batch_size = 100
```

In [39]:

```
model_0.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,248
dense_1 (Dense)	(None, 1)	129

Total params: 5,377 (21.00 KB)

Trainable params: 5,377 (21.00 KB)

Non-trainable params: 0 (0.00 B)

Lets use the recall score here. We need the recall score because we need to avoid False negatives (FN) which are real failures in order to avoid replacement costs.

In [40]:

```
optimizer = tf.keras.optimizers.SGD()   # defining SGD as the optimizer to be used
model_0.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['Recall'])
#Already mentioned why we need to use Recall above. Also using binary entropy for the loss.
```

In [41]:

```
start = time.time()
history = model_0.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()
```

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - Recall: 0.0465 - loss: 0.2593 - val_Recall: 0.1239 - val_l
oss: 0.1851
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.1089 - loss: 0.1950 - val_Recall: 0.2466 - val_l
oss: 0.1528
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.2096 - loss: 0.1630 - val_Recall: 0.3468 - val_l
oss: 0.1347
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.4066 - loss: 0.1424 - val_Recall: 0.3885 - val_l
oss: 0.1229
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.3980 - loss: 0.1281 - val_Recall: 0.4459 - val_l
oss: 0.1145
Epoch 6/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.4402 - loss: 0.1165 - val_Recall: 0.4876 - val_l
oss: 0.1084
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.4506 - loss: 0.1133 - val_Recall: 0.5394 - val_l
oss: 0.1039
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.5267 - loss: 0.0962 - val_Recall: 0.5631 - val_l
oss: 0.1000
Epoch 9/50
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.5313 - loss: 0.0957 - val_Recall: 0.5698 - val_l
oss: 0.0967
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.5457 - loss: 0.0948 - val_Recall: 0.6025 - val_l
oss: 0.0942
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.6245 - loss: 0.0934 - val_Recall: 0.6070 - val_l
oss: 0.0921
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.6106 - loss: 0.0942 - val_Recall: 0.6216 - val_l
oss: 0.0902
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.6029 - loss: 0.0903 - val_Recall: 0.6340 - val_l
oss: 0.0886
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - Recall: 0.7046 - loss: 0.0803 - val_Recall: 0.6385 - val_l
oss: 0.0872
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - Recall: 0.6593 - loss: 0.0760 - val_Recall: 0.6588 - val_l
oss: 0.0860
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - Recall: 0.6361 - loss: 0.0976 - val_Recall: 0.6633 - val_l
oss: 0.0849
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - Recall: 0.7398 - loss: 0.0699 - val_Recall: 0.6667 - val_l
oss: 0.0838
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.7162 - loss: 0.0754 - val_Recall: 0.6745 - val_l
oss: 0.0828
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.7046 - loss: 0.0848 - val_Recall: 0.6802 - val_l
oss: 0.0820
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - Recall: 0.7069 - loss: 0.0715 - val_Recall: 0.6869 - val_l
oss: 0.0812
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - Recall: 0.7156 - loss: 0.0818 - val_Recall: 0.6881 - val_l
oss: 0.0805
Epoch 22/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.7209 - loss: 0.0780 - val_Recall: 0.6892 - val_l
oss: 0.0799
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - Recall: 0.7433 - loss: 0.0867 - val_Recall: 0.6993 - val_l
oss: 0.0792
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7441 - loss: 0.0640 - val_Recall: 0.6993 - val_l
oss: 0.0787
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7347 - loss: 0.0806 - val_Recall: 0.6982 - val_l
oss: 0.0782
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - Recall: 0.7555 - loss: 0.0717 - val_Recall: 0.7005 - val_l
oss: 0.0776
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - Recall: 0.7786 - loss: 0.0751 - val_Recall: 0.7050 - val_l
oss: 0.0772
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - Recall: 0.7292 - loss: 0.0763 - val_Recall: 0.7095 - val_l
oss: 0.0767
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.7449 - loss: 0.0722 - val_Recall: 0.7128 - val_l
oss: 0.0763
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7235 - loss: 0.0780 - val_Recall: 0.7083 - val_l
oss: 0.0760
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - Recall: 0.7816 - loss: 0.0708 - val_Recall: 0.7173 - val_l
oss: 0.0755
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.8019 - loss: 0.0607 - val_Recall: 0.7230 - val_l
oss: 0.0751
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.7569 - loss: 0.0568 - val_Recall: 0.7286 - val_l
oss: 0.0748
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.6987 - loss: 0.0719 - val_Recall: 0.7196 - val_l
oss: 0.0746
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7423 - loss: 0.0634 - val_Recall: 0.7264 - val_l
oss: 0.0742
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7595 - loss: 0.0592 - val_Recall: 0.7275 - val_l
oss: 0.0739
```

```
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7542 - loss: 0.0669 - val_Recall: 0.7286 - val_l
oss: 0.0736
Epoch 38/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.7424 - loss: 0.0751 - val_Recall: 0.7286 - val_l
oss: 0.0734
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.7630 - loss: 0.0647 - val_Recall: 0.7297 - val_l
oss: 0.0731
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7471 - loss: 0.0632 - val_Recall: 0.7320 - val_l
oss: 0.0729
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - Recall: 0.7314 - loss: 0.0716 - val_Recall: 0.7331 - val_l
oss: 0.0726
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - Recall: 0.7841 - loss: 0.0617 - val_Recall: 0.7331 - val_l
oss: 0.0724
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7599 - loss: 0.0667 - val_Recall: 0.7342 - val_l
oss: 0.0722
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7421 - loss: 0.0753 - val_Recall: 0.7354 - val_l
oss: 0.0720
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.7782 - loss: 0.0650 - val_Recall: 0.7365 - val_l
oss: 0.0718
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.7219 - loss: 0.0764 - val_Recall: 0.7365 - val_l
oss: 0.0716
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - Recall: 0.7258 - loss: 0.0739 - val_Recall: 0.7354 - val_l
oss: 0.0715
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - Recall: 0.7633 - loss: 0.0616 - val_Recall: 0.7365 - val_l
oss: 0.0713
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - Recall: 0.7171 - loss: 0.0595 - val_Recall: 0.7387 - val_lo
ss: 0.0711
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - Recall: 0.7962 - loss: 0.0657 - val_Recall: 0.7387 - val_l
oss: 0.0709
```

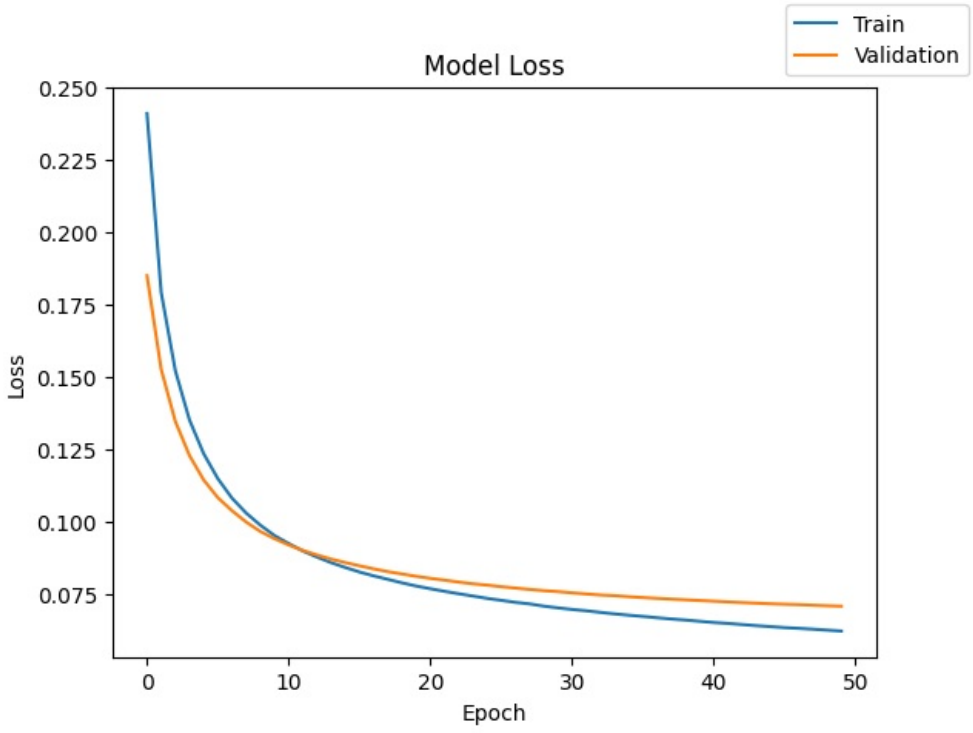
In [42]:

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

```
Time taken in seconds  48.18509912490845
```

In [43]:

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



In [44]:

```
model_0_train_perf = model_performance_classification(model_0, X_train, y_train)
model_0_train_perf
```

125/125 ━━━━━━━━━━━━━━━━━━━━ **0s** 1ms/step

Out[44]:

	Accuracy	Recall	Precision	F1 Score
0	0.98725	0.891495	0.985248	0.932735

In [45]:

```
model_0_val_perf = model_performance_classification(model_0,X_val,y_val)
model_0_val_perf
```

500/500 ━━━━━━━━━━━━━━━━━━━━ **1s** 1ms/step

Out[45]:

	Accuracy	Recall	Precision	F1 Score
0	0.983938	0.868542	0.974072	0.913878

In [46]:

```
y_train_pred_0 = model_0.predict(X_train)
y_val_pred_0 = model_0.predict(X_val)
```

125/125 ━━━━━━━━━━━━━━━━━━━━ **0s** 1ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ **1s** 1ms/step

In [47]:

```
print("Classification Report - Train data Model_0",end="\n\n")
cr_train_model_0 = classification_report(y_train,y_train_pred_0>0.5)
print(cr_train_model_0)
```

```
Classification Report - Train data Model_0

              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      3778
         1.0       0.98      0.78      0.87       222

    accuracy                           0.99      4000
   macro avg       0.99      0.89      0.93      4000
weighted avg       0.99      0.99      0.99      4000
```

In [48]:

```
print("Classification Report - Validation data Model_0",end="\n\n")
cr_val_model_0 = classification_report(y_val,y_val_pred_0>0.5)
print(cr_val_model_0)
```

```
Classification Report - Validation data Model_0

              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15112
         1.0       0.96      0.74      0.84       888

    accuracy                           0.98     16000
   macro avg       0.97      0.87      0.91     16000
weighted avg       0.98      0.98      0.98     16000
```

Model Performance Improvement

Model 1

In [49]:

```
# clears the current Keras session, resetting all layers and models previously created, freeing up memory and res
ources.
tf.keras.backend.clear_session()
epochs = 50
batch_size = 100
```

For Model 1 we will add another hidden layer with a tanH activation. Lets change the end to use softmax (instead of sigmoid) and continue to use SGD

In [50]:

```
#Initializing the neural network. Lets Add 2 hidden layers here with relu and tanh as the activiations respective
ly.
model_1 = Sequential()
model_1.add(Dense(128,activation="relu",input_dim = X_train.shape[1]))
model_1.add(Dense(64,activation="tanh"))
model_1.add(Dense(units=1,activation = 'softmax'))

optimizer = tf.keras.optimizers.SGD()
model_1.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['Recall'])
model_1.summary()

start = time.time()
history = model_1.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()

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

plot(history,'loss')
model_1_train_perf = model_performance_classification(model_1, X_train, y_train)
print(model_1_train_perf)

model_1_val_perf = model_performance_classification(model_1,X_val,y_val)
print(model_1_val_perf)

y_train_pred_1 = model_1.predict(X_train)
y_val_pred_1 = model_1.predict(X_val)

print("Classification Report - Train data Model_1",end="\n\n")
cr_train_model_1 = classification_report(y_train,y_train_pred_0>0.5)
print(cr_train_model_1)

print("Classification Report - Validation data Model_1",end="\n\n")
cr_val_model_1 = classification_report(y_val,y_val_pred_0>0.5)
print(cr_val_model_1)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,248
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 1)	65

Total params: 13,569 (53.00 KB)

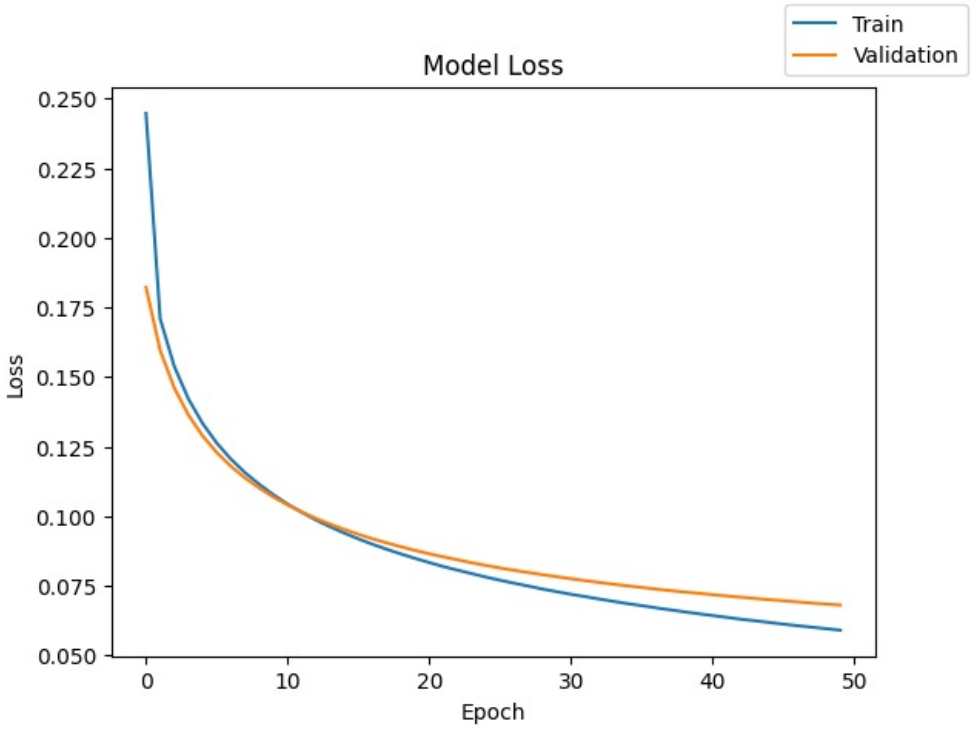
Trainable params: 13,569 (53.00 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - Recall: 1.0000 - loss: 0.3138 - val_Recall: 1.0000 - val_l
oss: 0.1822
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.1768 - val_Recall: 1.0000 - val_l
oss: 0.1596
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.1541 - val_Recall: 1.0000 - val_l
oss: 0.1460
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.1498 - val_Recall: 1.0000 - val_l
oss: 0.1363
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - Recall: 1.0000 - loss: 0.1385 - val_Recall: 1.0000 - val_l
oss: 0.1288
Epoch 6/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.1178 - val_Recall: 1.0000 - val_l
oss: 0.1229
```

```
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 1.0000 - loss: 0.1169 - val_Recall: 1.0000 - val_l
oss: 0.1179
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - Recall: 1.0000 - loss: 0.1143 - val_Recall: 1.0000 - val_l
oss: 0.1137
Epoch 9/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 1.0000 - loss: 0.1139 - val_Recall: 1.0000 - val_l
oss: 0.1101
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 1.0000 - loss: 0.1154 - val_Recall: 1.0000 - val_l
oss: 0.1069
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0978 - val_Recall: 1.0000 - val_l
oss: 0.1040
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.1051 - val_Recall: 1.0000 - val_l
oss: 0.1015
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - Recall: 1.0000 - loss: 0.0987 - val_Recall: 1.0000 - val_l
oss: 0.0992
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.1100 - val_Recall: 1.0000 - val_l
oss: 0.0971
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0971 - val_Recall: 1.0000 - val_l
oss: 0.0953
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0892 - val_Recall: 1.0000 - val_l
oss: 0.0935
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - Recall: 1.0000 - loss: 0.0825 - val_Recall: 1.0000 - val_l
oss: 0.0919
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0824 - val_Recall: 1.0000 - val_l
oss: 0.0904
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0921 - val_Recall: 1.0000 - val_l
oss: 0.0890
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0806 - val_Recall: 1.0000 - val_l
oss: 0.0877
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0857 - val_Recall: 1.0000 - val_l
oss: 0.0865
Epoch 22/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0857 - val_Recall: 1.0000 - val_l
oss: 0.0854
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0851 - val_Recall: 1.0000 - val_l
oss: 0.0843
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 1.0000 - loss: 0.0814 - val_Recall: 1.0000 - val_l
oss: 0.0833
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0724 - val_Recall: 1.0000 - val_l
oss: 0.0823
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0762 - val_Recall: 1.0000 - val_l
oss: 0.0814
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0787 - val_Recall: 1.0000 - val_l
oss: 0.0806
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 1.0000 - loss: 0.0789 - val_Recall: 1.0000 - val_l
oss: 0.0798
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 1.0000 - loss: 0.0763 - val_Recall: 1.0000 - val_l
oss: 0.0790
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 1.0000 - loss: 0.0728 - val_Recall: 1.0000 - val_l
oss: 0.0783
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - Recall: 1.0000 - loss: 0.0671 - val_Recall: 1.0000 - val_l
oss: 0.0775
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 1.0000 - loss: 0.0739 - val_Recall: 1.0000 - val_l
oss: 0.0769
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0663 - val_Recall: 1.0000 - val_l
oss: 0.0762
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0645 - val_Recall: 1.0000 - val_l
```

```
oss: 0.0756
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0721 - val_Recall: 1.0000 - val_l
oss: 0.0750
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0674 - val_Recall: 1.0000 - val_l
oss: 0.0744
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0593 - val_Recall: 1.0000 - val_l
oss: 0.0738
Epoch 38/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0593 - val_Recall: 1.0000 - val_l
oss: 0.0733
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0626 - val_Recall: 1.0000 - val_l
oss: 0.0728
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0677 - val_Recall: 1.0000 - val_l
oss: 0.0723
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - Recall: 1.0000 - loss: 0.0691 - val_Recall: 1.0000 - val_l
oss: 0.0718
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0572 - val_Recall: 1.0000 - val_l
oss: 0.0714
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0683 - val_Recall: 1.0000 - val_l
oss: 0.0709
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0645 - val_Recall: 1.0000 - val_l
oss: 0.0705
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - Recall: 1.0000 - loss: 0.0580 - val_Recall: 1.0000 - val_l
oss: 0.0700
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0688 - val_Recall: 1.0000 - val_l
oss: 0.0696
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 1.0000 - loss: 0.0682 - val_Recall: 1.0000 - val_l
oss: 0.0692
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0682 - val_Recall: 1.0000 - val_l
oss: 0.0688
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 1.0000 - loss: 0.0654 - val_Recall: 1.0000 - val_l
oss: 0.0685
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 1.0000 - loss: 0.0594 - val_Recall: 1.0000 - val_l
oss: 0.0681
Time taken in seconds  30.593169927597046
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
   Accuracy  Recall  Precision  F1 Score
0    0.0555     0.5    0.02775  0.052582
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
   Accuracy  Recall  Precision  F1 Score
0    0.0555     0.5    0.02775  0.052582
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Classification Report - Train data Model_1

              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      3778
         1.0       0.98      0.78      0.87       222

    accuracy                           0.99      4000
   macro avg       0.99      0.89      0.93      4000
weighted avg       0.99      0.99      0.99      4000

Classification Report - Validation data Model_1

              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15112
         1.0       0.96      0.74      0.84       888

    accuracy                           0.98     16000
   macro avg       0.97      0.87      0.91     16000
weighted avg       0.98      0.98      0.98     16000
```



Model 2

In [51]:

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

#Initializing the neural network
#adding 3 hidden layers with a dropout. Lets use SGD as the optimizer.
model_2 = Sequential()
model_2.add(Dense(128,activation="relu",input_dim = X_train.shape[1]))
model_2.add(Dropout(0.4))
model_2.add(Dense(64,activation="tanh"))
model_2.add(Dense(64,activation = "relu"))
model_2.add(Dense(units=1,activation="sigmoid"))

optimizer = tf.keras.optimizers.SGD()
model_2.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['Recall'])
model_2.summary()
epochs = 50
batch_size = 100

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

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

plot(history,'loss')
model_2_train_perf = model_performance_classification(model_2, X_train, y_train)
model_2_train_perf

model_2_val_perf = model_performance_classification(model_2,X_val,y_val)
model_2_val_perf

y_train_pred_2 = model_2.predict(X_train)
y_val_pred_2 = model_2.predict(X_val)

print("\nClassification Report - Train data Model_2",end="\n")
cr_train_model_2 = classification_report(y_train,y_train_pred_0>0.5)
print(cr_train_model_2)

print("\nClassification Report - Validation data Model_2",end="\n")
cr_val_model_2 = classification_report(y_val,y_val_pred_0>0.5)
print(cr_val_model_2)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,248
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 64)	4,160
dense_3 (Dense)	(None, 1)	65

Total params: 17,729 (69.25 KB)

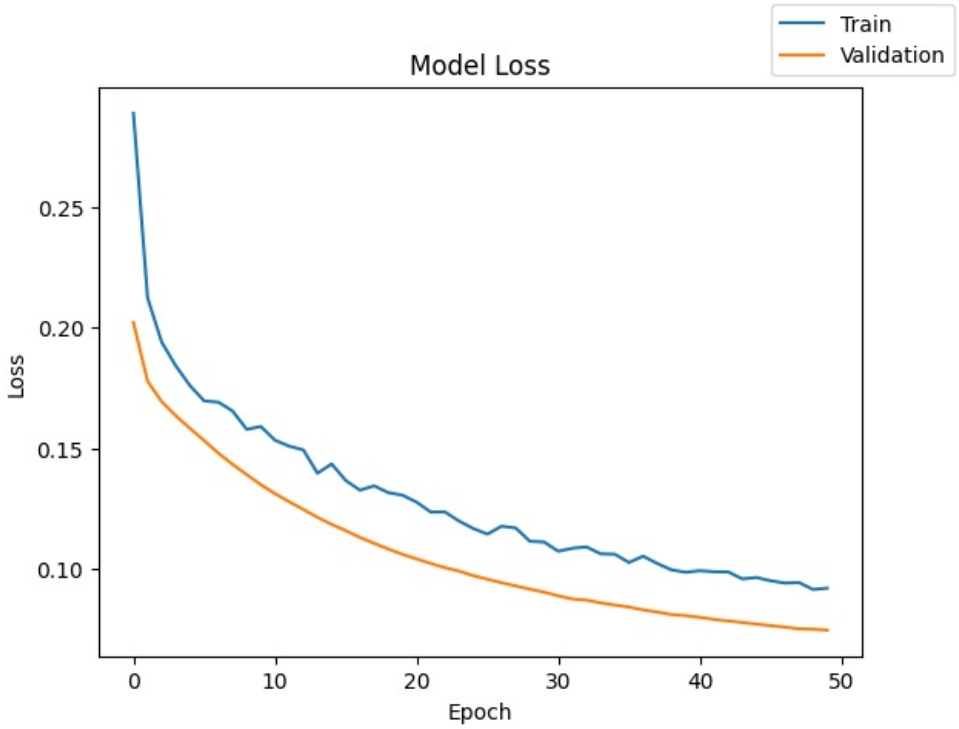
Trainable params: 17,729 (69.25 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - Recall: 0.0101 - loss: 0.3344 - val_Recall: 0.0000e+00 - v
al_loss: 0.2022
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.0035 - loss: 0.2206 - val_Recall: 0.0000e+00 - v
al_loss: 0.1778
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0000e+00 - loss: 0.2047 - val_Recall: 0.0000e+00
- val_loss: 0.1694
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0000e+00 - loss: 0.1867 - val_Recall: 0.0034 - v
al_loss: 0.1635
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0147 - loss: 0.1819 - val_Recall: 0.0045 - val_l
oss: 0.1582
Epoch 6/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.0159 - loss: 0.1686 - val_Recall: 0.0124 - val_l
oss: 0.1532
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0229 - loss: 0.1583 - val_Recall: 0.0372 - val_l
oss: 0.1480
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.0254 - loss: 0.1675 - val_Recall: 0.0664 - val_l
oss: 0.1434
Epoch 9/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0400 - loss: 0.1496 - val_Recall: 0.1081 - val_l
oss: 0.1391
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.0342 - loss: 0.1533 - val_Recall: 0.1441 - val_l
oss: 0.1349
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0328 - loss: 0.1484 - val_Recall: 0.1824 - val_l
oss: 0.1312
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.0665 - loss: 0.1644 - val_Recall: 0.2016 - val_l
oss: 0.1278
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.1322 - loss: 0.1561 - val_Recall: 0.2365 - val_l
oss: 0.1246
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.1733 - loss: 0.1347 - val_Recall: 0.2635 - val_l
oss: 0.1214
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.1417 - loss: 0.1555 - val_Recall: 0.3018 - val_l
oss: 0.1184
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.1726 - loss: 0.1450 - val_Recall: 0.3255 - val_l
oss: 0.1158
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.2741 - loss: 0.1280 - val_Recall: 0.3435 - val_l
oss: 0.1130
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.2344 - loss: 0.1378 - val_Recall: 0.3626 - val_l
oss: 0.1106
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.2031 - loss: 0.1404 - val_Recall: 0.3908 - val_l
oss: 0.1083
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.2680 - loss: 0.1373 - val_Recall: 0.4088 - val_l
oss: 0.1061
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.2274 - loss: 0.1274 - val_Recall: 0.4313 - val_l
oss: 0.1041
Epoch 22/50
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.3166 - loss: 0.1269 - val_Recall: 0.4437 - val_l
oss: 0.1023
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.2920 - loss: 0.1247 - val_Recall: 0.4628 - val_l
oss: 0.1005
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.4185 - loss: 0.1157 - val_Recall: 0.4752 - val_l
oss: 0.0991
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.3886 - loss: 0.1229 - val_Recall: 0.4932 - val_l
oss: 0.0972
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.4003 - loss: 0.1157 - val_Recall: 0.5023 - val_l
oss: 0.0957
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.3513 - loss: 0.1315 - val_Recall: 0.5101 - val_l
oss: 0.0942
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.3927 - loss: 0.1257 - val_Recall: 0.5214 - val_l
oss: 0.0929
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.3871 - loss: 0.1097 - val_Recall: 0.5293 - val_l
oss: 0.0916
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.4093 - loss: 0.1192 - val_Recall: 0.5372 - val_l
oss: 0.0903
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.4117 - loss: 0.1033 - val_Recall: 0.5495 - val_l
oss: 0.0888
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.4853 - loss: 0.0995 - val_Recall: 0.5541 - val_l
oss: 0.0875
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.4313 - loss: 0.1139 - val_Recall: 0.5541 - val_l
oss: 0.0870
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.4872 - loss: 0.0959 - val_Recall: 0.5631 - val_l
oss: 0.0859
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.4867 - loss: 0.1082 - val_Recall: 0.5732 - val_l
oss: 0.0850
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - Recall: 0.4235 - loss: 0.1101 - val_Recall: 0.5766 - val_l
oss: 0.0841
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.3820 - loss: 0.0989 - val_Recall: 0.5867 - val_l
oss: 0.0830
Epoch 38/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.5310 - loss: 0.1044 - val_Recall: 0.5935 - val_l
oss: 0.0821
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.4137 - loss: 0.1149 - val_Recall: 0.6081 - val_l
oss: 0.0810
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.5090 - loss: 0.1018 - val_Recall: 0.6092 - val_l
oss: 0.0806
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.5485 - loss: 0.1031 - val_Recall: 0.6137 - val_l
oss: 0.0799
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.5363 - loss: 0.0964 - val_Recall: 0.6250 - val_l
oss: 0.0790
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.4526 - loss: 0.0998 - val_Recall: 0.6318 - val_l
oss: 0.0784
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.5923 - loss: 0.0902 - val_Recall: 0.6340 - val_l
oss: 0.0778
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.5491 - loss: 0.0911 - val_Recall: 0.6430 - val_l
oss: 0.0771
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.5713 - loss: 0.0914 - val_Recall: 0.6475 - val_l
oss: 0.0765
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.5712 - loss: 0.0906 - val_Recall: 0.6554 - val_l
oss: 0.0758
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.5935 - loss: 0.0951 - val_Recall: 0.6622 - val_l
oss: 0.0751
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.5856 - loss: 0.0931 - val_Recall: 0.6644 - val_l
oss: 0.0750
```

```
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.5541 - loss: 0.0985 - val_Recall: 0.6667 - val_l
oss: 0.0746
Time taken in seconds  34.49052715301514
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step

Classification Report - Train data Model_2
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      3778
         1.0       0.98      0.78      0.87       222

    accuracy                           0.99      4000
   macro avg       0.99      0.89      0.93      4000
weighted avg       0.99      0.99      0.99      4000


Classification Report - Validation data Model_2
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15112
         1.0       0.96      0.74      0.84       888

    accuracy                           0.98     16000
   macro avg       0.97      0.87      0.91     16000
weighted avg       0.98      0.98      0.98     16000
```



Model 3

In [52]:

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

#Initializing the neural network. WIth 4 hidden layers this would be deep learning.
model_3 = Sequential()
model_3.add(Dense(128,activation="relu",input_dim = X_train.shape[1]))
model_3.add(Dropout(0.5))
model_3.add(Dense(64,activation="tanh"))
model_3.add(Dense(64,activation = "relu"))
model_3.add(Dropout(0.6))
model_3.add(Dense(128,activation = "relu"))
model_3.add(Dense(units=1,activation="sigmoid"))

optimizer = tf.keras.optimizers.SGD()
model_3.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['Recall'])
model_3.summary()
epochs = 50
batch_size = 100

start = time.time()
history = model_3.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()

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

plot(history,'loss')
model_3_train_perf = model_performance_classification(model_3, X_train, y_train)
model_3_train_perf

model_3_val_perf = model_performance_classification(model_3,X_val,y_val)
model_3_val_perf

y_train_pred_3 = model_3.predict(X_train)
y_val_pred_3 = model_3.predict(X_val)

print("\nClassification Report - Train data Model_3",end="\n")
cr_train_model_3 = classification_report(y_train,y_train_pred_0>0.5)
print(cr_train_model_3)

print("\nClassification Report - Validation data Model_3",end="\n")
cr_val_model_3 = classification_report(y_val,y_val_pred_0>0.5)
print(cr_val_model_3)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,248
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 64)	4,160
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 128)	8,320
dense_4 (Dense)	(None, 1)	129

Total params: 26,113 (102.00 KB)

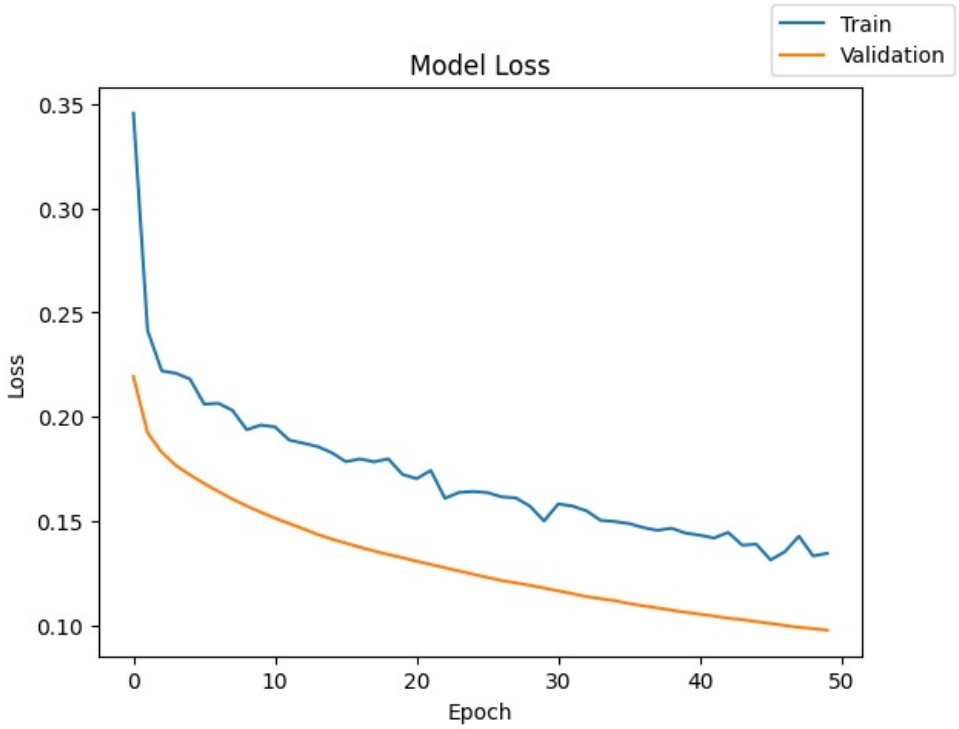
Trainable params: 26,113 (102.00 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - Recall: 0.0905 - loss: 0.4193 - val_Recall: 0.0000e+00 - v
al_loss: 0.2193
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.0000e+00 - loss: 0.2382 - val_Recall: 0.0000e+00
- val_loss: 0.1924
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0000e+00 - loss: 0.2086 - val_Recall: 0.0000e+00
- val_loss: 0.1831
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.0000e+00 - loss: 0.2131 - val_Recall: 0.0000e+00
- val_loss: 0.1767
```

```
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0000e+00 - loss: 0.2090 - val_Recall: 0.0000e+00
- val_loss: 0.1723
Epoch 6/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0000e+00 - loss: 0.1999 - val_Recall: 0.0000e+00
- val_loss: 0.1681
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0000e+00 - loss: 0.1857 - val_Recall: 0.0000e+00
- val_loss: 0.1643
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.0000e+00 - loss: 0.2038 - val_Recall: 0.0000e+00
- val_loss: 0.1606
Epoch 9/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.0000e+00 - loss: 0.1756 - val_Recall: 0.0000e+00
- val_loss: 0.1573
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.0015 - loss: 0.2064 - val_Recall: 0.0000e+00 - v
al_loss: 0.1543
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.0000e+00 - loss: 0.1924 - val_Recall: 0.0000e+00
- val_loss: 0.1515
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.0000e+00 - loss: 0.1856 - val_Recall: 0.0000e+00
- val_loss: 0.1489
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - Recall: 0.0000e+00 - loss: 0.1825 - val_Recall: 0.0000e+00
- val_loss: 0.1464
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0035 - loss: 0.1956 - val_Recall: 0.0000e+00 - v
al_loss: 0.1437
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0000e+00 - loss: 0.1847 - val_Recall: 0.0000e+00
- val_loss: 0.1415
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0028 - loss: 0.1856 - val_Recall: 0.0000e+00 - v
al_loss: 0.1396
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0000e+00 - loss: 0.1829 - val_Recall: 0.0000e+00
- val_loss: 0.1376
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0000e+00 - loss: 0.1616 - val_Recall: 0.0011 - v
al_loss: 0.1358
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.0050 - loss: 0.1849 - val_Recall: 0.0034 - val_l
oss: 0.1341
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.0064 - loss: 0.1761 - val_Recall: 0.0180 - val_l
oss: 0.1326
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0074 - loss: 0.1709 - val_Recall: 0.0428 - val_l
oss: 0.1309
Epoch 22/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0243 - loss: 0.1986 - val_Recall: 0.0642 - val_l
oss: 0.1293
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0012 - loss: 0.1637 - val_Recall: 0.1002 - val_l
oss: 0.1278
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0073 - loss: 0.1644 - val_Recall: 0.1318 - val_l
oss: 0.1262
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.0156 - loss: 0.1646 - val_Recall: 0.1723 - val_l
oss: 0.1247
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.0171 - loss: 0.1626 - val_Recall: 0.1959 - val_l
oss: 0.1231
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.0566 - loss: 0.1572 - val_Recall: 0.2342 - val_l
oss: 0.1217
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.0333 - loss: 0.1608 - val_Recall: 0.2680 - val_l
oss: 0.1205
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.0863 - loss: 0.1556 - val_Recall: 0.2860 - val_l
oss: 0.1194
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.0686 - loss: 0.1497 - val_Recall: 0.3176 - val_l
oss: 0.1180
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - Recall: 0.0736 - loss: 0.1619 - val_Recall: 0.3480 - val_l
oss: 0.1167
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.1233 - loss: 0.1578 - val_Recall: 0.3660 - val_l
```

```
oss: 0.1154
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.1059 - loss: 0.1428 - val_Recall: 0.3941 - val_l
oss: 0.1139
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.1586 - loss: 0.1284 - val_Recall: 0.4099 - val_l
oss: 0.1130
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.1093 - loss: 0.1558 - val_Recall: 0.4257 - val_l
oss: 0.1120
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.1004 - loss: 0.1431 - val_Recall: 0.4414 - val_l
oss: 0.1106
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.1406 - loss: 0.1462 - val_Recall: 0.4516 - val_l
oss: 0.1095
Epoch 38/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.1187 - loss: 0.1479 - val_Recall: 0.4718 - val_l
oss: 0.1085
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.1301 - loss: 0.1428 - val_Recall: 0.4910 - val_l
oss: 0.1074
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.1736 - loss: 0.1529 - val_Recall: 0.4966 - val_l
oss: 0.1064
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.1732 - loss: 0.1503 - val_Recall: 0.5079 - val_l
oss: 0.1055
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.1492 - loss: 0.1424 - val_Recall: 0.5146 - val_l
oss: 0.1046
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.1817 - loss: 0.1370 - val_Recall: 0.5304 - val_l
oss: 0.1036
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.1944 - loss: 0.1360 - val_Recall: 0.5315 - val_l
oss: 0.1029
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.1889 - loss: 0.1287 - val_Recall: 0.5383 - val_l
oss: 0.1019
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - Recall: 0.2886 - loss: 0.1292 - val_Recall: 0.5462 - val_l
oss: 0.1010
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.1932 - loss: 0.1460 - val_Recall: 0.5552 - val_l
oss: 0.1001
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.2458 - loss: 0.1345 - val_Recall: 0.5608 - val_l
oss: 0.0992
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.3140 - loss: 0.1256 - val_Recall: 0.5608 - val_l
oss: 0.0986
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.2607 - loss: 0.1340 - val_Recall: 0.5642 - val_l
oss: 0.0978
Time taken in seconds  40.62687277793884
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

Classification Report - Train data Model_3
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      3778
         1.0       0.98      0.78      0.87       222

    accuracy                           0.99      4000
   macro avg       0.99      0.89      0.93      4000
weighted avg       0.99      0.99      0.99      4000


Classification Report - Validation data Model_3
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15112
         1.0       0.96      0.74      0.84       888

    accuracy                           0.98     16000
   macro avg       0.97      0.87      0.91     16000
weighted avg       0.98      0.98      0.98     16000
```



Model 4

In [53]:

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

#Initializing the neural network.This will have 4 hidden layers with 2 dropouts. This time lets use SGD with mome
tum.
model_4 = Sequential()
model_4.add(Dense(128,activation="relu",input_dim = X_train.shape[1]))
model_4.add(Dropout(0.5))
model_4.add(Dense(64,activation="tanh"))
model_4.add(Dense(64,activation = "relu"))
model_4.add(Dropout(0.6))
model_4.add(Dense(128,activation = "tanh"))
model_4.add(Dense(units=1,activation="sigmoid"))

optimizer = tf.keras.optimizers.SGD(momentum=0.95)
model_4.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['Recall'])
model_4.summary()
epochs = 50
batch_size = 100

start = time.time()
history = model_4.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()

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

plot(history,'loss')
model_4_train_perf = model_performance_classification(model_4, X_train, y_train)
model_4_train_perf

model_4_val_perf = model_performance_classification(model_4,X_val,y_val)
model_4_val_perf

y_train_pred_4 = model_4.predict(X_train)
y_val_pred_4 = model_4.predict(X_val)

print("\nClassification Report - Train data Model_4",end="\n")
cr_train_model_4 = classification_report(y_train,y_train_pred_0>0.5)
print(cr_train_model_4)

print("\nClassification Report - Validation data Model_4",end="\n")
cr_val_model_4 = classification_report(y_val,y_val_pred_0>0.5)
print(cr_val_model_4)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,248
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 64)	4,160
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 128)	8,320
dense_4 (Dense)	(None, 1)	129

Total params: 26,113 (102.00 KB)

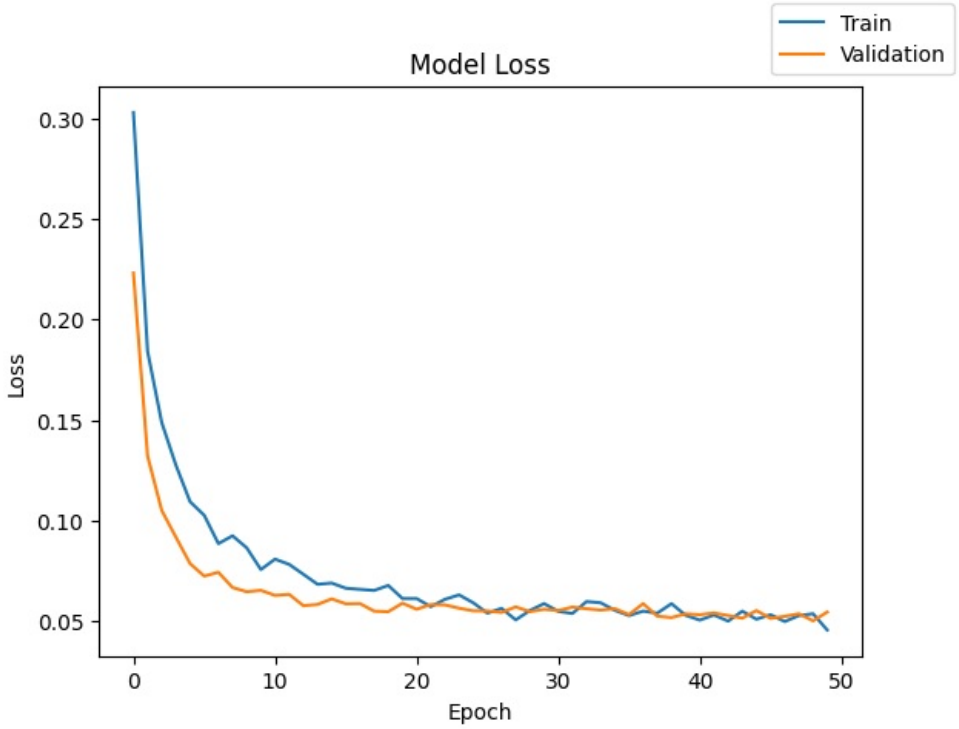
Trainable params: 26,113 (102.00 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - Recall: 0.2027 - loss: 0.4048 - val_Recall: 0.0000e+00 - v
al_loss: 0.2230
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.0046 - loss: 0.2013 - val_Recall: 0.1441 - val_l
oss: 0.1321
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.1331 - loss: 0.1459 - val_Recall: 0.5574 - val_l
oss: 0.1051
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.3253 - loss: 0.1391 - val_Recall: 0.6227 - val_l
oss: 0.0918
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.4998 - loss: 0.0998 - val_Recall: 0.6836 - val_l
oss: 0.0787
Epoch 6/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.6372 - loss: 0.0925 - val_Recall: 0.7095 - val_l
oss: 0.0725
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.6669 - loss: 0.0865 - val_Recall: 0.7207 - val_l
oss: 0.0744
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.6326 - loss: 0.0880 - val_Recall: 0.7646 - val_l
oss: 0.0668
Epoch 9/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.6408 - loss: 0.0818 - val_Recall: 0.7658 - val_l
oss: 0.0647
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.6870 - loss: 0.0719 - val_Recall: 0.7894 - val_l
oss: 0.0654
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.7326 - loss: 0.0713 - val_Recall: 0.7905 - val_l
oss: 0.0629
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.6972 - loss: 0.0824 - val_Recall: 0.7691 - val_l
oss: 0.0634
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.7430 - loss: 0.0596 - val_Recall: 0.8097 - val_l
oss: 0.0578
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7937 - loss: 0.0585 - val_Recall: 0.8052 - val_l
oss: 0.0584
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7552 - loss: 0.0675 - val_Recall: 0.7827 - val_l
oss: 0.0612
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - Recall: 0.7825 - loss: 0.0636 - val_Recall: 0.7905 - val_l
oss: 0.0587
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.7689 - loss: 0.0580 - val_Recall: 0.8018 - val_l
oss: 0.0588
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - Recall: 0.7722 - loss: 0.0707 - val_Recall: 0.8063 - val_l
oss: 0.0551
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.7814 - loss: 0.0679 - val_Recall: 0.8266 - val_l
oss: 0.0548
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.7445 - loss: 0.0614 - val_Recall: 0.8074 - val_l
oss: 0.0590
```

```
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - Recall: 0.7883 - loss: 0.0550 - val_Recall: 0.8187 - val_l
oss: 0.0560
Epoch 22/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - Recall: 0.7981 - loss: 0.0627 - val_Recall: 0.8119 - val_l
oss: 0.0583
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - Recall: 0.8014 - loss: 0.0662 - val_Recall: 0.8052 - val_l
oss: 0.0582
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.7909 - loss: 0.0606 - val_Recall: 0.8131 - val_l
oss: 0.0565
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.7635 - loss: 0.0595 - val_Recall: 0.8153 - val_l
oss: 0.0552
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.8502 - loss: 0.0527 - val_Recall: 0.8277 - val_l
oss: 0.0552
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.8491 - loss: 0.0534 - val_Recall: 0.8266 - val_l
oss: 0.0544
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8177 - loss: 0.0537 - val_Recall: 0.8108 - val_l
oss: 0.0572
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8301 - loss: 0.0536 - val_Recall: 0.8266 - val_l
oss: 0.0548
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8041 - loss: 0.0523 - val_Recall: 0.8300 - val_l
oss: 0.0560
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.7924 - loss: 0.0554 - val_Recall: 0.8119 - val_l
oss: 0.0554
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8369 - loss: 0.0519 - val_Recall: 0.8131 - val_l
oss: 0.0571
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7462 - loss: 0.0582 - val_Recall: 0.8153 - val_l
oss: 0.0563
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.7812 - loss: 0.0635 - val_Recall: 0.8288 - val_l
oss: 0.0556
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8587 - loss: 0.0464 - val_Recall: 0.8221 - val_l
oss: 0.0563
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.7800 - loss: 0.0618 - val_Recall: 0.8266 - val_l
oss: 0.0534
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.8554 - loss: 0.0519 - val_Recall: 0.8119 - val_l
oss: 0.0588
Epoch 38/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - Recall: 0.8093 - loss: 0.0548 - val_Recall: 0.8288 - val_l
oss: 0.0526
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - Recall: 0.8267 - loss: 0.0570 - val_Recall: 0.8243 - val_l
oss: 0.0519
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8594 - loss: 0.0443 - val_Recall: 0.8300 - val_l
oss: 0.0537
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.8354 - loss: 0.0452 - val_Recall: 0.8367 - val_l
oss: 0.0533
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8797 - loss: 0.0371 - val_Recall: 0.8311 - val_l
oss: 0.0541
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.8553 - loss: 0.0451 - val_Recall: 0.8232 - val_l
oss: 0.0529
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7934 - loss: 0.0699 - val_Recall: 0.8333 - val_l
oss: 0.0516
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.7992 - loss: 0.0490 - val_Recall: 0.8356 - val_l
oss: 0.0553
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7983 - loss: 0.0563 - val_Recall: 0.8322 - val_l
oss: 0.0515
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8202 - loss: 0.0545 - val_Recall: 0.8243 - val_l
oss: 0.0525
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.8422 - loss: 0.0454 - val_Recall: 0.8131 - val_l
```

```
oss: 0.0538
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.7953 - loss: 0.0556 - val_Recall: 0.8333 - val_l
oss: 0.0503
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8278 - loss: 0.0424 - val_Recall: 0.8322 - val_l
oss: 0.0546
Time taken in seconds  48.51417112350464
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

Classification Report - Train data Model_4
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      3778
         1.0       0.98      0.78      0.87       222

    accuracy                           0.99      4000
   macro avg       0.99      0.89      0.93      4000
weighted avg       0.99      0.99      0.99      4000


Classification Report - Validation data Model_4
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15112
         1.0       0.96      0.74      0.84       888

    accuracy                           0.98     16000
   macro avg       0.97      0.87      0.91     16000
weighted avg       0.98      0.98      0.98     16000
```



Model 5

In [54]:

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

#Initializing the neural network. Reduce the hidden layers and changethe SGD with momentum param.
model_5 = Sequential()
model_5.add(Dense(128,activation="relu",input_dim = X_train.shape[1]))
model_5.add(Dense(64,activation="tanh"))
model_5.add(Dropout(0.6))
model_5.add(Dense(128,activation = "tanh"))
model_5.add(Dense(units=1,activation="sigmoid"))

#optimizer = tf.keras.optimizers.SGD()
optimizer = tf.keras.optimizers.SGD(momentum=0.75)
#optimizer = tf.keras.optimizers.adam(learning_rate=0.001)
model_5.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['Recall'])
model_5.summary()
epochs = 50
batch_size = 100

start = time.time()
history = model_5.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()

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

plot(history,'loss')
model_5_train_perf = model_performance_classification(model_5, X_train, y_train)
model_5_train_perf

model_5_val_perf = model_performance_classification(model_5,X_val,y_val)
model_5_val_perf

y_train_pred_5 = model_5.predict(X_train)
y_val_pred_5 = model_5.predict(X_val)

print("\nClassification Report - Train data Model_5",end="\n")
cr_train_model_5 = classification_report(y_train,y_train_pred_0>0.5)
print(cr_train_model_5)

print("\nClassification Report - Validation data Model_5",end="\n")
cr_val_model_5 = classification_report(y_val,y_val_pred_0>0.5)
print(cr_val_model_5)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,248
dense_1 (Dense)	(None, 64)	8,256
dropout (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 128)	8,320
dense_3 (Dense)	(None, 1)	129

Total params: 21,953 (85.75 KB)

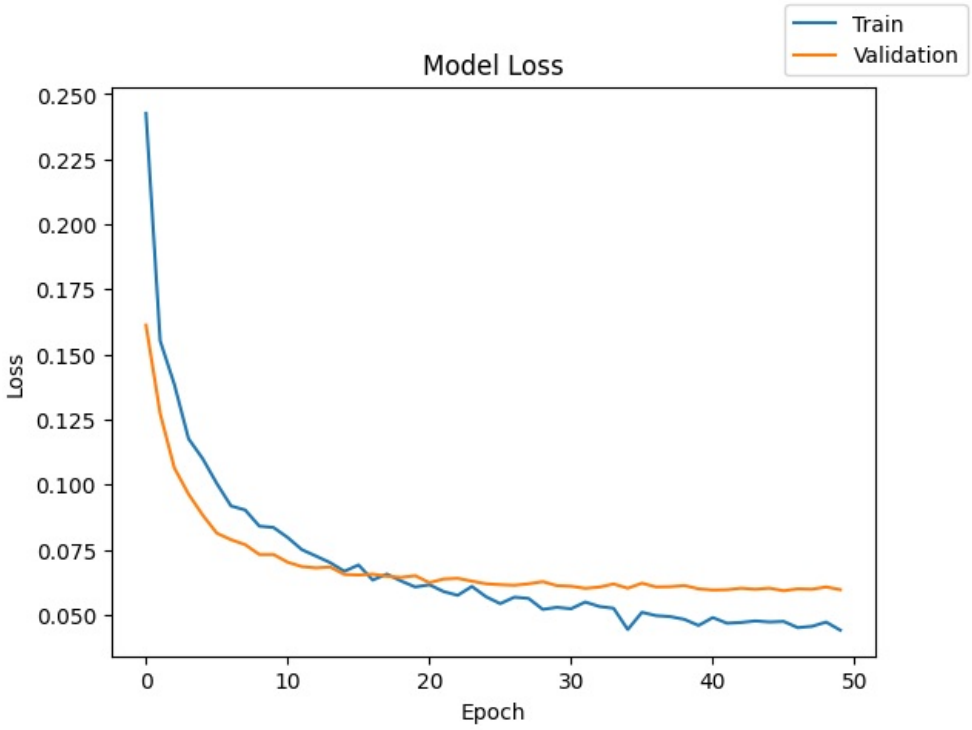
Trainable params: 21,953 (85.75 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - Recall: 0.0878 - loss: 0.3537 - val_Recall: 0.0124 - val_l
oss: 0.1612
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.1267 - loss: 0.1541 - val_Recall: 0.2027 - val_l
oss: 0.1273
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.2782 - loss: 0.1347 - val_Recall: 0.3863 - val_l
oss: 0.1064
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.3894 - loss: 0.1203 - val_Recall: 0.4797 - val_l
oss: 0.0964
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.5448 - loss: 0.1075 - val_Recall: 0.5439 - val_l
oss: 0.0883
Epoch 6/50
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.5272 - loss: 0.1049 - val_Recall: 0.6250 - val_l
oss: 0.0813
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.6204 - loss: 0.0852 - val_Recall: 0.6374 - val_l
oss: 0.0788
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.6067 - loss: 0.0926 - val_Recall: 0.6633 - val_l
oss: 0.0769
Epoch 9/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.6645 - loss: 0.0785 - val_Recall: 0.7095 - val_l
oss: 0.0731
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.6646 - loss: 0.0927 - val_Recall: 0.7061 - val_l
oss: 0.0731
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.6774 - loss: 0.0791 - val_Recall: 0.7286 - val_l
oss: 0.0702
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.7186 - loss: 0.0736 - val_Recall: 0.7399 - val_l
oss: 0.0685
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.6972 - loss: 0.0686 - val_Recall: 0.7387 - val_l
oss: 0.0680
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.8047 - loss: 0.0624 - val_Recall: 0.7387 - val_l
oss: 0.0683
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7680 - loss: 0.0651 - val_Recall: 0.7658 - val_l
oss: 0.0655
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.7434 - loss: 0.0719 - val_Recall: 0.7646 - val_l
oss: 0.0653
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8215 - loss: 0.0553 - val_Recall: 0.7568 - val_l
oss: 0.0656
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.7955 - loss: 0.0550 - val_Recall: 0.7658 - val_l
oss: 0.0648
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7537 - loss: 0.0590 - val_Recall: 0.7748 - val_l
oss: 0.0643
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7854 - loss: 0.0613 - val_Recall: 0.7601 - val_l
oss: 0.0650
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.7285 - loss: 0.0721 - val_Recall: 0.7962 - val_l
oss: 0.0623
Epoch 22/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7903 - loss: 0.0605 - val_Recall: 0.7849 - val_l
oss: 0.0637
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8472 - loss: 0.0479 - val_Recall: 0.7804 - val_l
oss: 0.0640
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8015 - loss: 0.0609 - val_Recall: 0.7894 - val_l
oss: 0.0629
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8090 - loss: 0.0538 - val_Recall: 0.7939 - val_l
oss: 0.0619
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8320 - loss: 0.0510 - val_Recall: 0.8007 - val_l
oss: 0.0615
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.7776 - loss: 0.0553 - val_Recall: 0.8029 - val_l
oss: 0.0613
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.8457 - loss: 0.0518 - val_Recall: 0.7917 - val_l
oss: 0.0619
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.8565 - loss: 0.0424 - val_Recall: 0.7894 - val_l
oss: 0.0627
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.8236 - loss: 0.0511 - val_Recall: 0.7973 - val_l
oss: 0.0612
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.7966 - loss: 0.0627 - val_Recall: 0.8007 - val_l
oss: 0.0609
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8249 - loss: 0.0560 - val_Recall: 0.8074 - val_l
oss: 0.0601
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8320 - loss: 0.0501 - val_Recall: 0.7995 - val_l
oss: 0.0606
```

```
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.8428 - loss: 0.0488 - val_Recall: 0.7950 - val_l
oss: 0.0618
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8063 - loss: 0.0512 - val_Recall: 0.8086 - val_l
oss: 0.0602
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.7937 - loss: 0.0610 - val_Recall: 0.7984 - val_l
oss: 0.0621
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8536 - loss: 0.0518 - val_Recall: 0.8063 - val_l
oss: 0.0607
Epoch 38/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8739 - loss: 0.0402 - val_Recall: 0.8074 - val_l
oss: 0.0608
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8320 - loss: 0.0498 - val_Recall: 0.7984 - val_l
oss: 0.0612
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8462 - loss: 0.0410 - val_Recall: 0.8119 - val_l
oss: 0.0599
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8636 - loss: 0.0397 - val_Recall: 0.8142 - val_l
oss: 0.0595
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8410 - loss: 0.0447 - val_Recall: 0.8164 - val_l
oss: 0.0596
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8455 - loss: 0.0425 - val_Recall: 0.8131 - val_l
oss: 0.0602
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8364 - loss: 0.0509 - val_Recall: 0.8131 - val_l
oss: 0.0598
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8707 - loss: 0.0369 - val_Recall: 0.8108 - val_l
oss: 0.0602
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.8360 - loss: 0.0411 - val_Recall: 0.8209 - val_l
oss: 0.0592
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8516 - loss: 0.0532 - val_Recall: 0.8108 - val_l
oss: 0.0599
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8513 - loss: 0.0474 - val_Recall: 0.8187 - val_l
oss: 0.0598
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.8440 - loss: 0.0503 - val_Recall: 0.8119 - val_l
oss: 0.0607
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.8758 - loss: 0.0365 - val_Recall: 0.8198 - val_l
oss: 0.0596
Time taken in seconds  37.506916761398315
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

Classification Report - Train data Model_5
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      3778
         1.0       0.98      0.78      0.87       222

    accuracy                           0.99      4000
   macro avg       0.99      0.89      0.93      4000
weighted avg       0.99      0.99      0.99      4000


Classification Report - Validation data Model_5
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15112
         1.0       0.96      0.74      0.84       888

    accuracy                           0.98     16000
   macro avg       0.97      0.87      0.91     16000
weighted avg       0.98      0.98      0.98     16000
```



In Model 5, we added 3 hidden layers and a drropout of 60%, which gives is pretty good results in training and validation for the recall.

Model 6

In [55]:

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

#Initializing the neural network. Lets keep it simple. 3 hidden layers and 1 dropout.
model_6 = Sequential()
model_6.add(Dense(128,activation="relu",input_dim = X_train.shape[1]))
model_6.add(Dense(64,activation="tanh"))
model_6.add(Dense(128,activation = "tanh"))
model_6.add(Dropout(0.6))
model_6.add(Dense(units=1,activation="sigmoid"))

#optimizer = tf.keras.optimizers.SGD()
optimizer = tf.keras.optimizers.SGD(momentum=0.9)
model_6.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['Recall'])
model_6.summary()
epochs = 50
batch_size = 100

start = time.time()
history = model_6.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()

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

plot(history,'loss')
model_6_train_perf = model_performance_classification(model_6, X_train, y_train)
model_6_train_perf

model_6_val_perf = model_performance_classification(model_6,X_val,y_val)
model_6_val_perf

y_train_pred_6 = model_6.predict(X_train)
y_val_pred_6 = model_6.predict(X_val)

print("\nClassification Report - Train data Model_6",end="\n")
cr_train_model_6 = classification_report(y_train,y_train_pred_0>0.5)
print(cr_train_model_6)

print("\nClassification Report - Validation data Model_6",end="\n")
cr_val_model_6 = classification_report(y_val,y_val_pred_0>0.5)
print(cr_val_model_6)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,248
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 128)	8,320
dropout (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129

Total params: 21,953 (85.75 KB)

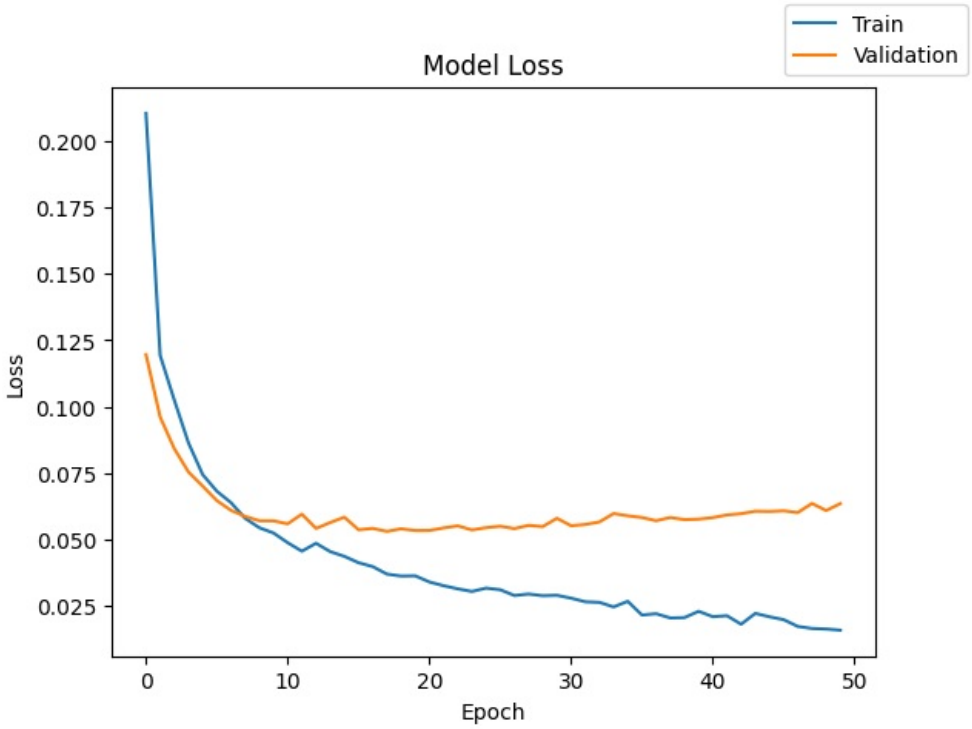
Trainable params: 21,953 (85.75 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - Recall: 0.2410 - loss: 0.3163 - val_Recall: 0.4155 - val_l
oss: 0.1196
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.4256 - loss: 0.1237 - val_Recall: 0.5282 - val_l
oss: 0.0961
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.5504 - loss: 0.1015 - val_Recall: 0.5890 - val_l
oss: 0.0841
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.5783 - loss: 0.0903 - val_Recall: 0.6791 - val_l
oss: 0.0753
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7031 - loss: 0.0769 - val_Recall: 0.6712 - val_l
oss: 0.0700
Epoch 6/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7333 - loss: 0.0645 - val_Recall: 0.7432 - val_l
oss: 0.0646
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8047 - loss: 0.0562 - val_Recall: 0.7534 - val_l
oss: 0.0609
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7883 - loss: 0.0543 - val_Recall: 0.7827 - val_l
oss: 0.0586
Epoch 9/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.8637 - loss: 0.0469 - val_Recall: 0.7962 - val_l
oss: 0.0570
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.8211 - loss: 0.0611 - val_Recall: 0.7714 - val_l
oss: 0.0569
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.8203 - loss: 0.0513 - val_Recall: 0.7782 - val_l
oss: 0.0558
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7944 - loss: 0.0566 - val_Recall: 0.7556 - val_l
oss: 0.0595
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.7920 - loss: 0.0569 - val_Recall: 0.8097 - val_l
oss: 0.0541
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.8351 - loss: 0.0578 - val_Recall: 0.7827 - val_l
oss: 0.0563
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.7980 - loss: 0.0527 - val_Recall: 0.7714 - val_l
oss: 0.0583
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8450 - loss: 0.0396 - val_Recall: 0.7950 - val_l
oss: 0.0536
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8748 - loss: 0.0371 - val_Recall: 0.7984 - val_l
oss: 0.0541
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8472 - loss: 0.0374 - val_Recall: 0.8187 - val_l
oss: 0.0530
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8911 - loss: 0.0380 - val_Recall: 0.8063 - val_l
oss: 0.0539
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8879 - loss: 0.0295 - val_Recall: 0.8345 - val_l
oss: 0.0533
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8913 - loss: 0.0357 - val_Recall: 0.8176 - val_l
oss: 0.0533
Epoch 22/50
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9059 - loss: 0.0311 - val_Recall: 0.8142 - val_l
oss: 0.0543
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.8527 - loss: 0.0347 - val_Recall: 0.8074 - val_l
oss: 0.0551
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.9062 - loss: 0.0276 - val_Recall: 0.8288 - val_l
oss: 0.0535
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8781 - loss: 0.0327 - val_Recall: 0.8300 - val_l
oss: 0.0544
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8835 - loss: 0.0377 - val_Recall: 0.8243 - val_l
oss: 0.0549
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8838 - loss: 0.0284 - val_Recall: 0.8356 - val_l
oss: 0.0540
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.9164 - loss: 0.0297 - val_Recall: 0.8119 - val_l
oss: 0.0552
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.9015 - loss: 0.0268 - val_Recall: 0.8300 - val_l
oss: 0.0548
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.9147 - loss: 0.0277 - val_Recall: 0.8018 - val_l
oss: 0.0579
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - Recall: 0.8766 - loss: 0.0230 - val_Recall: 0.8311 - val_l
oss: 0.0551
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.8812 - loss: 0.0302 - val_Recall: 0.8345 - val_l
oss: 0.0556
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8989 - loss: 0.0294 - val_Recall: 0.8221 - val_l
oss: 0.0565
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9147 - loss: 0.0237 - val_Recall: 0.8041 - val_l
oss: 0.0597
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8845 - loss: 0.0331 - val_Recall: 0.8097 - val_l
oss: 0.0588
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9257 - loss: 0.0175 - val_Recall: 0.8142 - val_l
oss: 0.0582
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9379 - loss: 0.0148 - val_Recall: 0.8367 - val_l
oss: 0.0570
Epoch 38/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9182 - loss: 0.0197 - val_Recall: 0.8255 - val_l
oss: 0.0582
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.9265 - loss: 0.0256 - val_Recall: 0.8457 - val_l
oss: 0.0574
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.9254 - loss: 0.0192 - val_Recall: 0.8468 - val_l
oss: 0.0576
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9224 - loss: 0.0188 - val_Recall: 0.8300 - val_l
oss: 0.0582
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9137 - loss: 0.0201 - val_Recall: 0.8300 - val_l
oss: 0.0592
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9598 - loss: 0.0134 - val_Recall: 0.8288 - val_l
oss: 0.0596
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9218 - loss: 0.0204 - val_Recall: 0.8266 - val_l
oss: 0.0606
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9066 - loss: 0.0215 - val_Recall: 0.8356 - val_l
oss: 0.0605
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9538 - loss: 0.0170 - val_Recall: 0.8345 - val_l
oss: 0.0607
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.9440 - loss: 0.0126 - val_Recall: 0.8300 - val_l
oss: 0.0601
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - Recall: 0.8946 - loss: 0.0203 - val_Recall: 0.8164 - val_l
oss: 0.0635
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - Recall: 0.9476 - loss: 0.0123 - val_Recall: 0.8288 - val_l
oss: 0.0608
```

```
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.9356 - loss: 0.0162 - val_Recall: 0.8176 - val_l
oss: 0.0634
Time taken in seconds  36.97223782539368
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

Classification Report - Train data Model_6
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      3778
         1.0       0.98      0.78      0.87       222

    accuracy                           0.99      4000
   macro avg       0.99      0.89      0.93      4000
weighted avg       0.99      0.99      0.99      4000


Classification Report - Validation data Model_6
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15112
         1.0       0.96      0.74      0.84       888

    accuracy                           0.98     16000
   macro avg       0.97      0.87      0.91     16000
weighted avg       0.98      0.98      0.98     16000
```



Model 6 is pretty similar to Model 5, except increase the optimizer momentum. It takes more time and with similar performance results.

Model 7

In [56]:

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

#Initializing the neural network. 3 hidden layers, Adam optimizer and a dropout.
model_7 = Sequential()
model_7.add(Dense(128,activation="relu",input_dim = X_train.shape[1]))
model_7.add(Dense(64,activation="tanh"))
model_7.add(Dense(128,activation = "tanh"))
model_7.add(Dropout(0.6))
model_7.add(Dense(units=1,activation="sigmoid"))

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model_7.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['Recall'])
model_7.summary()
epochs = 50
batch_size = 100

start = time.time()
history = model_7.fit(X_train, y_train, validation_data=(X_val,y_val) , batch_size=batch_size, epochs=epochs)
end=time.time()

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

plot(history,'loss')
model_7_train_perf = model_performance_classification(model_7, X_train, y_train)
model_7_train_perf

model_7_val_perf = model_performance_classification(model_7,X_val,y_val)
model_7_val_perf

y_train_pred_7 = model_7.predict(X_train)
y_val_pred_7 = model_7.predict(X_val)

print("\nClassification Report - Train data Model_7",end="\n")
cr_train_model_7 = classification_report(y_train,y_train_pred_0>0.5)
print(cr_train_model_7)

print("\nClassification Report - Validation data Model_7",end="\n")
cr_val_model_7 = classification_report(y_val,y_val_pred_0>0.5)
print(cr_val_model_7)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,248
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 128)	8,320
dropout (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129

Total params: 21,953 (85.75 KB)

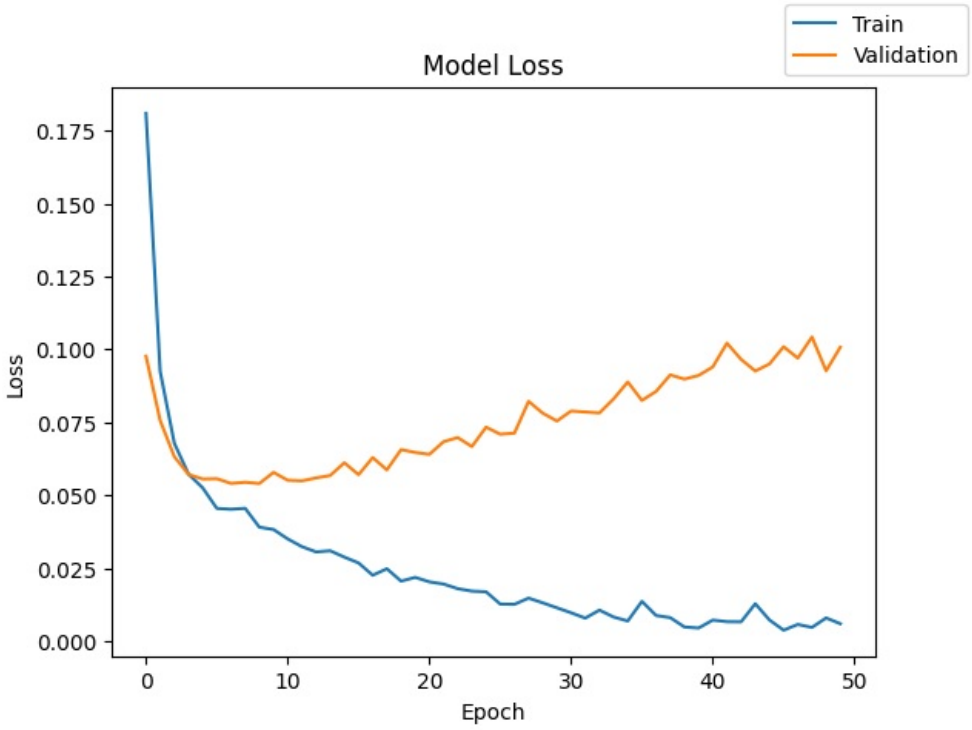
Trainable params: 21,953 (85.75 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - Recall: 0.1739 - loss: 0.2684 - val_Recall: 0.4944 - val_l
oss: 0.0976
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.4643 - loss: 0.0998 - val_Recall: 0.6318 - val_l
oss: 0.0756
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7209 - loss: 0.0665 - val_Recall: 0.7275 - val_l
oss: 0.0632
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.7520 - loss: 0.0735 - val_Recall: 0.7849 - val_l
oss: 0.0571
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8421 - loss: 0.0495 - val_Recall: 0.8063 - val_l
oss: 0.0556
Epoch 6/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8839 - loss: 0.0358 - val_Recall: 0.8221 - val_l
oss: 0.0557
```

```
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8659 - loss: 0.0406 - val_Recall: 0.8142 - val_l
oss: 0.0541
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - Recall: 0.8625 - loss: 0.0489 - val_Recall: 0.8356 - val_l
oss: 0.0544
Epoch 9/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.8751 - loss: 0.0346 - val_Recall: 0.8367 - val_l
oss: 0.0540
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.9228 - loss: 0.0341 - val_Recall: 0.7928 - val_l
oss: 0.0578
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.8749 - loss: 0.0260 - val_Recall: 0.8266 - val_l
oss: 0.0551
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.8981 - loss: 0.0297 - val_Recall: 0.8277 - val_l
oss: 0.0549
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9188 - loss: 0.0259 - val_Recall: 0.8209 - val_l
oss: 0.0559
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9057 - loss: 0.0268 - val_Recall: 0.8288 - val_l
oss: 0.0568
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9062 - loss: 0.0266 - val_Recall: 0.8097 - val_l
oss: 0.0611
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9045 - loss: 0.0229 - val_Recall: 0.8412 - val_l
oss: 0.0570
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9224 - loss: 0.0176 - val_Recall: 0.7995 - val_l
oss: 0.0629
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.8638 - loss: 0.0309 - val_Recall: 0.8345 - val_l
oss: 0.0587
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9144 - loss: 0.0198 - val_Recall: 0.8097 - val_l
oss: 0.0656
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - Recall: 0.9406 - loss: 0.0195 - val_Recall: 0.8266 - val_l
oss: 0.0647
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.8989 - loss: 0.0214 - val_Recall: 0.8401 - val_l
oss: 0.0640
Epoch 22/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9094 - loss: 0.0212 - val_Recall: 0.8176 - val_l
oss: 0.0683
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.9281 - loss: 0.0177 - val_Recall: 0.8119 - val_l
oss: 0.0698
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.9361 - loss: 0.0158 - val_Recall: 0.8288 - val_l
oss: 0.0667
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.9440 - loss: 0.0123 - val_Recall: 0.7917 - val_l
oss: 0.0733
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.9440 - loss: 0.0139 - val_Recall: 0.8468 - val_l
oss: 0.0709
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.9347 - loss: 0.0108 - val_Recall: 0.8266 - val_l
oss: 0.0713
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.9670 - loss: 0.0117 - val_Recall: 0.7736 - val_l
oss: 0.0822
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9468 - loss: 0.0112 - val_Recall: 0.8052 - val_l
oss: 0.0781
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9472 - loss: 0.0101 - val_Recall: 0.8300 - val_l
oss: 0.0754
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9613 - loss: 0.0079 - val_Recall: 0.8074 - val_l
oss: 0.0789
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9630 - loss: 0.0072 - val_Recall: 0.8086 - val_l
oss: 0.0785
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9519 - loss: 0.0086 - val_Recall: 0.8345 - val_l
oss: 0.0782
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - Recall: 0.9613 - loss: 0.0088 - val_Recall: 0.8131 - val_l
```

```
oss: 0.0830
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9580 - loss: 0.0079 - val_Recall: 0.8131 - val_l
oss: 0.0888
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9531 - loss: 0.0158 - val_Recall: 0.8277 - val_l
oss: 0.0825
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - Recall: 0.9668 - loss: 0.0061 - val_Recall: 0.8198 - val_l
oss: 0.0856
Epoch 38/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9618 - loss: 0.0086 - val_Recall: 0.7950 - val_l
oss: 0.0912
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.9786 - loss: 0.0041 - val_Recall: 0.8018 - val_l
oss: 0.0898
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9688 - loss: 0.0055 - val_Recall: 0.8198 - val_l
oss: 0.0910
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - Recall: 0.9615 - loss: 0.0095 - val_Recall: 0.8108 - val_l
oss: 0.0940
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9649 - loss: 0.0065 - val_Recall: 0.7793 - val_l
oss: 0.1021
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - Recall: 0.9666 - loss: 0.0056 - val_Recall: 0.8131 - val_l
oss: 0.0966
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - Recall: 0.9821 - loss: 0.0081 - val_Recall: 0.8333 - val_l
oss: 0.0926
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.9731 - loss: 0.0083 - val_Recall: 0.8029 - val_l
oss: 0.0950
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - Recall: 0.9961 - loss: 0.0040 - val_Recall: 0.8018 - val_l
oss: 0.1009
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - Recall: 0.9750 - loss: 0.0048 - val_Recall: 0.8074 - val_l
oss: 0.0970
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9748 - loss: 0.0051 - val_Recall: 0.7894 - val_l
oss: 0.1042
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - Recall: 0.9711 - loss: 0.0102 - val_Recall: 0.8311 - val_l
oss: 0.0926
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - Recall: 0.9925 - loss: 0.0050 - val_Recall: 0.8164 - val_l
oss: 0.1007
Time taken in seconds  38.77640628814697
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

Classification Report - Train data Model_7
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      3778
         1.0       0.98      0.78      0.87       222

    accuracy                           0.99      4000
   macro avg       0.99      0.89      0.93      4000
weighted avg       0.99      0.99      0.99      4000


Classification Report - Validation data Model_7
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15112
         1.0       0.96      0.74      0.84       888

    accuracy                           0.98     16000
   macro avg       0.97      0.87      0.91     16000
weighted avg       0.98      0.98      0.98     16000
```



We added the adam optimizer to Model 7. This increased hte time taken to 56 seconds and we didnt get a lot of performance gain.

Model Performance Comparison and Final Model Selection

Now, in order to select the final model, we will compare the performances of all the models for the training and validation sets.

In [57]:

```
# training performance comparison

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,
        model_6_train_perf.T,
        model_7_train_perf.T
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Model 0",
    "Model 1",
    "Model 2",
    "Model 3",
    "Model 4",
    "Model 5",
    "Model 6",
    "Model 7"
]
print("Training set performance comparison:")
models_train_comp_df
```

```
Training set performance comparison:
```

Out[57]:

	Model 0	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7
Accuracy	0.987250	0.055500	0.981000	0.972250	0.991750	0.992250	0.996500	0.999000
Recall	0.891495	0.500000	0.837308	0.769079	0.934155	0.938660	0.970588	0.995231
Precision	0.985248	0.027750	0.977653	0.951941	0.986035	0.986398	0.995904	0.995231
F1 Score	0.932735	0.052582	0.893952	0.834624	0.958444	0.961138	0.982871	0.995231

In [58]:

```
# Validation performance comparison

models_val_comp_df = pd.concat( [model_0_val_perf.T,model_1_val_perf.T,model_2_val_perf.T, model_3_val_perf.T,  m
odel_4_val_perf.T, model_5_val_perf.T, model_6_val_perf.T, model_7_val_perf.T], axis=1,)
models_val_comp_df.columns = [
    "Model 0",
    "Model 1",
    "Model 2",
    "Model 3",
    "Model 4",
    "Model 5",
    "Model 6",
    "Model 7"
]
print("Validation set performance comparison:")
models_val_comp_df
```

```
Validation set performance comparison:
```

Out[58]:

	Model 0	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7
Accuracy	0.983938	0.055500	0.979500	0.972125	0.989000	0.987812	0.987938	0.985062
Recall	0.868542	0.500000	0.832275	0.780143	0.915210	0.908752	0.907758	0.905706
Precision	0.974072	0.027750	0.964734	0.934789	0.977486	0.971814	0.974211	0.947197
F1 Score	0.913878	0.052582	0.886155	0.838695	0.943896	0.937732	0.938158	0.925306

From the above reslts we can tell that Model 6 with a recall score of 0.93 in training and 0.91 in validation with a 36.1 seconds running time has the best score and also runs in the least time.

SO, WE WILL GO WITH MODEL 6

Also, the other scores on model 6 the precision score, F1 score and Accuracy are also good.

On the test set

In [59]:

```
# Test set performance for the best model
best_model_test_perf = model_performance_classification(model_6,X_test,y_test)
best_model_test_perf
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

Out[59]:

	Accuracy	Recall	Precision	F1 Score
0	0.9858	0.890784	0.97196	0.927057

In [65]:

```
y_test_pred_best = model_6.predict(X_test)

cr_test_best_model = classification_report(y_test, y_test_pred_best>0.5) # Check the classification report of bes
t model on test data.
print(cr_test_best_model)
print(y_test_pred_best)
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      4718
         1.0       0.96      0.78      0.86       282

    accuracy                           0.99      5000
   macro avg       0.97      0.89      0.93      5000
weighted avg       0.99      0.99      0.99      5000

[[3.1423382e-04]
 [1.3502027e-03]
 [7.2512426e-05]
 ...
 [1.9259626e-06]
 [7.5510255e-04]
 [3.5153824e-04]]
```

Actionable Insights and Recommendations

We have 282 sensors which have a high possibility of failure out of 5000 sensors, which is about 5.64%

In order to avoid replacement costs, its important to inspect these ASAP.

Its not possible to predict which sensors since we dont have that data with us.

Furthur steps - THis can be run through a decision tree or other ensemble models to ensure we get the important features or the specific ones.