

# Hybrid Network Intrusion Detection System

## Objective

The Objective of this notebook is to Make a Classification Model that accurately classify the given data points

## Dataset

The dataset used in this notebook is the '**UNSW\_NB15 - Dataset**'

---

## Table of contents

- 1 | Importing Required Libraries & DataSource
  - Performance monitoring
  - Load and Inspect Dataset
  - Understanding About features
  - Concatenating the Data
- 2 | Data Analysis and basic preprocessing
  - Computing Dimension of Dataset
  - Statistical Summary of Dataset
  - Checking if There's Any Duplicate Records
  - Computing Total No. of Missing Values and the Percentage of Missing Values
  - Performing Descriptive Analysis
  - Performing Descriptive Analysis on Categorical Attributes
  - Checking Unique Class of Categorical Attributes
- 3 | Exploratory Data Analysis
  - Checking for outliers
  - Visualising Data Distribution and Skewness
  - Visualising Class Distribution
- 4 | Preprocessing
  - Feature Engineering
  - Encode leftover categorical features
  - Checking Highly Correlated Features
  - Splitting the features in dependent and independent features
  - Applying SMOTE to balance the unbalanced data

- Checking Feature Importance
- Dividing in train-test split + Scaling and PCA variance
- 5 | Model Building
  - Decision Tree Model
  - Random Forest Model
  - Gradient Boosting Model
  - Logistic Regression Model
  - KNeighbors Classifier model
  - Extra Trees Model
  - SVM model (SGDClassifier)
  - SVM model (SVC)
  - SVM model (OneClassSVM) - Unsupervised
  - Summary and Classifier Ensemble
- 6 | Final Metrics
- 7 | Keras Models
  - Neural Network MLP (Keras)
  - GRU (Keras)
  - LSTM (Keras)
- 8 | Evaluation with Keras Models
- 9 | End of Performance Monitoring

# 1 | Importing Required Libraries & DataSource

## Performance monitoring

This counts the entire pre/post-processing and train-evaluation period.

```
In [ ]: import time
import psutil

# Record the start time and initial system statistics
notebook_start_time = time.time()
process = psutil.Process()

start_cpu_times = psutil.cpu_times()
start_memory_info = process.memory_info()
start_disk_io = psutil.disk_io_counters()
start_net_io = psutil.net_io_counters()
```

```
In [ ]: # Libraries for Data Manipulation
import pandas as pd
import numpy as np

import gc
import os
import poplib
# Libraries for Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
import altair as alt
from scipy.stats import skew
sns.set_theme(style="white", font_scale=1.5)
sns.set_theme(rc={"axes.facecolor": "#FFFACD", "figure.facecolor": "#FFFACD"})
sns.set_context("poster", font_scale=.7)
import matplotlib.ticker as ticker

# Libraries to Handle Warnings
import warnings
warnings.filterwarnings('ignore')

# Libraries for Statistical Analysis
from scipy import stats
from scipy.stats import chi2, chi2_contingency

# Setting Display Options
pd.set_option("display.max.columns", None)

# Machine Learning Algorithms
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import mutual_info_regression
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import joblib
from joblib import dump

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout, LeakyReLU
from tensorflow.keras.initializers import HeNormal
from tensorflow.keras.regularizers import l1, l2
from tensorflow.keras.optimizers import SGD, Adam
import keras_tuner
import keras

# Model Evaluation Metrics
from sklearn.metrics import (
    accuracy_score,
    recall_score,
    precision_score,
    f1_score, silhouette_score,
    confusion_matrix,
    classification_report,
    roc_auc_score,
    roc_curve,
```

```

        mean_absolute_error
    )

# Data Source
# import os
# for dirname, _, filenames in os.walk('/kaggle/input'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))

```

## Load and Inspect Dataset

```
In [ ]: testing_set = pd.read_csv('kaggle/input/unsw-nb15/UNSW_NB15_testing-set.csv')
training_set = pd.read_csv('kaggle/input/unsw-nb15/UNSW_NB15_training-set.csv')
LIST_EVENTS = pd.read_csv('kaggle/input/unsw-nb15/UNSW-NB15_LIST_EVENTS.csv')
NB15_1 = pd.read_csv('kaggle/input/unsw-nb15/UNSW-NB15_1.csv')
NB15_2 = pd.read_csv('kaggle/input/unsw-nb15/UNSW-NB15_2.csv')
NB15_3 = pd.read_csv('kaggle/input/unsw-nb15/UNSW-NB15_3.csv')
NB15_4 = pd.read_csv('kaggle/input/unsw-nb15/UNSW-NB15_4.csv')
NB15_features = pd.read_csv('kaggle/input/unsw-nb15/NUSW-NB15_features.csv', enc
```

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

```
Out[ ]:
```

	<b>id</b>	<b>dur</b>	<b>proto</b>	<b>service</b>	<b>state</b>	<b>spkts</b>	<b>dpkts</b>	<b>sbytes</b>	<b>dbbytes</b>	<b>rate</b>	<b>sttl</b>	<b>dt</b>
<b>0</b>	1	0.121478	tcp	-	FIN	6	4	258	172	74.087490	252	25
<b>1</b>	2	0.649902	tcp	-	FIN	14	38	734	42014	78.473372	62	25
<b>2</b>	3	1.623129	tcp	-	FIN	8	16	364	13186	14.170161	62	25
<b>3</b>	4	1.681642	tcp	ftp	FIN	12	12	628	770	13.677108	62	25
<b>4</b>	5	0.449454	tcp	-	FIN	10	6	534	268	33.373826	254	25

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

```
Out[ ]:
```

	<b>id</b>	<b>dur</b>	<b>proto</b>	<b>service</b>	<b>state</b>	<b>spkts</b>	<b>dpkts</b>	<b>sbytes</b>	<b>dbbytes</b>	<b>rate</b>	<b>sttl</b>
<b>0</b>	1	0.000011	udp	-	INT	2	0	496	0	90909.0902	254
<b>1</b>	2	0.000008	udp	-	INT	2	0	1762	0	125000.0003	254
<b>2</b>	3	0.000005	udp	-	INT	2	0	1068	0	200000.0051	254
<b>3</b>	4	0.000006	udp	-	INT	2	0	900	0	166666.6608	254
<b>4</b>	5	0.000010	udp	-	INT	2	0	2126	0	100000.0025	254

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

Out[ ]: **Attack category** **Attack subcategory** **Number of events**

<b>0</b>	normal	NaN	2218761
<b>1</b>	Fuzzers	FTP	558
<b>2</b>	Fuzzers	HTTP	1497
<b>3</b>	Fuzzers	RIP	3550
<b>4</b>	Fuzzers	SMB	5245

In [ ]: NB15\_1.head()

Out[ ]: **59.166.0.0** **1390** **149.171.126.6** **53** **udp** **CON** **0.001055** **132** **164** **31** **29** **0**

<b>0</b>	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	29	0
<b>1</b>	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	29	0
<b>2</b>	59.166.0.5	3593	149.171.126.5	53	udp	CON	0.001209	132	164	31	29	0
<b>3</b>	59.166.0.3	49664	149.171.126.0	53	udp	CON	0.001169	146	178	31	29	0
<b>4</b>	59.166.0.0	32119	149.171.126.9	111	udp	CON	0.078339	568	312	31	29	0

◀ ▶

In [ ]: NB15\_2.head()

Out[ ]: **59.166.0.0** **6055** **149.171.126.5** **54145** **tcp** **FIN** **0.072974** **4238** **60788** **31** **29**

<b>0</b>	59.166.0.0	7832	149.171.126.3	5607	tcp	FIN	0.144951	5174	91072	31	29
<b>1</b>	59.166.0.8	11397	149.171.126.6	21	tcp	FIN	0.116107	2934	3742	31	29
<b>2</b>	59.166.0.0	3804	149.171.126.3	53	udp	CON	0.000986	146	178	31	29
<b>3</b>	59.166.0.8	14339	149.171.126.6	14724	tcp	FIN	0.038480	8928	320	31	29
<b>4</b>	59.166.0.8	39094	149.171.126.3	53	udp	CON	0.001026	130	162	31	29

◀ ▶

In [ ]: NB15\_3.head()

Out[ ]:

	<b>59.166.0.1</b>	<b>18247</b>	<b>149.171.126.4</b>	<b>7662</b>	<b>tcp</b>	<b>FIN</b>	<b>0.119596</b>	<b>4550</b>	<b>68342</b>	<b>31</b>
<b>0</b>	59.166.0.3	54771	149.171.126.2	27709	tcp	FIN	0.650574	8928	320	31
<b>1</b>	59.166.0.8	13289	149.171.126.9	5190	tcp	FIN	0.007980	2158	2464	31
<b>2</b>	149.171.126.18	1043	175.45.176.3	53	udp	INT	0.000005	264	0	60
<b>3</b>	149.171.126.18	1043	175.45.176.3	53	udp	INT	0.000005	264	0	60
<b>4</b>	59.166.0.3	10275	149.171.126.0	25	tcp	FIN	0.486578	37462	3380	31

In [ ]: NB15\_4.head()

Out[ ]:

	<b>59.166.0.9</b>	<b>7045</b>	<b>149.171.126.7</b>	<b>25</b>	<b>tcp</b>	<b>FIN</b>	<b>0.201886</b>	<b>37552</b>	<b>3380</b>	<b>31</b>
<b>0</b>	59.166.0.9	9685	149.171.126.2	80	tcp	FIN	5.864748	19410	1087890	31
<b>1</b>	59.166.0.2	1421	149.171.126.4	53	udp	CON	0.001391	146	178	31
<b>2</b>	59.166.0.2	21553	149.171.126.2	25	tcp	FIN	0.053948	37812	3380	31
<b>3</b>	59.166.0.8	45212	149.171.126.4	53	udp	CON	0.000953	146	178	31
<b>4</b>	59.166.0.0	59922	149.171.126.8	6881	tcp	FIN	8.633186	25056	1094788	31

## Understanding About features

In [ ]: NB15\_features

Out[ ]:

No.	Name	Type	Description
0	srcip	nominal	Source IP address
1	sport	integer	Source port number
2	dstip	nominal	Destination IP address
3	dsport	integer	Destination port number
4	proto	nominal	Transaction protocol
5	state	nominal	Indicates to the state and its dependent proto...
6	dur	Float	Record total duration
7	sbytes	Integer	Source to destination transaction bytes
8	dbytes	Integer	Destination to source transaction bytes
9	sttl	Integer	Source to destination time to live value
10	dttl	Integer	Destination to source time to live value
11	sloss	Integer	Source packets retransmitted or dropped
12	dloss	Integer	Destination packets retransmitted or dropped
13	service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and ...
14	Sload	Float	Source bits per second
15	Dload	Float	Destination bits per second
16	Spkts	integer	Source to destination packet count
17	Dpkts	integer	Destination to source packet count
18	swin	integer	Source TCP window advertisement value
19	dwin	integer	Destination TCP window advertisement value
20	stcpb	integer	Source TCP base sequence number
21	dtcpb	integer	Destination TCP base sequence number
22	smeansz	integer	Mean of the ?ow packet size transmitted by the...
23	dmeansz	integer	Mean of the ?ow packet size transmitted by the...
24	trans_depth	integer	Represents the pipelined depth into the connec...
25	res_bdy_len	integer	Actual uncompressed content size of the data t...
26	Sjit	Float	Source jitter (mSec)
27	Djit	Float	Destination jitter (mSec)
28	Stime	Timestamp	record start time
29	Ltime	Timestamp	record last time
30	Sintpkt	Float	Source interpacket arrival time (mSec)
31	Dintpkt	Float	Destination interpacket arrival time (mSec)
32	tcprtt	Float	TCP connection setup round-trip time, the sum ...

No.	Name	Type	Description
33	34	synack	Float TCP connection setup time, the time between th...
34	35	ackdat	Float TCP connection setup time, the time between th...
35	36	is_sm_ips_ports	Binary If source (1) and destination (3)IP addresses ...
36	37	ct_state_ttl	Integer No. for each state (6) according to specific r...
37	38	ct_flw_http_mthd	Integer No. of flows that has methods such as Get and ...
38	39	is_ftp_login	Binary If the ftp session is accessed by user and pas...
39	40	ct_ftp_cmd	integer No of flows that has a command in ftp session.
40	41	ct_srv_src	integer No. of connections that contain the same servi...
41	42	ct_srv_dst	integer No. of connections that contain the same servi...
42	43	ct_dst_ltm	integer No. of connections of the same destination add...
43	44	ct_src_ltm	integer No. of connections of the same source address ...
44	45	ct_src_dport_ltm	integer No of connections of the same source address (...)
45	46	ct_dst_sport_ltm	integer No of connections of the same destination addr...
46	47	ct_dst_src_ltm	integer No of connections of the same source (1) and t...
47	48	attack_cat	nominal The name of each attack category. In this data...
48	49	Label	binary 0 for normal and 1 for attack records

## Concatenating the Data

```
In [ ]: NB15_1.columns = NB15_features['Name']
NB15_2.columns = NB15_features['Name']
NB15_3.columns = NB15_features['Name']
NB15_4.columns = NB15_features['Name']

In [ ]: train_df = pd.concat([NB15_1, NB15_2, NB15_3, NB15_4], ignore_index=True)
train_df = train_df.rename(columns={'ct_src_ltm': 'ct_src_ltm'})

In [ ]: # Shuffle the data points in train_df
train_df = train_df.sample(frac=1, random_state=42).reset_index(drop=True)

In [ ]: train_df
```

Out[ ]:

Name	srcip	sport	dstip	dsport	proto	state	dur	sbytes
0	175.45.176.3	57672	149.171.126.15	3260	tcp	CON	0.285356	986
1	59.166.0.8	38052	149.171.126.9	6881	tcp	FIN	0.314311	1540
2	59.166.0.0	42911	149.171.126.2	38558	udp	CON	0.301180	536
3	175.45.176.1	47439	149.171.126.14	53	udp	INT	0.000009	114
4	59.166.0.5	61544	149.171.126.6	53	udp	CON	0.001079	146
...	...	...	...	...	...	...	...	...
<b>2540038</b>	59.166.0.8	34415	149.171.126.0	5190	tcp	FIN	0.008119	1920
<b>2540039</b>	59.166.0.8	56352	149.171.126.2	53	udp	CON	0.001047	130
<b>2540040</b>	59.166.0.9	25527	149.171.126.4	6881	tcp	FIN	0.013106	1540
<b>2540041</b>	175.45.176.1	47439	149.171.126.14	53	udp	INT	0.000003	114
<b>2540042</b>	149.171.126.14	1043	175.45.176.1	53	udp	INT	0.000002	264

2540043 rows × 49 columns



In [ ]: `# Save the shuffled DataFrame to a new CSV file  
train_df.to_csv('rawdataset.csv', index=False)`

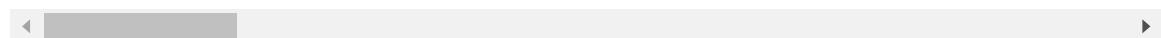
In [ ]: `list_drop = ['attack_cat', 'srcip', 'dstip']  
train_df.drop(list_drop, axis=1, inplace=True)  
  
# test_list_drop = ['id', 'attack_cat']  
# testing_set.drop(test_list_drop, axis=1, inplace=True)  
  
# testing_set.rename(columns={'Label': 'Label'}, inplace=True)`

In [ ]: `train_df`

Out[ ]:

Name	sport	dsport	proto	state	dur	sbytes	dbytes	sttl	dttl	sloss	dloss
0	57672	3260	tcp	CON	0.285356	986	86	62	252	2	1
1	38052	6881	tcp	FIN	0.314311	1540	1644	31	29	4	4
2	42911	38558	udp	CON	0.301180	536	304	31	29	0	0
3	47439	53	udp	INT	0.000009	114	0	254	0	0	0
4	61544	53	udp	CON	0.001079	146	178	31	29	0	0
...	...	...	...	...	...	...	...	...	...	...	...
<b>2540038</b>	34415	5190	tcp	FIN	0.008119	1920	4312	31	29	6	6
<b>2540039</b>	56352	53	udp	CON	0.001047	130	162	31	29	0	0
<b>2540040</b>	25527	6881	tcp	FIN	0.013106	1540	1644	31	29	4	4
<b>2540041</b>	47439	53	udp	INT	0.000003	114	0	254	0	0	0
<b>2540042</b>	1043	53	udp	INT	0.000002	264	0	60	0	0	0

2540043 rows × 46 columns



## 2 | Data Analysis and Basic preprocessing

### Computing Dimension of Dataset

```
In [ ]: print("dataset shape: ",train_df.shape)
```

```
dataset shape: (2540043, 46)
```



#### Inference:

- There is total **2540043 records** and **49 columns** available in the train\_dataset.

### Statistical Summary of Dataset

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

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2540043 entries, 0 to 2540042
Data columns (total 46 columns):
 #   Column           Dtype  
--- 
 0   sport            object  
 1   dsport           object  
 2   proto            object  
 3   state             object  
 4   dur              float64 
 5   sbytes           int64  
 6   dbytes           int64  
 7   sttl              int64  
 8   ttl              int64  
 9   sloss             int64  
 10  dloss             int64  
 11  service           object  
 12  Sload             float64 
 13  Dload             float64 
 14  Spkts             int64  
 15  Dpkts             int64  
 16  swin              int64  
 17  dwin              int64  
 18  stcpb             int64  
 19  dtcpb             int64  
 20  smeansz           int64  
 21  dmeansz           int64  
 22  trans_depth       int64  
 23  res_bdy_len       int64  
 24  Sjit              float64 
 25  Djit              float64 
 26  Stime             int64  
 27  Ltime             int64  
 28  Sintpkt           float64 
 29  Dintpkt           float64 
 30  tcprtt            float64 
 31  synack            float64 
 32  ackdat            float64 
 33  is_sm_ips_ports  int64  
 34  ct_state_ttl      int64  
 35  ct_flw_http_mthd float64 
 36  is_ftp_login      float64 
 37  ct_ftp_cmd        object  
 38  ct_srv_src        int64  
 39  ct_srv_dst        int64  
 40  ct_dst_ltm         int64  
 41  ct_src_ltm         int64  
 42  ct_src_dport_ltm  int64  
 43  ct_dst_sport_ltm  int64  
 44  ct_dst_src_ltm    int64  
 45  Label              int64  
dtypes: float64(12), int64(28), object(6)
memory usage: 891.4+ MB

```

```

In [ ]: # Identify the data types of columns
column_data_types = train_df.dtypes

# Count the numerical and categorical columns
numerical_count = 0
categorical_count = 0

```

```

for column_name, data_type in column_data_types.items():
    if np.issubdtype(data_type, np.number):
        numerical_count += 1
    else:
        categorical_count += 1

# Print the counts
print(f"There are {numerical_count} Numerical Columns in dataset")
print(f"There are {categorical_count} Categorical Columns in dataset")

```

There are 40 Numerical Columns in dataset  
 There are 6 Categorical Columns in dataset

## Checking if There's Any Duplicate Records

```
In [ ]: print("Duplicates in train_df: ",train_df.duplicated().sum())
```

Duplicates in train\_df: 491849

```
In [ ]: train_df = train_df.drop_duplicates()
```



### Inference:

- There are 480626 duplicate records present in the dataset.

## Computing Total No. of Missing Values and the Percentage of Missing Values

```
In [ ]: missing_data = train_df.isnull().sum().to_frame().rename(columns={0:"Total No
missing_data["% of Missing Values"] = round((missing_data["Total No. of Missi
missing_data
```

Out[ ]:

Name	Total No. of Missing Values	% of Missing Values
<b>sport</b>	0	0.00
<b>dsport</b>	0	0.00
<b>proto</b>	0	0.00
<b>state</b>	0	0.00
<b>dur</b>	0	0.00
<b>sbytes</b>	0	0.00
<b>dbytes</b>	0	0.00
<b>sttl</b>	0	0.00
<b>dttl</b>	0	0.00
<b>sloss</b>	0	0.00
<b>dloss</b>	0	0.00
<b>service</b>	0	0.00
<b>Sload</b>	0	0.00
<b>Dload</b>	0	0.00
<b>Spkts</b>	0	0.00
<b>Dpkts</b>	0	0.00
<b>swin</b>	0	0.00
<b>dwin</b>	0	0.00
<b>stcpb</b>	0	0.00
<b>dtcpb</b>	0	0.00
<b>smeansz</b>	0	0.00
<b>dmeansz</b>	0	0.00
<b>trans_depth</b>	0	0.00
<b>res_bdy_len</b>	0	0.00
<b>Sjit</b>	0	0.00
<b>Djit</b>	0	0.00
<b>Stime</b>	0	0.00
<b>Ltime</b>	0	0.00
<b>Sintpkt</b>	0	0.00
<b>Dintpkt</b>	0	0.00
<b>tcprtt</b>	0	0.00
<b>synack</b>	0	0.00

Name	Total No. of Missing Values	% of Missing Values
ackdat	0	0.00
is_sm_ips_ports	0	0.00
ct_state_ttl	0	0.00
ct_flw_http_mthd	923892	45.11
is_ftp_login	1004211	49.03
ct_ftp_cmd	0	0.00
ct_srv_src	0	0.00
ct_srv_dst	0	0.00
ct_dst_ltm	0	0.00
ct_src_ltm	0	0.00
ct_src_dport_ltm	0	0.00
ct_dst_sport_ltm	0	0.00
ct_dst_src_ltm	0	0.00
Label	0	0.00

**Handling Null values here because it create wrong inference in EDA**

```
In [ ]: # # NaN values implies that no attack are there for that particular row data,
# train_df['attack_cat'].fillna('normal', inplace = True)
# train_df['attack_cat'] = train_df['attack_cat'].apply(lambda x: x.strip().l

# missing values imply that there were no flows with HTTP methods in certain
train_df['ct_flw_http_mthd'].fillna(0, inplace=True)

# is_ftp_login is of type binary that mean it takes 1(user has loged in) or 0
train_df['is_ftp_login'].fillna(0, inplace=True)
```



### Inference:

- ct\_flw\_http\_mthd, is\_ftp\_login and attack\_cat of the Attribute are having Missing Values we handle it later.

## Performing Descriptive Analysis

```
In [ ]: round(train_df.describe().T,2)
```

Out[ ]:

count mean std min 2!

Name					
<b>dur</b>	2048194.0	6.400000e-01	1.105000e+01	0.000000e+00	0.000000e+
<b>sbytes</b>	2048194.0	5.115540e+03	6.227923e+04	0.000000e+00	4.240000e+
<b> nbytes</b>	2048194.0	4.214292e+04	1.700893e+05	0.000000e+00	3.040000e+
<b>sttl</b>	2048194.0	4.252000e+01	4.834000e+01	0.000000e+00	3.100000e+
<b>dttl</b>	2048194.0	3.666000e+01	4.367000e+01	0.000000e+00	2.900000e+
<b>sloss</b>	2048194.0	6.150000e+00	2.463000e+01	0.000000e+00	0.000000e+
<b>dloss</b>	2048194.0	1.903000e+01	5.959000e+01	0.000000e+00	0.000000e+
<b>Sload</b>	2048194.0	8.335933e+06	7.207406e+07	0.000000e+00	8.036232e+
<b>Dload</b>	2048194.0	2.970461e+06	4.487254e+06	0.000000e+00	7.638325e+
<b>Spkts</b>	2048194.0	3.882000e+01	8.063000e+01	0.000000e+00	4.000000e+
<b>Dpkts</b>	2048194.0	5.004000e+01	1.273500e+02	0.000000e+00	4.000000e+
<b>swin</b>	2048194.0	1.803400e+02	1.160300e+02	0.000000e+00	0.000000e+
<b>dwin</b>	2048194.0	1.799700e+02	1.162000e+02	0.000000e+00	0.000000e+
<b>stcpb</b>	2048194.0	1.516590e+09	1.429920e+09	0.000000e+00	0.000000e+
<b>d tcb p b</b>	2048194.0	1.516382e+09	1.429779e+09	0.000000e+00	0.000000e+
<b>smeansz</b>	2048194.0	1.301100e+02	1.649300e+02	0.000000e+00	6.200000e+
<b>dmeansz</b>	2048194.0	3.319400e+02	3.397700e+02	0.000000e+00	8.100000e+
<b>trans_depth</b>	2048194.0	1.000000e-01	3.800000e-01	0.000000e+00	0.000000e+
<b>res_bdy_len</b>	2048194.0	5.180000e+03	5.225101e+04	0.000000e+00	0.000000e+
<b>S jit</b>	2048194.0	1.654010e+03	1.359472e+04	0.000000e+00	0.000000e+
<b>D jit</b>	2048194.0	8.457600e+02	3.258080e+03	0.000000e+00	3.300000e-
<b>Stime</b>	2048194.0	1.423098e+09	1.145972e+06	1.421927e+09	1.421949e+
<b>Ltime</b>	2048194.0	1.423098e+09	1.145972e+06	1.421927e+09	1.421949e+
<b>Sintpkt</b>	2048194.0	1.534600e+02	2.420990e+03	0.000000e+00	2.800000e-
<b>Dintpkt</b>	2048194.0	6.746000e+01	1.283740e+03	0.000000e+00	2.300000e-
<b>tcp rtt</b>	2048194.0	1.000000e-02	5.000000e-02	0.000000e+00	0.000000e+
<b>synack</b>	2048194.0	0.000000e+00	3.000000e-02	0.000000e+00	0.000000e+
<b>ackdat</b>	2048194.0	0.000000e+00	3.000000e-02	0.000000e+00	0.000000e+
<b>is_sm_ips_ports</b>	2048194.0	0.000000e+00	3.000000e-02	0.000000e+00	0.000000e+
<b>ct_state_ttl</b>	2048194.0	9.000000e-02	4.200000e-01	0.000000e+00	0.000000e+
<b>ct_flw_http_mthd</b>	2048194.0	1.200000e-01	5.000000e-01	0.000000e+00	0.000000e+
<b>is_ftp_login</b>	2048194.0	2.000000e-02	1.400000e-01	0.000000e+00	0.000000e+

Name	count	mean	std	min	2!
<b>ct_srv_src</b>	2048194.0	5.410000e+00	5.190000e+00	1.000000e+00	2.000000e+
<b>ct_srv_dst</b>	2048194.0	5.160000e+00	5.010000e+00	1.000000e+00	2.000000e+
<b>ct_dst_ltm</b>	2048194.0	3.680000e+00	3.270000e+00	1.000000e+00	2.000000e+
<b>ct_src_ltm</b>	2048194.0	4.160000e+00	3.590000e+00	1.000000e+00	2.000000e+
<b>ct_src_dport_ltm</b>	2048194.0	1.620000e+00	2.660000e+00	1.000000e+00	1.000000e+
<b>ct_dst_sport_ltm</b>	2048194.0	1.250000e+00	1.750000e+00	1.000000e+00	1.000000e+
<b>ct_dst_src_ltm</b>	2048194.0	2.640000e+00	4.140000e+00	1.000000e+00	1.000000e+
<b>Label</b>	2048194.0	4.000000e-02	2.000000e-01	0.000000e+00	0.000000e+

## Performing Descriptive Analysis on Categorical Attributes.

In [ ]:	train_df.describe(include="O").T			
Out[ ]:	count	unique	top	freq
Name				
<b>sport</b>	2048194	100343	1043	18503
<b>dsport</b>	2048194	128297	53	233820
<b>proto</b>	2048194	135	tcp	1448579
<b>state</b>	2048194	16	FIN	1433230
<b>service</b>	2048194	13	-	1158268
<b>ct_ftp_cmd</b>	2048194	13		1004211

## Checking Unique Values of Attributes

```
In [ ]: cols = train_df.columns

for column in cols:
    print('Unique values of ',column , 'with dtype',train_df[column].dtype,
    print('-'*100)
```

```

Unique values of sport with dtype object have total values 100343 -> [57672
38052 42911 ... '17953' '20349' '53235']

-----
Unique values of dsport with dtype object have total values 128297 -> ['326
0' '6881' 38558 ... 62291 28427 38903]

-----
Unique values of proto with dtype object have total values 135 -> ['tcp' 'ud
p' 'arp' 'ptp' 'unas' 'ip' 'vrrp' 'pnni' '3pc' 'sun-nd' 'wsn'
'ospf' 'st2' 'kryptolan' 'sctp' 'pvp' 'fc' 'wb-expak' 'secure-vmtp'
'ipv6-no' 'mhrp' 'xtp' 'hmp' 'idpr-cmtp' 'merit-inp' 'icmp' 'pim' 'any'
'cbt' 'crudp' 'ipcv' 'larp' 'sm' 'iatp' 'iso-ip' 'rsvp' 'ddx' 'chaos'
'tp++' 'mobile' 'cftp' 'ttp' 'sprite-rpc' 'aes-sp3-d' 'aris' 'sep'
'encap' 'dcn' 'igmp' 'isis' 'leaf-2' 'ipnip' 'argus' 'compaq-peer' 'vmtp'
'ipv6-route' 'egp' 'uti' 'zero' 'tlsp' 'wb-mon' 'sdrp' 'swipe' 'tcf'
'iso-tp4' 'narpp' 'ippc' 'vines' 'skip' 'rvd' 'emcon' 'ipv6-opt' 'a/n'
'gre' 'nvp' 'gmtp' 'ipv6' 'br-sat-mon' 'dgp' 'idrp' 'pri-enc' 'eigrp'
'pup' 'sat-mon' 'trunk-2' 'xnet' 'ipv6-frag' 'nsfnet-igp' 'ipx-n-ip'
'leaf-1' 'stp' 'cphb' 'cpnx' 'ipip' 'i-nlsp' 'micp' 'visa' 'prm'
'sccpmce' 'il' 'igp' 'sat-expak' 'pgm' 'rdp' 'srp' 'ipt' 'trunk-1'
'ggp' 'ax.25' 'netblt' 'ipcomp' 'crtp' 'smp' 'mfe-nsp' 'mux' 'l2tp'
'irtcp' 'bbn-rcc' 'etherip' 'ifmp' 'snp' 'fire' 'xns-idp' 'mtp' 'ib' 'qnx'
'idpr' 'bna' 'sps' 'scps' 'ddp' 'pipe' 'rtp' 'esp' 'udt']
-----
```

```

Unique values of state with dtype object have total values 16 -> ['CON' 'FI
N' 'INT' 'REQ' 'ECR' 'ECO' 'ACC' 'CLO' 'RST' 'PAR' 'URH' 'MAS'
'URN' 'TXD' 'no' 'TST']

-----
Unique values of dur with dtype float64 have total values 587303 -> [0.28535
6 0.314311 0.30118 ... 0.195193 0.594081 0.631521]
-----
```

```

Unique values of sbytes with dtype int64 have total values 14155 -> [ 986
1540 536 ... 48424 69911 28210]
-----
```

```

Unique values of dbytes with dtype int64 have total values 19166 -> [ 86
1644 304 ... 484528 33722 28330]
-----
```

```

Unique values of sttl with dtype int64 have total values 13 -> [ 62 31 254
60 0 32 1 29 64 63 30 255 252]
-----
```

```

Unique values of ttl with dtype int64 have total values 11 -> [252 29 0
30 31 60 254 32 62 253 64]
-----
```

```

Unique values of sloss with dtype int64 have total values 544 -> [ 2 4
0 7 1 19 3 21 30 6 54 9 10 11
18 77 27 5 14 20 38 22 35 28 12 15 26 42
8 17 78 32 81 16 37 13 144 39 118 55 31 53
24 52 36 622 130 29 23 34 47 70 41 85 46 50
49 33 74 129 51 43 25 82 449 110 177 71 68 2986
344 102 44 57 120 272 72 48 327 126 45 143 218 2203
100 56 121 1540 278 127 125 40 193 840 128 58 184 444
392 91 2687 76 310 124 123 75 328 98 967 94 137 146
]
```

266	2820	366	1940	478	79	334	211	83	358	105	163	274	131
194	66	133	69	279	809	167	73	2401	89	314	182	3669	187
236	568	353	142	114	117	62	195	138	198	141	385	202	90
350	149	2226	1536	4694	155	296	337	109	116	96	113	67	122
106	61	165	934	304	3057	624	289	2982	3417	281	65	2303	307
180	107	1458	850	152	287	495	631	2003	332	546	181	1225	140
101	2682	186	1350	63	99	136	340	2690	302	154	362	1347	5319
2179	238	80	103	560	191	188	473	303	799	2535	625	348	3164
108	319	192	881	2530	59	313	349	229	93	164	269	189	257
410	315	95	341	271	87	139	64	196	1364	398	1220	360	179
150	228	293	111	97	104	119	345	159	381	231	376	830	2746
88	171	217	286	346	3290	343	507	84	183	1030	2699	406	208
409	370	401	1367	2698	92	375	512	407	586	135	331	325	2706
294	999	157	285	326	261	2157	244	566	254	160	1627	288	2201
691	3518	2405	3838	3407	3607	2591	134	112	170	1335	548	339	352
132	2832	2457	263	233	168	273	335	60	1372	174	230	393	219
3338	932	933	280	394	153	2045	216	487	391	4158	1976	2101	797
368	2309	464	4206	2014	1362	207	1352	301	259	209	1876	3436	1424
3624	161	386	4745	1236	770	237	427	1812	147	241	377	913	2822
4210	1841	3386	847	2707	414	240	86	3592	930	5096	300	329	355
2718	243	297	2184	145	166	2085	396	2688	199	3213	1346	308	2065
270	357	502	532	402	2403	537	311	205	535	1985	250	2709	1484
4803	1358	330	309	395	3993	2730	210	354	1004	451	148	291	425
476	284	3951	227	175	156	1877	438	1241	169	276	525	413	905
2468	356	389	185	2070	221	4707	3484	4033	408	3476	1341	2343	2840
2011	322	295	556	2594	204	3307	336	200	1906	3129	324	115	178
162	1111	268	2019	528	448	390	1288	388	275	1910	367	943	151
1157	232	4546	190	808	4439	2719	1357	2703	380	1360	239	2110	277
758	2528	213	1922	552	1214	365	741	305	726	2693	333	364	2705
420	911	3830	283	619	422	299	611	2679	460	253	2848	2851	2225
426	338	747	399	3643	795	347	816	403	712	2623	2205]		

---

-----  
Unique values of dloss with dtype int64 have total values 707 -> [ 1 4

0	14	15	2	16	11	12	8	20	370	5	197		
27	32	7	67	22	18	21	17	6	24	39	9	177	36
71	42	13	30	26	861	33	182	390	207	175	28	260	280
365	89	130	164	31	10	192	29	118	65	352	103	3	95
294	77	380	199	94	74	59	163	44	116	389	109	357	40
98	255	363	169	35	147	48	583	318	25	308	72	349	19
63	373	155	372	1315	345	359	354	87	43	188	183	151	193
4829	64	144	312	379	196	76	124	362	257	364	125	286	66
335	350	38	361	83	134	178	214	179	91	497	367	85	172
132	366	45	289	423	356	180	80	258	248	376	154	403	272
141	474	69	328	108	375	523	388	343	344	369	167	111	358
401	52	514	301	117	696	174	73	274	126	165	536	325	332
415	86	270	195	446	311	266	146	148	547	107	512	341	187
101	259	251	161	419	58	41	300	37	337	56	253	119	273
377	1799	336	284	152	143	23	378	351	338	384	121	190	410
206	68	61	554	383	55	544	518	96	34	70	189	194	322
171	346	475	82	537	110	321	524	485	331	166	496	269	495
452	319	340	47	314	205	487	553	186	530	567	288	54	292
368	279	112	353	329	237	191	488	53	327	105	348	230	276
575	434	355	220	235	307	123	78	129	360	473	204	542	461
310	181	156	551	104	347	216	201	93	478	138	386	302	562
557	309	136	79	323	221	120	476	173	285	127	428	371	405
97	2193	532	381	513	256	106	170	184	333	57	424	374	334
159	569	275	158	100	543	535	558	412	447	295	245	382	168
46	238	49	62	342	305	387	252	122	541	503	81	298	113
150	200	157	531	306	404	546	316	1648	88	60	330	128	222

471	231	282	135	228	176	137	278	224	271	160	239	75	455
268	162	450	324	315	264	246	400	296	326	215	1561	261	240
133	297	185	392	522	247	114	449	145	439	115	564	581	277
748	563	533	320	396	571	287	2627	634	290	51	317	291	50
468	1558	653	582	491	500	99	620	293	1654	102	510	486	299
548	1667	92	304	493	84	430	577	198	4672	692	90	451	226
411	707	244	281	229	232	385	463	462	492	549	208	490	339
139	241	579	820	303	313	489	806	645	283	2262	432	131	580
263	234	149	689	210	213	904	479	250	448	391	769	5507	249
431	153	507	1368	724	520	2633	397	559	212	1467	1750	499	1572
572	521	526	262	511	209	442	494	1013	573	142	509	435	227
921	406	414	480	560	502	236	243	454	393	1517	140	482	254
398	433	501	519	625	1848	895	856	529	534	202	574	2257	686
2277	613	540	399	217	420	565	438	1436	1316	3497	3246	727	465
402	715	506	550	611	997	555	3338	422	225	987	1004	525	436
578	629	576	860	814	775	714	429	763	456	1000	749	233	1699
1651	267	2454	443	561	516	570	1390	484	1831	417	211	1133	477
242	864	223	1035	615	1331	805	453	732	458	568	203	265	481
508	1674	1676	545	413	444	426	407	1846	1008	2576	606	1312	5425
694	3545	1291	604	2272	416	1303	859	505	5483	441	472	408	498
767	691	641	409	1151	556	825	869	1320	2198	218	742	703	734
459	698	2234	999	717	2950	1072	517	712	504	797	623	609	765
1091	445	690	1844	1886	395	848	680	528	1070	427	5436	3978	738
1601	1010	538	552	729	610	721	515	1307	3864	3039	990	566	608
873	5484	2273	601	1005	778	1318]							

-----  
-----  
Unique values of service with dtype object have total values 13 -> [ '-' 'dns' 'ftp-data' 'smtp' 'http' 'ssh' 'pop3' 'ftp' 'ssl' 'dhcp' 'snmp' 'irc' 'radius' ]  
-----

-----  
-----  
Unique values of Sload with dtype float64 have total values 1100258 -> [ 23 044.90039 36753.40625 10678. .... 168178.8125 1629641.375 1806133.75 ]  
-----

-----  
-----  
Unique values of Dload with dtype float64 have total values 1173739 -> [ 1.20 551172e+03 3.95277305e+04 6.05617871e+03 ... 3.22186325e+06 3.75048625e+06 4.07242275e+06 ]  
-----

-----  
-----  
Unique values of Spkts with dtype int64 have total values 933 -> [ 6 1  
6 4 2 40 42 8 44 34 36 54 330  
12 236 14 66 122 10 24 28 230 56 48 60  
46 18 52 90 32 20 424 84 22 232 38 360  
96 72 64 296 358 62 228 78 215 226 454 1  
50 244 124 462 222 68 308 26 326 346 116 100  
434 162 238 198 58 442 234 106 70 364 332 98  
110 342 368 456 235 340 138 88 126 338 334 17  
310 5 146 73 168 366 86 446 352 104 102 45  
328 420 626 92 324 57 460 80 336 108 190 612  
314 196 356 142 224 344 266 437 74 0 220 134  
31 51 337 464 240 452 30 214 302 448 174 662  
436 312 49 127 112 347 242 444 354 187 370 23  
427 55 440 350 82 426 128 195 1458 634 120 170  
172 3 438 684 348 401 374 362 1248 209 130 176  
292 156 27 349 94 9 140 91 343 117 158 620  
204 218 212 652 564 418 458 494 425 11 294 178  
132 118 463 664 192 555 660 150 450 408 592 59 ]  
-----

208	622	322	646	668	217	37	144	325	189	568	333
384	318	154	67	624	433	164	316	200	616	392	323
216	21	305	47	306	15	210	508	19	191	202	223
155	606	13	290	307	680	111	188	472	119	76	321
182	136	161	329	372	29	900	430	432	576	213	250
148	199	658	300	457	656	114	25	428	43	152	363
41	590	357	194	636	476	610	670	625	166	71	160
258	241	260	33	5974	500	678	542	705	614	309	548
207	496	572	320	550	505	399	388	186	400	341	488
246	599	644	627	632	412	264	466	7	61	654	139
345	451	600	335	87	206	404	331	169	35	227	265
315	422	53	524	416	672	4410	596	123	411	197	650
311	439	386	304	584	251	393	231	526	248	3086	180
89	378	373	298	435	414	39	628	562	602	201	648
115	613	640	429	498	353	486	618	149	694	394	615
293	135	1686	339	221	121	578	682	484	638	574	666
502	390	295	406	619	445	75	976	784	79	419	582
510	389	653	359	277	69	490	184	598	674	113	474
261	63	5376	268	319	193	604	398	109	282	256	588
284	313	383	655	492	65	289	77	417	482	171	415
623	382	1940	504	233	468	538	5646	732	288	376	3882
211	970	153	649	380	686	103	396	361	676	630	786
327	566	506	540	286	262	107	270	185	254	143	274
219	410	97	371	317	81	299	1630	413	585	351	99
252	403	441	532	4806	713	355	522	480	7340	280	478
263	1142	387	642	708	487	291	512	611	367	275	276
633	278	579	544	402	365	95	560	818	776	514	706
409	272	273	554	151	545	4458	175	205	536	470	3074
203	9392	580	586	141	303	534	556	159	225	243	530
607	1874	593	6118	1254	5966	6836	283	4608	2922	1422	1704
1006	1268	4018	1159	247	105	2462	179	101	173	528	5370
181	443	2706	531	147	395	5386	369	730	2700	10646	375
4360	558	1233	954	629	1604	518	603	5179	516	1256	704
423	710	6332	397	521	722	719	570	297	229	83	1770
5062	617	407	546	525	183	85	826	1642	285	775	493
605	608	800	431	177	2734	391	798	520	2446	790	455
379	405	594	645	253	513	764	269	754	1664	385	663
5494	145	6584	692	565	421	1113	167	447	2066	449	601
267	381	5404	621	690	816	133	583	822	742	259	882
541	93	515	2740	794	5402	752	1026	245	1174	715	702
5418	643	740	2004	509	609	271	4316	591	1044	978	1138
696	597	557	3262	4404	163	1388	7040	4812	7684	6820	552
7226	639	675	581	5184	287	2676	377	1096	998	301	499
1624	5666	4920	681	281	700	689	2750	595	237	637	792
641	6680	1870	1872	688	575	4094	501	1068	788	631	8324
257	760	3960	165	4212	1600	952	804	453	4624	930	8669
4040	2730	2712	762	279	3760	6874	2854	7252	239	772	9492
635	2478	1544	936	3638	758	589	1832	5652	8424	3684	6778
1708	573	547	5422	836	7194	1866	577	10200	716	5442	549
4370	4174	511	255	5382	129	6430	2698	157	4132	1012	1072
810	4808	1194	806	249	661	1074	3974	1638	5424	2978	9616
2722	738	7990	1082	5466	559	714	2014	864	695	941	964
7908	1632	527	3758	5532	890	2494	667	1154	832	840	1816
4938	720	459	4144	665	9416	469	6974	659	797	8070	6954
2690	4694	5686	137	698	4026	1118	5196	6616	3818	6262	898
2228	553	567	1062	924	561	782	2578	6776	467	651	3822
131	744	1892	571	2320	9094	749	1622	465	125	8882	5444
2720	5412	2726	4232	491	1522	5060	1192	3844	1110	2436	1488
1456	5392	687	734	1876	5416	846	711	1834	7660	1178	912
1246	1634	1228	587	5364	1654	922	677	5702	5708	4450	854

```
1500 7292 523 1596 1250 820 1430 5254 4416]
```

```
-----  
-----  
Unique values of Dpkts with dtype int64 have total values 1255 -> [ 2 18  
4 ... 1560 2642 890]
```

```
-----  
-----  
Unique values of swin with dtype int64 have total values 36 -> [255 0 168  
70 1 179 188 45 43 210 46 61 156 232 167 87 160 202  
31 203 192 42 99 76 67 92 52 103 172 134 154 38 245 5 14 145]
```

```
-----  
-----  
Unique values of dwin with dtype int64 have total values 32 -> [255 0 37  
209 125 35 137 1 171 40 229 164 236 81 77 70 91 224  
27 43 33 160 253 46 48 197 244 192 108 12 90 175]
```

```
-----  
-----  
Unique values of stcpb with dtype int64 have total values 1429571 -> [ 76193  
4099 734569334 0 ... 1419799572 2097393639 1097293365]
```

```
-----  
-----  
Unique values of dtcpb with dtype int64 have total values 1429031 -> [389336  
5633 2907227880 0 ... 3566126516 4247267529 3252878568]
```

```
-----  
-----  
Unique values of smeansz with dtype int64 have total values 1415 -> [ 164  
96 134 ... 1471 1415 1439]
```

```
-----  
-----  
Unique values of dmeansz with dtype int64 have total values 1417 -> [ 43  
91 76 ... 1385 1417 1479]
```

```
-----  
-----  
Unique values of trans_depth with dtype int64 have total values 14 -> [ 0  
1 2 155 131 3 39 4 9 5 163 172 80 8]
```

```
-----  
-----  
Unique values of res_bdy_len with dtype int64 have total values 2907 -> [  
0 83 524288 ... 2501 3062 22545]
```

```
-----  
-----  
Unique values of Sjit with dtype float64 have total values 1381001 -> [3661.  
562382 1385.62388 116.927883 ... 22.480384 17.05975  
59.343892]
```

```
-----  
-----  
Unique values of Djit with dtype float64 have total values 1424689 -> [ 0.  
35.106383 123.543578 ... 14.605255 21.052982 1.359939]
```

```
-----  
-----  
Unique values of Stime with dtype int64 have total values 85348 -> [14242243  
69 1424252728 1421934312 ... 1424219165 1424219034 1424219062]
```

```
-----  
-----  
Unique values of Ltime with dtype int64 have total values 85361 -> [14242243  
69 1424252728 1421934313 ... 1424219108 1424219035 1424219062]
```

```
-----  
-----  
Unique values of Sintpkt with dtype float64 have total values 975275 -> [57.  
0712 20.925067 82.683 ... 2.983349 0.322291 0.286682]
```

```
-----  
Unique values of Dintpkt with dtype float64 have total values 960353 -> [2.0  
0.00000e-03 1.8459766e+01 8.7363000e+01 ... 1.3073930e+00 5.9233821e+01  
2.8471330e+00]  
-----  
Unique values of tcprtt with dtype float64 have total values 76931 -> [0.257  
434 0.000596 0. ... 0.133212 0.253942 0.20435 ]  
-----  
Unique values of synack with dtype float64 have total values 66997 -> [0.182  
93 0.000494 0. ... 0.054071 0.167714 0.11567 ]  
-----  
Unique values of ackdat with dtype float64 have total values 60844 -> [0.074  
504 0.000102 0. ... 0.013522 0.026766 0.08868 ]  
-----  
Unique values of is_sm_ips_ports with dtype int64 have total values 2 -> [0  
1]  
-----  
Unique values of ct_state_ttl with dtype int64 have total values 7 -> [3 0 2  
1 6 5 4]  
-----  
Unique values of ct_flw_http_mthd with dtype float64 have total values 16 ->  
[ 0. 1. 6. 4. 3. 5. 14. 16. 9. 2. 10. 8. 30. 36. 12. 25.]  
-----  
Unique values of is_ftp_login with dtype float64 have total values 4 -> [0.  
1. 4. 2.]  
-----  
Unique values of ct_ftp_cmd with dtype object have total values 13 -> [' ' 0  
1 '1' 2 '0' '4' 5 3 4 6 8 '2']  
-----  
Unique values of ct_srv_src with dtype int64 have total values 65 -> [10 14  
9 33 2 5 3 1 35 4 25 36 12 24 6 44 26 31 27 51 8 7 18 21  
30 41 13 11 16 15 57 39 47 46 32 59 34 28 22 17 23 42 43 37 29 45 20 40  
38 19 49 60 55 48 50 54 53 58 52 66 56 64 63 61 67]  
-----  
Unique values of ct_srv_dst with dtype int64 have total values 66 -> [ 8 6  
16 33 2 5 1 35 14 7 25 36 24 3 9 4 26 31 27 51 10 18 21 30  
41 11 12 15 17 57 19 39 13 47 46 32 59 34 20 28 22 23 44 42 43 37 29 45  
40 38 49 60 55 48 50 54 53 58 52 66 56 64 62 63 61 67]  
-----  
Unique values of ct_dst_ltm with dtype int64 have total values 62 -> [ 4 8  
7 17 5 3 2 35 1 22 6 24 32 23 21 19 25 51 18 16 9 31 10 15  
11 14 34 39 12 26 42 30 59 13 20 47 43 40 45 36 44 60 37 27 33 48 49 28  
38 54 41 29 46 58 52 57 50 56 53 55 61 67]  
-----  
Unique values of ct_src_ltm with dtype int64 have total values 62 -> [ 5 12  
6 17 4 2 1 35 7 3 22 24 18 8 11 9 23 25 19 51 21 16 31 10  
15 14 34 39 26 42 40 59 20 44 47 32 13 30 29 45 36 60 37 43 28 33 48 49
```

```

38 54 27 41 46 58 52 57 50 56 53 55 61 67]
-----
-----
Unique values of ct_src_dport_ltm with dtype int64 have total values 62 -> [
3 5 1 17 35 7 22 6 24 2 4 23 25 19 51 18 16 31 9 15 8 34 39 12
26 42 30 10 59 20 44 21 43 32 14 40 13 45 33 60 37 11 27 48 49 28 38 36
54 41 29 46 58 47 52 57 50 56 53 55 61 67]
-----
-----
Unique values of ct_dst_sport_ltm with dtype int64 have total values 55 -> [
1 17 18 7 3 22 24 4 23 21 19 13 27 16 2 12 9 6 8 10 30 11 20 5
25 15 14 54 39 33 58 37 31 48 52 29 57 28 41 60 32 35 26 51 47 49 50 38
46 56 40 36 44 53 34]
-----
-----
Unique values of ct_dst_src_ltm with dtype int64 have total values 65 -> [
7
6 3 33 1 2 35 14 25 36 24 9 26 4 31 27 51 5 18 21 30 41 16 11
12 15 8 57 39 13 19 47 46 32 59 10 34 28 22 23 44 42 43 17 37 29 45 20
40 38 49 60 55 48 50 54 53 58 52 66 56 65 63 61 67]
-----
-----
Unique values of Label with dtype int64 have total values 2 -> [0 1]
-----
```



## Inference:

- Observed that some values are repeated due to their wrong format , for example in column 'ct\_ftp\_cmd' the unique values are [0 ' '1' 1 '0' 2 8 '2' 4 5 3 6 '4'] in which we see that same values reapeated and treated as differently
- In column 'is\_ftp\_login' unique values are [0. 1. 2. 4.] which is not correct because according to given NB15\_features.csv this column is a binary column

## Handling ct\_ftp\_cmd

```
In [ ]: # Function to clean and convert to numeric
def clean_and_convert_ct_ftp_cmd(df, column):
    df[column] = df[column].astype('str').replace(' ', '0')
    df[column] = pd.to_numeric(df[column], errors='coerce').fillna(0).astype('int64')
    return df
```

```
In [ ]: # Apply to 'ct_ftp_cmd'
train_df = clean_and_convert_ct_ftp_cmd(train_df, 'ct_ftp_cmd')
```

```
In [ ]: # Verify
print(f"Unique values of 'ct_ftp_cmd' after processing:", train_df['ct_ftp_cmd'])

Unique values of 'ct_ftp_cmd' after processing: [0 1 2 4 5 3 6 8]
```

## Handling is\_ftp\_login

```
In [ ]: # Function to ensure binary column
def convert_to_binary(df, column):
    df[column] = pd.to_numeric(df[column], errors='coerce').fillna(0).astype('int64')
    df[column] = (df[column] > 0).astype('int64')
    return df

In [ ]: # Apply to 'is_ftp_login'
train_df = convert_to_binary(train_df, 'is_ftp_login')

In [ ]: # Verify
print(f"Unique values of 'is_ftp_login' after processing:", train_df['is_f
Unique values of 'is_ftp_login' after processing: [0 1]
```

## Handling sport and dsport

```
In [ ]: # Function to convert to numeric and handle NaNs
def convert_sport_dsport(df, column):
    df[column] = df[column].astype('str')
    df[column] = pd.to_numeric(df[column], errors='coerce').fillna(0).astype('int64')
    return df

In [ ]: # Apply to 'sport' and 'dsport'
train_df = convert_sport_dsport(train_df, 'sport')
train_df = convert_sport_dsport(train_df, 'dsport')

In [ ]: # Verify
print(f"Unique values of 'sport' after processing:", train_df['sport'].unique())
print(f"Unique values of 'dsport' after processing:", train_df['dsport'].unique())

Unique values of 'sport' after processing: [57672 38052 42911 ... 922
19 74]
Unique values of 'dsport' after processing: [ 3260 6881 38558 ... 36581 59
554 9856]

In [ ]: # Re-check if 'sport' and 'dport' are numerical
if train_df['sport'].dtype in ['int64', 'float64']:
    print("'sport' column is numerical.")
else:
    print("'sport' column is not numerical.")

if train_df['dsport'].dtype in ['int64', 'float64']:
    print("'dsport' column is numerical.")
else:
    print("'dsport' column is not numerical.")

'sport' column is numerical.
'dsport' column is numerical.

In [ ]: print(f"Unique values of 'Label':", train_df['Label'].unique())

Unique values of 'Label': [0 1]
```

## 3 | Exploratory Data Analysis (EDA)

## Checking for outliers

```
In [ ]: numerical_columns = train_df.select_dtypes(include=['float64', 'int64']).count()
len(numerical_columns)
```

```
Out[ ]: 43
```

```
In [ ]: # Print all numerical columns to verify
print("Numerical columns:", numerical_columns)
```

```
Numerical columns: ['sport', 'dsport', 'dur', 'sbytes', 'dbytes', 'sttl', 'dttl', 'sloss', 'dloss', 'Sload', 'Dload', 'Spkts', 'Dpkts', 'swin', 'dwi_n', 'stcpb', 'dtcpb', 'smeansz', 'dmeansz', 'trans_depth', 'res_bdy_len', 'Sjit', 'Djit', 'Stime', 'Ltime', 'Sintpkt', 'Dintpkt', 'tcprrtt', 'synack', 'ackdat', 'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_mthd', 'is_ftp_login', 'ct_ftp_cmd', 'ct_srv_src', 'ct_srv_dst', 'ct_dst_ltm', 'ct_src_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'Label']
```

```
In [ ]: # Check the data types of all columns in your notebook
print(train_df.dtypes)
```

```
# List all columns to compare with the other notebook
all_columns = train_df.columns.tolist()
print("All columns in your notebook:", all_columns)
```

```
# Check numerical columns
numerical_columns = train_df.select_dtypes(include=['float64', 'int64']).count()
print("Numerical columns in your notebook:", numerical_columns)
```

```

Name
sport           int64
dsport          int64
proto           object
state           object
dur             float64
sbytes          int64
dbytes          int64
sttl            int64
dttl            int64
sloss           int64
dloss           int64
service         object
Sload           float64
Dload           float64
Spkts           int64
Dpkts           int64
swin            int64
dwin            int64
stcpb           int64
dtcpb           int64
smeansz         int64
dmeansz         int64
trans_depth     int64
res_bdy_len    int64
Sjit            float64
Djit             float64
Stime           int64
Ltime            int64
Sintpkt         float64
Dintpkt         float64
tcprtt          float64
synack          float64
ackdat          float64
is_sm_ips_ports int64
ct_state_ttl    int64
ct_flw_http_mthd float64
is_ftp_login    int64
ct_ftp_cmd      int64
ct_srv_src      int64
ct_srv_dst      int64
ct_dst_ltm      int64
ct_src_ltm      int64
ct_src_dport_ltm int64
ct_dst_sport_ltm int64
ct_dst_src_ltm   int64
Label            int64
dtype: object

All columns in your notebook: ['sport', 'dsport', 'proto', 'state', 'dur', 'sbytes', 'dbytes', 'sttl', 'dttl', 'sloss', 'dloss', 'service', 'Sload', 'Dload', 'Spkts', 'Dpkts', 'swin', 'dwin', 'stcpb', 'dtcpb', 'smeansz', 'dmeansz', 'trans_depth', 'res_bdy_len', 'Sjit', 'Djit', 'Stime', 'Ltime', 'Sintpkt', 'Dintpkt', 'tcprtt', 'synack', 'ackdat', 'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_mthd', 'is_ftp_login', 'ct_ftp_cmd', 'ct_srv_src', 'ct_srv_dst', 'ct_dst_ltm', 'ct_src_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'Label']

Numerical columns in your notebook: ['sport', 'dsport', 'dur', 'sbytes', 'dbytes', 'sttl', 'dttl', 'sloss', 'dloss', 'Sload', 'Dload', 'Spkts', 'Dpkts', 'swin', 'dwin', 'stcpb', 'dtcpb', 'smeansz', 'dmeansz', 'trans_depth', 'res_bdy_len', 'Sjit', 'Djit', 'Stime', 'Ltime', 'Sintpkt', 'Dintpkt', 'tcp

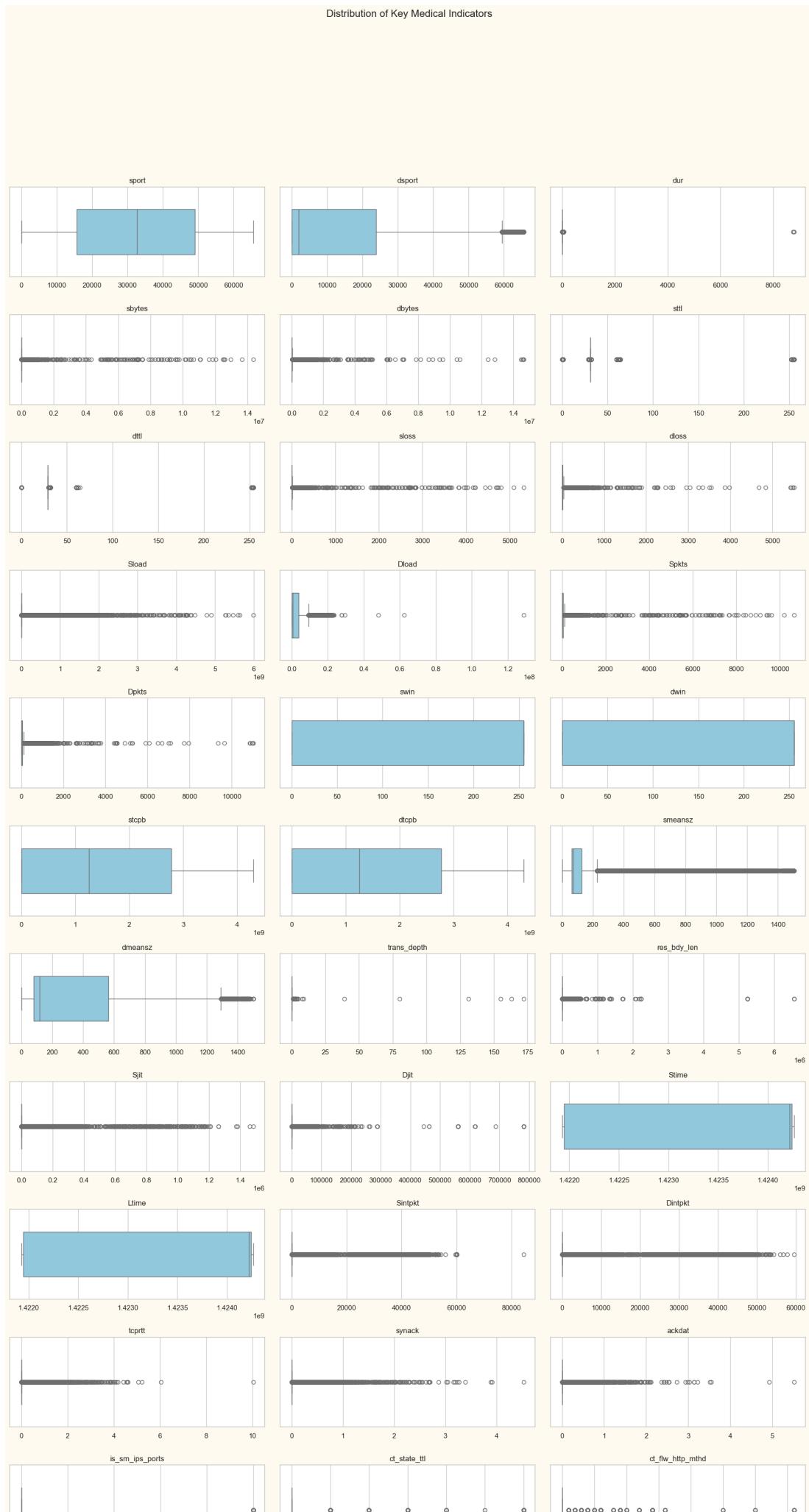
```

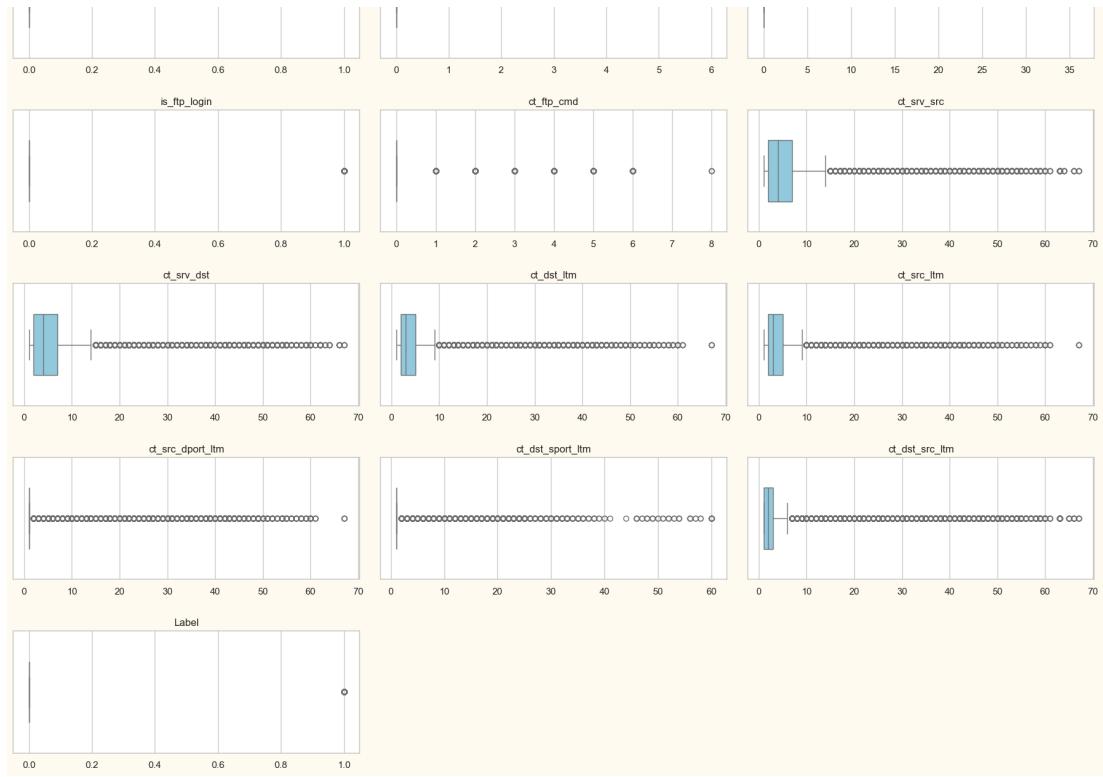
```
rtt', 'synack', 'ackdat', 'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_m  
thd', 'is_ftp_login', 'ct_ftp_cmd', 'ct_srv_src', 'ct_srv_dst', 'ct_dst_lt  
m', 'ct_src_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm',  
'Label']
```

```
In [ ]: print(f"Unique values of 'Label':", train_df['Label'].unique())
```

```
Unique values of 'Label': [0 1]
```

```
In [ ]: import math  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Number of numerical columns  
num_columns = len(numerical_columns)  
  
# Calculate the number of rows and columns needed for the subplots  
num_cols = 3 # Fixed number of columns  
num_rows = math.ceil(num_columns / num_cols) # Calculate rows needed  
  
plt.figure(figsize=(18, num_rows * 3))  
  
sns.set_palette("husl")  
sns.set_theme(style="whitegrid")  
  
for i, col in enumerate(numerical_columns, 1):  
    plt.subplot(num_rows, num_cols, i)  
    sns.boxplot(x=train_df[col], color='skyblue', width=0.5)  
    plt.title(col)  
    plt.xlabel("")  
  
plt.suptitle("Distribution of Key Medical Indicators", y=1.02, fontsize=16)  
plt.tight_layout(rect=[0, 0, 1, 0.95])  
plt.show()
```





```
In [ ]: # Extract numerical columns
numerical_columns = train_df.select_dtypes(include=['float64', 'int64']).columns

# Columns to exclude
exclude_columns = ['sport', 'swim', 'dwim', 'stcpb', 'dtcpb', 'Stime', 'Ltime']

# Filter out the columns to exclude
numerical_columns = [col for col in numerical_columns if col not in exclude_columns]
```

```
In [ ]: # for col in numerical_columns:
#     median_value = train_df[col].median()
#     lower_bound = train_df[col].quantile(0.25) - 1.5 * (train_df[col].quantile(0.75) - train_df[col].quantile(0.25))
#     upper_bound = train_df[col].quantile(0.75) + 1.5 * (train_df[col].quantile(0.75) - train_df[col].quantile(0.25))
#     train_df[col] = train_df[col].apply(lambda x: median_value if x < lower_bound else upper_bound if x > upper_bound else x)
```

```
In [ ]: train_df
```

Out[ ]:	Name	sport	dsport	proto	state	dur	sbytes	dbbytes	sttl	dttl	sloss
	<b>0</b>	57672	3260	tcp	CON	0.285356	986	86	62	252	2
	<b>1</b>	38052	6881	tcp	FIN	0.314311	1540	1644	31	29	4
	<b>2</b>	42911	38558	udp	CON	0.301180	536	304	31	29	0
	<b>3</b>	47439	53	udp	INT	0.000009	114	0	254	0	0
	<b>4</b>	61544	53	udp	CON	0.001079	146	178	31	29	0
	...	...	...	...	...	...	...	...	...	...	...
	<b>2540035</b>	29290	143	tcp	FIN	0.031870	7820	15060	31	29	30
	<b>2540037</b>	18378	17406	udp	CON	0.035147	528	304	31	29	0
	<b>2540038</b>	34415	5190	tcp	FIN	0.008119	1920	4312	31	29	6
	<b>2540039</b>	56352	53	udp	CON	0.001047	130	162	31	29	0
	<b>2540040</b>	25527	6881	tcp	FIN	0.013106	1540	1644	31	29	4

2048194 rows × 46 columns

In [ ]: `print(f"Unique values of 'Label':", train_df['Label'].unique())`

Unique values of 'Label': [0 1]

## Visualising Data Distribution and Skewness

```
# # Set the figure size and arrange plots horizontally in pairs
# numerical_features = train_df.select_dtypes(include=['number']).columns
# num_plots = len(numerical_features)
# num_rows = (num_plots + 1) // 2 # Ensure enough rows to accommodate all
# fig, axes = plt.subplots(nrows=num_rows, ncols=2, figsize=(14, 7*num_rows))

# # Flatten the axes array for easy indexing
# axes = axes.flatten()

# # Loop through the selected columns and create histograms with density
# for i, col in enumerate(numerical_features):
#     sns.histplot(data=train_df, x=col, kde=True, ax=axes[i])
#     axes[i].set_title(f'Histogram with Density for {col}')
#     axes[i].set_xlabel(col)
#     axes[i].set_ylabel('Density')

#     # Calculate skewness
#     skewness = skew(train_df[col].dropna())
#     axes[i].text(0.5, 0.95, f'Skewness: {skewness:.2f}', horizontalalignment='center')

# # Remove any empty subplots if the number of features is odd
# if num_plots % 2 != 0:
#     fig.delaxes(axes[-1])

# plt.tight_layout()
```

```
# plt.show()
```

- $|\text{Skewness}| < 1$ : The distribution is approximately symmetric.
- $|\text{Skewness}| > 1$ : The distribution is highly skewed.
- $|\text{Skewness}|$  between 1 and 2: Moderately skewed distribution.

```
In [ ]: def transform(x):
    eps = 1e-5
    numerical_columns = x.select_dtypes(include=['float64', 'int64']).columns

    # Exclude the 'Label' column
    numerical_columns = [col for col in numerical_columns if col != 'Label']

    for col in numerical_columns:
        skewness = skew(x[col])
        if skewness > 0: # Positive skew
            x[col] = np.log(x[col] + eps)
        elif skewness < 0: # Negative skew
            x[col] = np.log(np.max(x[col] + eps) - x[col] + eps)
        else: # Symmetric or zero skew
            x[col] = x[col] # No transformation needed

    return x
```

```
In [ ]: # transform(train_df)
# train_df
```

## Visualising Class Distribution

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

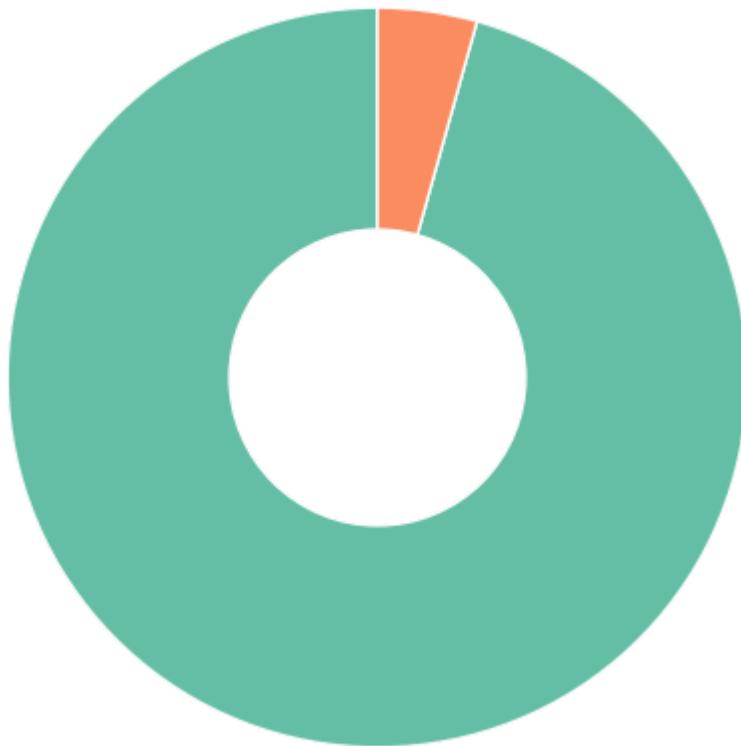
def pie_bar_plot(df, col):
    plt.figure(figsize=(10, 6))

    # Extract value counts for the specified column
    value_counts = df[col].value_counts().sort_index()

    ax1 = value_counts
    plt.title(f"Distribution by {col}", fontweight="black", size=14, pad=1)
    colors = sns.color_palette('Set2', len(ax1))
    plt.pie(ax1.values, labels=None, autopct="", startangle=90, colors=colors)
    center_circle = plt.Circle((0, 0), 0.4, fc='white')
    fig = plt.gcf()
    fig.gca().add_artist(center_circle)
    plt.show()
```

```
In [ ]: pie_bar_plot(train_df, 'Label')
```

## Distribution by Label



```
In [ ]: # Print class distribution before resampling
print("Before resampling:", train_df['Label'].value_counts())
print()
```

```
Before resampling: Label
0    1959348
1     88846
Name: count, dtype: int64
```

## 4 | Preprocessing

### Feature Engineering

```
In [ ]: # experimenting with only keeping selective features : dur, proto, sport, ...
# selected_columns = ['dur', 'proto', 'sport', 'dsport', 'state', 'service']
# train_df = train_df[selected_columns]

train_df.head()
```

	Name	sport	dsport	proto	state	dur	sbytes	dbytes	sttl	dttl	sloss	d
0	57672	3260	tcp	CON	0.285356	986	86	62	252	2		
1	38052	6881	tcp	FIN	0.314311	1540	1644	31	29	4		
2	42911	38558	udp	CON	0.301180	536	304	31	29	0		
3	47439	53	udp	INT	0.000009	114	0	254	0	0		
4	61544	53	udp	CON	0.001079	146	178	31	29	0		

```
In [ ]: def generate_features(df):
    # Duration
    df['duration'] = df['Ltime'] - df['Stime']

    # Ratios
    df['byte_ratio'] = df['sbytes'] / (df['dbytes'] + 1)
    df['pkt_ratio'] = df['Spkts'] / (df['Dpkts'] + 1)
    df['load_ratio'] = df['Sload'] / (df['Dload'] + 1)
    df['jit_ratio'] = df['Sjit'] / (df['Djit'] + 1)
    df['inter_pkt_ratio'] = df['Sintpkt'] / (df['Dintpkt'] + 1)
    df['tcp_setup_ratio'] = df['tcprrt'] / (df['synack'] + df['ackdat'] + 1)

    # Aggregate Features
    df['total_bytes'] = df['sbytes'] + df['dbytes']
    df['total_pkts'] = df['Spkts'] + df['Dpkts']
    df['total_load'] = df['Sload'] + df['Dload']
    df['total_jitter'] = df['Sjit'] + df['Djit']
    df['total_inter_pkt'] = df['Sintpkt'] + df['Dintpkt']
    df['total_tcp_setup'] = df['tcprrt'] + df['synack'] + df['ackdat']

    # Interaction Features
    df['byte_pkt_interaction_src'] = df['sbytes'] * df['Spkts']
    df['byte_pkt_interaction_dst'] = df['dbytes'] * df['Dpkts']
    df['load_jit_interaction_src'] = df['Sload'] * df['Sjit']
    df['load_jit_interaction_dst'] = df['Dload'] * df['Djit']
    df['pkt_jit_interaction_src'] = df['Spkts'] * df['Sjit']
    df['pkt_jit_interaction_dst'] = df['Dpkts'] * df['Djit']

    # Statistical Features
    df['mean_pkt_size'] = df['smeansz'] + df['dmeansz']
    df['tcp_seq_diff'] = df['stcpb'] - df['dtcpb']

    return df
```

```
In [ ]: generate_features(train_df)
train_df
```

Out[ ]:	Name	sport	dsport	proto	state	dur	sbytes	dbbytes	sttl	dttl	sloss
	0	57672	3260	tcp	CON	0.285356	986	86	62	252	2
	1	38052	6881	tcp	FIN	0.314311	1540	1644	31	29	4
	2	42911	38558	udp	CON	0.301180	536	304	31	29	0
	3	47439	53	udp	INT	0.000009	114	0	254	0	0
	4	61544	53	udp	CON	0.001079	146	178	31	29	0
	...	...	...	...	...	...	...	...	...	...	...
	2540035	29290	143	tcp	FIN	0.031870	7820	15060	31	29	30
	2540037	18378	17406	udp	CON	0.035147	528	304	31	29	0
	2540038	34415	5190	tcp	FIN	0.008119	1920	4312	31	29	6
	2540039	56352	53	udp	CON	0.001047	130	162	31	29	0
	2540040	25527	6881	tcp	FIN	0.013106	1540	1644	31	29	4

2048194 rows × 67 columns

```
In [ ]: #Checking the categorical columns
cat_columns = train_df.select_dtypes(include=['O']).columns.tolist()
cat_columns
```

Out[ ]: ['proto', 'state', 'service']

## Encode leftover categorical features

In this case im also encoding Label though unnecessary, but just to be sure

```
In [ ]: # Initialize LabelEncoder
# label_encoder = LabelEncoder()
label_encoder_proto = LabelEncoder()
label_encoder_state = LabelEncoder()
label_encoder_service = LabelEncoder()

# Apply LabelEncoder to each categorical feature
# train_df['Label'] = label_encoder.fit_transform(train_df['Label'])
train_df['proto'] = label_encoder_proto.fit_transform(train_df['proto'])
train_df['state'] = label_encoder_state.fit_transform(train_df['state'])
train_df['service'] = label_encoder_service.fit_transform(train_df['service'])

# Create the label mapping
# label_mapping = dict(zip(label_encoder.classes_, label_encoder.transform))
proto_mapping = dict(zip(label_encoder_proto.classes_, label_encoder_proto.transform))
state_mapping = dict(zip(label_encoder_state.classes_, label_encoder_state.transform))
service_mapping = dict(zip(label_encoder_service.classes_, label_encoder_service.transform))

# print("Label Mapping:")
# print(label_mapping)
```

```

print("Proto Mapping:")
print(proto_mapping)

print("State Mapping:")
print(state_mapping)

print("Service Mapping:")
print(service_mapping)

```

Proto Mapping:

```
{'3pc': 0, 'a/n': 1, 'aes-sp3-d': 2, 'any': 3, 'argus': 4, 'aris': 5, 'arp': 6, 'ax.25': 7, 'bbn-rcc': 8, 'bna': 9, 'br-sat-mon': 10, 'cbt': 11, 'cf tp': 12, 'chaos': 13, 'compaq-peer': 14, 'cphb': 15, 'cpnx': 16, 'crtp': 17, 'crudp': 18, 'dcn': 19, 'ddp': 20, 'ddx': 21, 'dgp': 22, 'egp': 23, 'eig rp': 24, 'emcon': 25, 'encap': 26, 'esp': 27, 'etherip': 28, 'fc': 29, 'fir e': 30, 'ggp': 31, 'gmtcp': 32, 'gre': 33, 'hmp': 34, 'i-nlsp': 35, 'iatp': 36, 'ib': 37, 'icmp': 38, 'idpr': 39, 'idpr-cmtp': 40, 'idrp': 41, 'ifmp': 42, 'igmp': 43, 'igp': 44, 'il': 45, 'ip': 46, 'ipcomp': 47, 'ipcv': 48, 'ipip': 49, 'iplt': 50, 'ipnip': 51, 'ippc': 52, 'ipv6': 53, 'ipv6-frag': 54, 'ipv6-no': 55, 'ipv6-opt': 56, 'ipv6-route': 57, 'ipx-n-ip': 58, 'irtp': 59, 'isis': 60, 'iso-ip': 61, 'iso-tp4': 62, 'kryptolan': 63, 'l2tp': 64, 'larp': 65, 'leaf-1': 66, 'leaf-2': 67, 'merit-inp': 68, 'mfe-nsp': 69, 'mhr p': 70, 'micp': 71, 'mobile': 72, 'mtp': 73, 'mux': 74, 'narlp': 75, 'nettbl t': 76, 'nsfnet-igp': 77, 'nvp': 78, 'ospf': 79, 'pgm': 80, 'pim': 81, 'pipe': 82, 'pnni': 83, 'pri-enc': 84, 'prm': 85, 'ptp': 86, 'pup': 87, 'pvp': 88, 'qnx': 89, 'rdp': 90, 'rsvp': 91, 'rtp': 92, 'rvd': 93, 'sat-expak': 94, 'sat-mon': 95, 'scocopmce': 96, 'scps': 97, 'sctp': 98, 'sdrp': 99, 'secu re-vmtp': 100, 'sep': 101, 'skip': 102, 'sm': 103, 'smp': 104, 'snp': 105, 'sprite-rpc': 106, 'sps': 107, 'srp': 108, 'st2': 109, 'stp': 110, 'sun-n d': 111, 'swipe': 112, 'tcf': 113, 'tcp': 114, 'tlsp': 115, 'tp++': 116, 'trunk-1': 117, 'trunk-2': 118, 'ttp': 119, 'udp': 120, 'udt': 121, 'unas': 122, 'uti': 123, 'vines': 124, 'visa': 125, 'vmtcp': 126, 'vrrp': 127, 'wb-expak': 128, 'wb-mon': 129, 'wsn': 130, 'xnet': 131, 'xns-idp': 132, 'xtp': 133, 'zero': 134}
```

State Mapping:

```
{'ACC': 0, 'CLO': 1, 'CON': 2, 'ECO': 3, 'ECR': 4, 'FIN': 5, 'INT': 6, 'MAS': 7, 'PAR': 8, 'REQ': 9, 'RST': 10, 'TST': 11, 'TXD': 12, 'URH': 13, 'URN': 14, 'no': 15}
```

Service Mapping:

```
{'': 0, 'dhcp': 1, 'dns': 2, 'ftp': 3, 'ftp-data': 4, 'http': 5, 'irc': 6, 'pop3': 7, 'radius': 8, 'smtp': 9, 'snmp': 10, 'ssh': 11, 'ssl': 12}
```

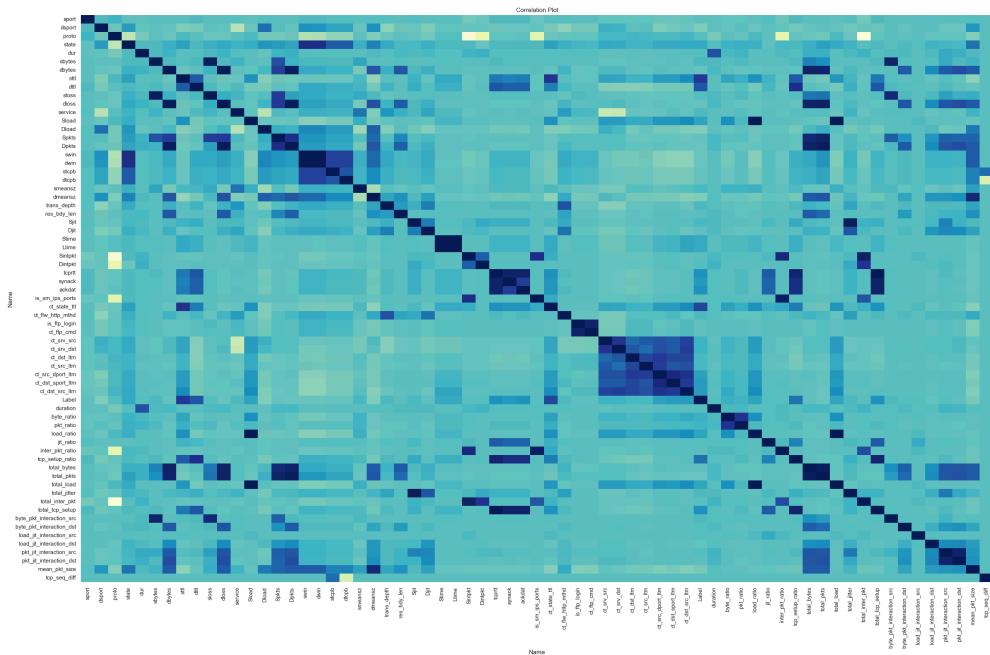
In [ ]: #Checking the categorical columns  
`cat_columns = train_df.select_dtypes(include=['O']).columns.tolist()  
cat_columns`

Out[ ]: []

## Checking Highly Correlated Features

In [ ]: `plt.figure(figsize=(40,20))  
plt.title("Correlation Plot")  
sns.heatmap(train_df.corr(), cmap='YlGnBu')`

Out[ ]: <Axes: title={'center': 'Correlation Plot'}, xlabel='Name', ylabel='Name'>



```
In [ ]: # Calculate the correlation matrix
correlation_matrix = train_df.corr()

# Set the correlation threshold (adjust if necessary)
correlation_threshold = 0.8

# Identify highly correlated features
highly_correlated_features = set()

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) >= correlation_threshold:
            feature1 = correlation_matrix.columns[i]
            feature2 = correlation_matrix.columns[j]
            highly_correlated_features.add((feature1, feature2))

# Print the highly correlated features
print("Highly correlated features:")
for feature1, feature2 in highly_correlated_features:
    print(f"{feature1} and {feature2}")
```

Highly correlated features:

```
sloss and sbytes
total_pkts and dloss
dwin and swin
ct_ftp_cmd and is_ftp_login
total_load and Sload
dwin and state
mean_pkt_size and dmeansz
total_pkts and dbytes
tcp_setup_ratio and ackdat
Ltime and Stime
inter_pkt_ratio and Sintpkt
total_tcp_setup and ackdat
total_load and load_ratio
total_inter_pkt and Sintpkt
total_pkts and total_bytes
total_inter_pkt and proto
ackdat and tcprtt
swin and state
synack and tcprtt
Dpkts and Spkts
total_bytes and Dpkts
load_ratio and Sload
tcp_setup_ratio and tcprtt
tcp_setup_ratio and synack
total_bytes and Spkts
byte_pkt_interaction_src and sloss
dloss and dbytes
Dpkts and dloss
total_tcp_setup and tcp_setup_ratio
total_bytes and dloss
pkt_jit_interaction_dst and pkt_jit_interaction_src
byte_pkt_interaction_src and sbytes
Dpkts and dbytes
ct_dst_sport_ltm and ct_src_dport_ltm
total_jitter and Sjit
total_pkts and Spkts
total_pkts and Dpkts
total_bytes and dbytes
total_tcp_setup and tcprtt
is_sm_ips_ports and Sintpkt
total_tcp_setup and synack
inter_pkt_ratio and is_sm_ips_ports
```

```
In [ ]: # Create a set of features to drop
features_to_drop = set()

# Ensure required features are preserved
required_features = {'dport', 'sport', 'state', 'proto', 'service', 'Label'}

# Drop the least important feature from each correlated pair
for feature1, feature2 in highly_correlated_features:
    if feature1 not in required_features and feature2 not in required_features:
        # Choose the feature with higher correlation to the Label for retention
        if abs(correlation_matrix.loc[feature1, 'Label']) > abs(correlation_matrix.loc[feature2, 'Label']):
            features_to_drop.add(feature2)
        else:
            features_to_drop.add(feature1)

# Drop the features from the DataFrame
```

```

train_df = train_df.drop(columns=list(features_to_drop))

# Print the remaining features
print("Remaining features after dropping highly correlated ones:")
print(train_df.columns)

```

Remaining features after dropping highly correlated ones:

Index(['sport', 'dsport', 'proto', 'state', 'dur', 'sbytes', 'sttl', 'dttl',  
'service', 'Dload', 'Dpkts', 'swin', 'stcpb', 'dtcpb', 'smeansz',  
'dmeansz', 'trans\_depth', 'res\_bdy\_len', 'Sjit', 'Djit', 'Ltime',  
'Dintpkt', 'is\_sm\_ips\_ports', 'ct\_state\_ttl', 'ct\_flw\_http\_mthd',  
'is\_ftp\_login', 'ct\_srv\_src', 'ct\_srv\_dst', 'ct\_dst\_ltm', 'ct\_src\_ltm',  
'ct\_dst\_sport\_ltm', 'ct\_dst\_src\_ltm', 'Label', 'duration', 'byte\_ratio',  
'pkt\_ratio', 'load\_ratio', 'jit\_ratio', 'tcp\_setup\_ratio',  
'byte\_pkt\_interaction\_dst', 'load\_jit\_interaction\_src',  
'load\_jit\_interaction\_dst', 'pkt\_jit\_interaction\_dst', 'tcp\_seq\_dif'],  
dtype='object', name='Name')

## Splitting the features in dependent and independent features

Split for Anomaly detection One Class SVM

```

In [ ]: # # from sklearn.model_selection import train_test_split

# # View how many rows-columns exists in the main dataset
# rows, columns = train_df.shape
# print(f"Number of rows: {rows}")
# print(f"Number of columns: {columns}")

# # x and y to be used for SVM unsupervised Learning trained in the full capacity
# # Sample 5% of the data, or about 100k entries, since its over 2 million
# sample_df, _ = train_test_split(train_df, test_size=0.95, stratify=train_df)
# sample_df.to_csv('sample_df.csv', index=False)

# x_train_ocsvm = sample_df[sample_df['Label'] == 0].drop(['Label'], axis=1)
# y_train_ocsvm = sample_df[sample_df['Label'] == 0]['Label']

# x_test_ocsvm = sample_df.drop(['Label'], axis=1)
# y_test_ocsvm = sample_df[['Label']]

# rows, columns = x_train_ocsvm.shape
# print(f"Number of rows: {rows}")
# print(f"Number of columns: {columns}")
# print("1 and 0 distribution:", y_train_ocsvm.value_counts())

# # # The above information will be used only at the OneClassSVM model training

```

```
In [ ]: train_df.shape
```

```
Out[ ]: (2048194, 44)
```

```
In [ ]: train_df['Label'].value_counts()
```

```
Out[ ]: Label
0    1959348
1     88846
Name: count, dtype: int64
```

```
In [ ]: train_df.to_csv('dataset.csv', index=False)
```

```
In [ ]: x = train_df.drop(['Label'], axis=1)
y = train_df[['Label']]

# x_train_ocsvm = train_df[train_df['Label'] == 0].drop(['Label'], axis=1)

# Filter rows where Label is 0
label_0_df = train_df[train_df['Label'] == 0]

# Randomly sample 10% of these rows
x_train_ocsvm = label_0_df.sample(frac=0.20, random_state=42)

# Drop the 'Label' column
x_train_ocsvm = x_train_ocsvm.drop(['Label'], axis=1)

# Now x_train_ocsvm contains 10% of the rows where Label is 0
x_train_ocsvm.shape
```

```
Out[ ]: (391870, 43)
```

```
In [ ]: # x = train_df.drop(['Label'], axis=1)
# y = train_df[['Label']]

x.shape
```

```
Out[ ]: (2048194, 43)
```

## Applying SMOTE to balance the unbalanced data

```
In [ ]: from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
import pandas as pd

# Define the desired number of samples for each class
desired_count = 88000

# oversample_strategy = {key: desired_count for key in y.value_counts().index}
# undersample_strategy = {key: desired_count for key in y.value_counts().index}

# Define the oversampling strategy for SMOTE
oversample_strategy = {i: desired_count for i in range(len(y.value_counts()))}

# Define the undersampling strategy for RandomUnderSampler
undersample_strategy = {i: desired_count for i in range(len(y.value_counts()))}

# Create the SMOTE and RandomUnderSampler objects
```

```

smote = SMOTE(sampling_strategy=oversample_strategy)
undersample = RandomUnderSampler(sampling_strategy=undersample_strategy)

# Combine SMOTE and RandomUnderSampler in a pipeline
pipeline = Pipeline(steps=[('smote', smote), ('undersample', undersample)])

# Print class distribution before resampling
print("Before resampling:", y.value_counts())
print()

# Apply the pipeline to resample the dataset
x_resampled, y_resampled = pipeline.fit_resample(x, y)

# Print class distribution after resampling
print("After resampling:", y_resampled.value_counts())

```

Before resampling: Label  
0 1959348  
1 88846  
Name: count, dtype: int64

After resampling: Label  
0 88000  
1 88000  
Name: count, dtype: int64

In [ ]: # x and y to be used for supervised Learning Classification

```

x = x_resampled
y = y_resampled

```

## Checking Feature Importance

### Method 1: Mutual Information Score

In [ ]: # Determine which features are discrete

```

from sklearn.feature_selection import mutual_info_classif

discrete_features = x.dtypes == int

# Function to calculate mutual information scores for feature selection
def mi_score_maker(x, y, discrete_features):
    """
    This function calculates mutual information scores for each feature in
    relative to the target variable. It helps in identifying the importance
    in predicting the target.

    Parameters:
    x (DataFrame): The feature matrix.
    y (Series): The target variable.
    discrete_features (Series): Boolean series indicating which features are
        discrete.

    Returns:
    DataFrame: A DataFrame containing features and their corresponding mutual
        information scores, sorted in descending order of the scores.
    """

```

```
# Calculate mutual information scores for each feature
scores = mutual_info_classif(x, y, discrete_features=discrete_features)

# Create a DataFrame to hold the feature names and their scores
df = pd.DataFrame({'Features': x.columns, 'Scores': scores})

# Sort the DataFrame by scores in descending order and reset the index
df = df.sort_values('Scores', ascending=False).reset_index(drop=True)

return df
```

```
In [ ]: # mi_scores = mi_score_maker(x,y.astype('float64'),discrete_features)
# mi_scores = mi_score_maker(x,y.astype('float64', 'int64'),discrete_features)

# Calculate mutual information scores
mi_scores = mi_score_maker(x, y, discrete_features)

# Display the scores
mi_scores
```

Out[ ]:	Features	Scores
<b>0</b>	sttl	0.662488
<b>1</b>	ct_state_ttl	0.644003
<b>2</b>	dttl	0.633695
<b>3</b>	sbytes	0.595352
<b>4</b>	byte_ratio	0.593600
<b>5</b>	byte_pkt_interaction_dst	0.586293
<b>6</b>	pkt_ratio	0.503880
<b>7</b>	dmeansz	0.467830
<b>8</b>	Dintpkt	0.464515
<b>9</b>	Dload	0.448338
<b>10</b>	tcp_setup_ratio	0.427092
<b>11</b>	load_ratio	0.412480
<b>12</b>	dur	0.409299
<b>13</b>	smeansz	0.406691
<b>14</b>	Dpkts	0.404009
<b>15</b>	tcp_seq_diff	0.328808
<b>16</b>	Sjit	0.283194
<b>17</b>	dsport	0.249609
<b>18</b>	Djit	0.241875
<b>19</b>	load_jit_interaction_dst	0.207029
<b>20</b>	state	0.189354
<b>21</b>	jit_ratio	0.186153
<b>22</b>	Ltime	0.179878
<b>23</b>	ct_dst_ltm	0.174385
<b>24</b>	ct_src_ltm	0.140555
<b>25</b>	pkt_jit_interaction_dst	0.139187
<b>26</b>	ct_srv_dst	0.132586
<b>27</b>	load_jit_interaction_src	0.097678
<b>28</b>	ct_srv_src	0.093678
<b>29</b>	sport	0.082218
<b>30</b>	ct_dst_src_ltm	0.072311
<b>31</b>	ct_dst_sport_ltm	0.063569
<b>32</b>	res_bdy_len	0.062311

	Features	Scores
<b>33</b>	service	0.041273
<b>34</b>	duration	0.035977
<b>35</b>	swin	0.026265
<b>36</b>	proto	0.013133
<b>37</b>	trans_depth	0.012291
<b>38</b>	ct_flw_http_mthd	0.012162
<b>39</b>	dtcpb	0.008033
<b>40</b>	stcpb	0.007441
<b>41</b>	is_ftp_login	0.003436
<b>42</b>	is_sm_ips_ports	0.000000

```
In [ ]: # Drop low-score features based on mutual information threshold
mi_threshold = 0.1
low_score_features = mi_scores[mi_scores['Scores'] < mi_threshold]

# Extract the feature names
low_score_feature_names = low_score_features['Features'].tolist()

# Print the list of low score feature names
low_score_feature_names
```

```
Out[ ]: ['load_jit_interaction_src',
 'ct_srv_src',
 'sport',
 'ct_dst_src_ltm',
 'ct_dst_sport_ltm',
 'res_bdy_len',
 'service',
 'duration',
 'swin',
 'proto',
 'trans_depth',
 'ct_flw_http_mthd',
 'dtcpb',
 'stcpb',
 'is_ftp_login',
 'is_sm_ips_ports']
```

```
In [ ]: plt.figure(figsize=(10, 8))

# Create the barplot
sns.barplot(x='Scores', y='Features', data=mi_scores)

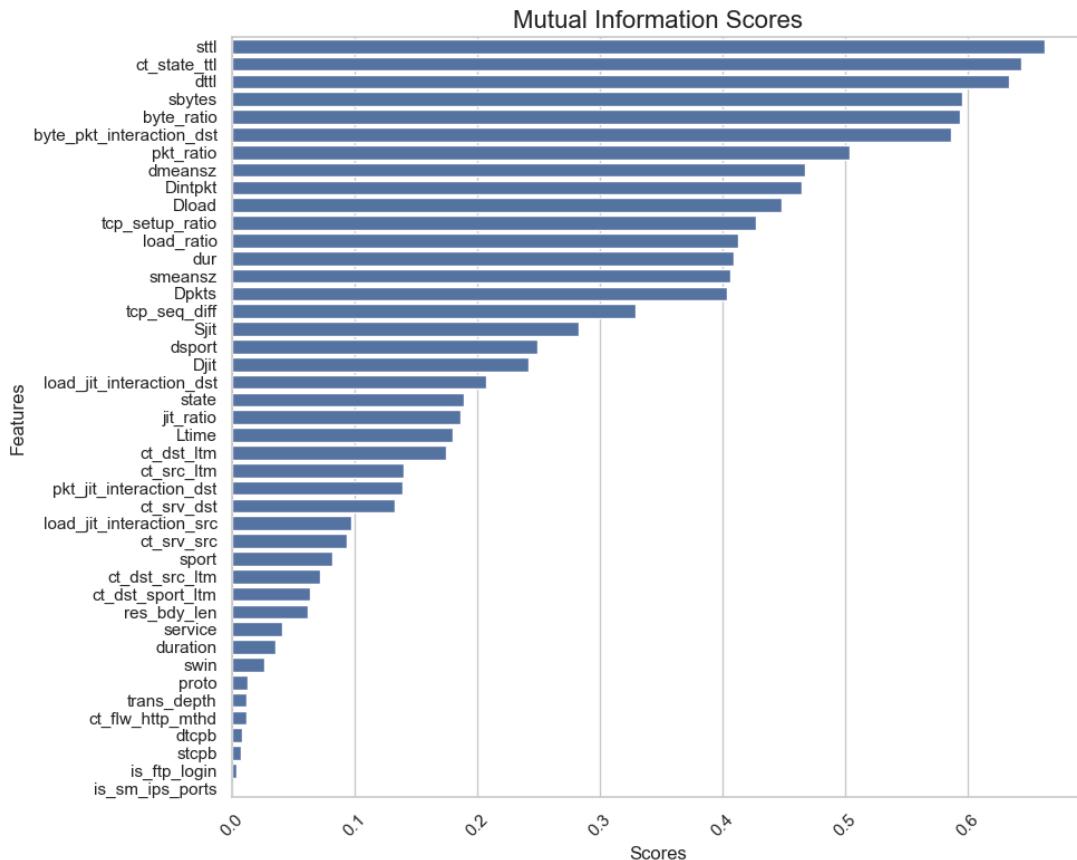
# Add a title
plt.title("Mutual Information Scores", fontsize=16)

# Rotate the y-axis labels (if needed)
plt.yticks(rotation=0)

# Rotate the x-axis labels (if needed)
```

```
plt.xticks(rotation=45)

# Display the plot
plt.tight_layout() # Adjusts the plot to ensure everything fits without overlap
plt.show()
```



```
In [ ]: x.drop(low_score_feature_names, axis=1, inplace = True)
remaining_features = x.columns
print(remaining_features)
```

```
Index(['dsport', 'state', 'dur', 'sbytes', 'sttl', 'dttl', 'Dload', 'Dpkts',
       'smmeansz', 'dmeansz', 'Sjit', 'Djيت', 'Ltime', 'Dintpkt',
       'ct_state_ttl', 'ct_src_dst', 'ct_dst_ltm', 'ct_src_ltm', 'byte_ratio',
       'pkt_ratio', 'load_ratio', 'jit_ratio', 'tcp_setup_ratio',
       'byte_pkt_interaction_dst', 'load_jit_interaction_dst',
       'pkt_jit_interaction_dst', 'tcp_seq_diff'],
      dtype='object', name='Name')
```

## Method 2: p-values and VIF

```
In [ ]: import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler

# Adding a constant term for the intercept in the regression model
x = sm.add_constant(x)

# Function to calculate p-values
```

```

def calculate_p_values(x, y):
    model = sm.OLS(y, x).fit()
    return model.pvalues

# Function to calculate VIF
def calculate_vif(x):
    vif_data = pd.DataFrame()
    vif_data["feature"] = x.columns
    vif_data["VIF"] = [variance_inflation_factor(x.values, i) for i in range(x.shape[1])]
    return vif_data

# Initial calculation of p-values and VIF
p_values = calculate_p_values(x, y)
vif_data = calculate_vif(x)

# Combine p-values and VIF into a single DataFrame with rounding
feature_stats = pd.DataFrame({
    "feature": p_values.index,
    "p_value": np.round(p_values.values, 2),
    "VIF": np.round(vif_data["VIF"], 2)
})

# Sort by p_value first, then by VIF in descending order
feature_stats = feature_stats[feature_stats["feature"] != "const"]

# Display the sorted feature statistics
print(feature_stats)

```

	feature	p_value	VIF
1	dsport	0.00	1.48
2	state	0.00	1.91
3	dur	0.00	1.97
4	sbytes	0.00	1.39
5	sttl	0.00	4.27
6	dttl	0.00	4.96
7	Dload	0.00	2.02
8	Dpkts	0.00	5.28
9	smeansz	0.00	1.33
10	dmeansz	0.00	3.14
11	Sjit	0.00	1.53
12	Djit	0.07	1.75
13	Ltime	0.00	1.19
14	Dintpkt	0.00	1.34
15	ct_state_ttl	0.00	4.39
16	ct_srv_dst	0.00	3.08
17	ct_dst_ltm	0.46	4.48
18	ct_src_ltm	0.00	3.48
19	byte_ratio	0.00	3.16
20	pkt_ratio	0.01	2.83
21	load_ratio	0.00	1.55
22	jit_ratio	0.00	1.07
23	tcp_setup_ratio	0.00	4.37
24	byte_pkt_interaction_dst	0.00	2.89
25	load_jit_interaction_dst	0.00	1.42
26	pkt_jit_interaction_dst	0.51	2.07
27	tcp_seq_diff	0.26	1.00

In [ ]: x.shape

Out[ ]: (176000, 28)

```
In [ ]: # Define thresholds
p_value_threshold = 0.05
vif_threshold = 10

# Iterative process
while True:
    # Step 1: Identify features with high p-value (regardless of VIF)
    high_p_features = feature_stats[feature_stats['p_value'] > p_value_threshold]

    # Step 2: Drop these features from the dataset
    if not high_p_features.empty:
        x.drop(columns=high_p_features['feature'], axis=1, inplace=True)
    else:
        # No more features with high p-value, now check VIF
        high_vif_features = feature_stats[feature_stats['VIF'] > vif_threshold]

        if not high_vif_features.empty:
            x.drop(columns=high_vif_features['feature'], axis=1, inplace=True)
        else:
            # If there are no features to drop, break the loop
            break

    # Recalculate p-values and VIF for the reduced feature set
    p_values = calculate_p_values(x, y)
    vif_data = calculate_vif(x)

    # Update the combined feature statistics
    feature_stats = pd.DataFrame({
        "feature": p_values.index,
        "p_value": np.round(p_values.values, 2),
        "VIF": np.round(vif_data["VIF"], 2)
    })

    # Remove 'const' if added
    feature_stats = feature_stats[feature_stats["feature"] != "const"]

# Final selected features
final_selected_features = feature_stats[
    (feature_stats['p_value'] <= p_value_threshold) &
    (feature_stats['VIF'] <= vif_threshold)
]['feature'].tolist()

print(f"Final selected features: {final_selected_features}")
```

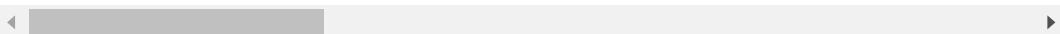
```
Final selected features: ['dsport', 'state', 'dur', 'sbytes', 'sttl', 'dttl', 'Dload', 'Dpkts', 'smeansz', 'dmeansz', 'Sjit', 'Ltime', 'Dintpkt', 'ct_state_ttl', 'ct_srv_dst', 'ct_src_ltm', 'byte_ratio', 'pkt_ratio', 'load_ratio', 'jit_ratio', 'tcp_setup_ratio', 'byte_pkt_interaction_dst', 'load_jit_interaction_dst']
```

```
In [ ]: x = x.drop(columns=['const'])
```

```
In [ ]: x
```

Out[ ]:	dsport	state	dur	sbytes	sttl	dttl	Dload	Dpkts	smean
<b>1068944</b>	53	2	0.001079	146	31	29	6.598702e+05	2	1
<b>715655</b>	13037	5	0.037937	2766	31	29	5.960197e+06	46	6
<b>1130880</b>	14918	5	0.081643	2334	31	29	1.561334e+06	38	6
<b>1436155</b>	63552	5	0.025030	2438	31	29	6.456572e+06	40	6
<b>1141050</b>	24288	5	0.002896	320	31	29	4.516574e+06	8	1
...	...	...	...	...	...	...	...	...	...
<b>985025</b>	80	5	0.670600	724	254	252	2.672234e+03	6	1
<b>1638031</b>	179	5	0.360085	534	254	252	4.976603e+03	6	1
<b>68749</b>	53	6	0.000008	114	254	0	0.000000e+00	0	1
<b>1744713</b>	80	5	0.382740	12550	254	252	4.682030e+03	6	6
<b>605375</b>	10607	5	0.497307	549	254	252	7.013776e+03	10	1

176000 rows × 23 columns



## Method 3: Recursive Feature Elimination (RFE)

```
In [ ]: from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# Initialize the model (Logistic Regression is used as a baseline)
model = LogisticRegression(max_iter=1000)

# Initialize RFE with the model and select the top 'n' features
rfe = RFE(estimator=model, n_features_to_select=20) # Adjust 'n_features_to_select'
rfe.fit(x, y)

# Get the ranking of the features
rfe_ranking = rfe.ranking_

# Filter out features that were not selected by RFE (ranking > 1)
x = x.loc[:, rfe.support_]

# Print the selected features
print("Selected features after RFE:", x.columns.tolist())
```

Selected features after RFE: ['dsport', 'state', 'dur', 'sbytes', 'sttl', 'dttl', 'Dload', 'Dpkts', 'smeansz', 'dmeansz', 'Sjit', 'Dintpkt', 'ct\_stat\_e\_ttl', 'ct\_srv\_dst', 'ct\_src\_ltm', 'byte\_ratio', 'pkt\_ratio', 'load\_ratio', 'jit\_ratio', 'tcp\_setup\_ratio']

```
In [ ]: x.shape
```

```
Out[ ]: (176000, 20)
```

```
In [ ]: # Giving the final features of classifier x train, to the one class x train
x_train_ocsvm = x_train_ocsvm[x.columns]
```

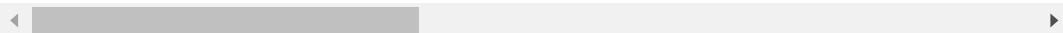
In [ ]: `x_train_ocsvm.shape`

Out[ ]: (391870, 20)

In [ ]: `x_train_ocsvm`

Out[ ]:	Name	dsport	state	dur	sbytes	sttl	dttl	Dload	Dpkts	smean:
	<b>412401</b>	53	2	0.001032	146	31	29	6.899225e+05	2	7
	<b>1489647</b>	25290	5	0.013036	2646	31	29	1.527585e+07	44	6
	<b>371934</b>	5190	5	0.004248	1064	31	29	3.902071e+06	12	8
	<b>1908726</b>	5190	5	0.748613	1920	31	29	4.416701e+04	24	8
	<b>729971</b>	53	2	0.001065	132	31	29	6.159624e+05	2	6
	...	...	...	...	...	...	...	...	...	...
	<b>1390698</b>	11972	5	0.024998	2646	31	29	6.872230e+06	40	6
	<b>2039135</b>	111	2	0.004606	568	31	29	4.168476e+05	4	14
	<b>80809</b>	53	2	0.001024	130	31	29	6.328125e+05	2	6
	<b>619382</b>	53	2	0.001044	130	31	29	6.206896e+05	2	6
	<b>1539459</b>	53	2	0.001059	130	31	29	6.118980e+05	2	6

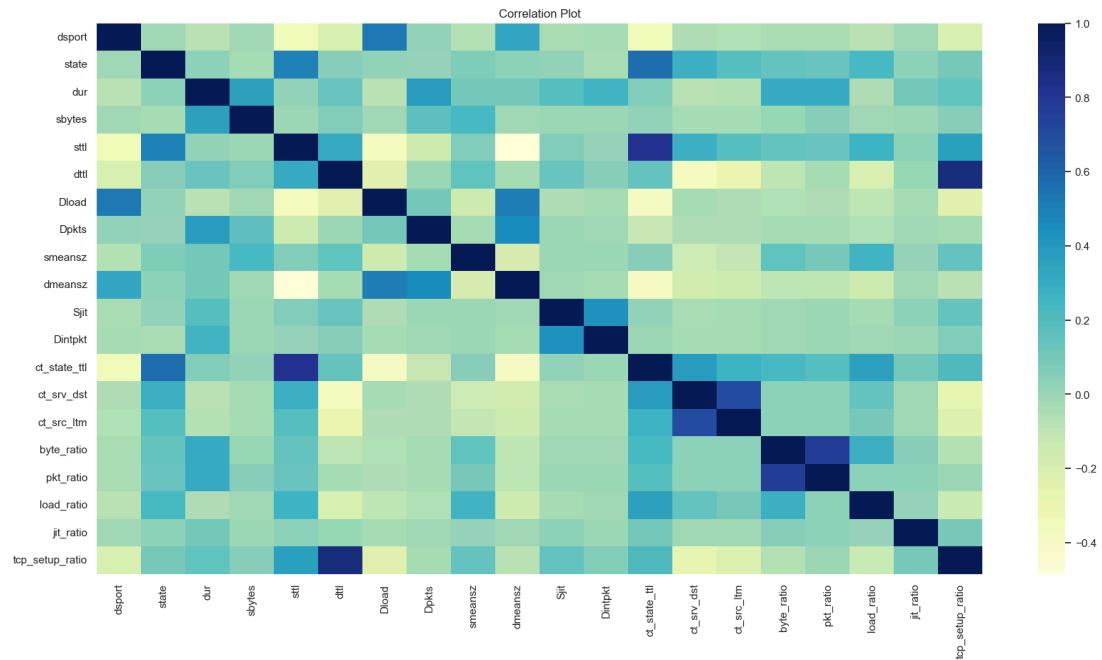
391870 rows × 20 columns



## New Correlation Plot showing only high-scored features

In [ ]: `plt.figure(figsize=(20,10))  
plt.title("Correlation Plot")  
sns.heatmap(x.corr(), cmap='YlGnBu')`

Out[ ]: <Axes: title={'center': 'Correlation Plot'}>



## Saving the resampled data to CSV files before scaling, to be reused in the training scripts

```
In [ ]: x.to_csv('x_resampled.csv', index=False)
y.to_csv('y_resampled.csv', index=False)
```

## Feature Scaling

```
In [ ]: # Standardize the features
# scaler = StandardScaler()
# x_scaled = scaler.fit_transform(x)
```

## Checking Variance captured by features

```
In [ ]: # pca = PCA()
# pca.fit(x)

# explained_variance_ratio = pca.explained_variance_ratio_
# cumulative_variance_ratio = explained_variance_ratio.cumsum()

# plt.plot(cumulative_variance_ratio)
# plt.xlabel('Number of Components')
# plt.ylabel('Cumulative Explained Variance')
# plt.show()
```

```
In [ ]: # rows, columns = x.shape
# print(f"Number of rows: {rows}")
# print(f"Number of columns: {columns}")
# print("1 and 0 distribution:", y.value_counts())
```



**Inference:**

- No need to apply pca as all thw features are capturing full variance

## Dividing in train-test split

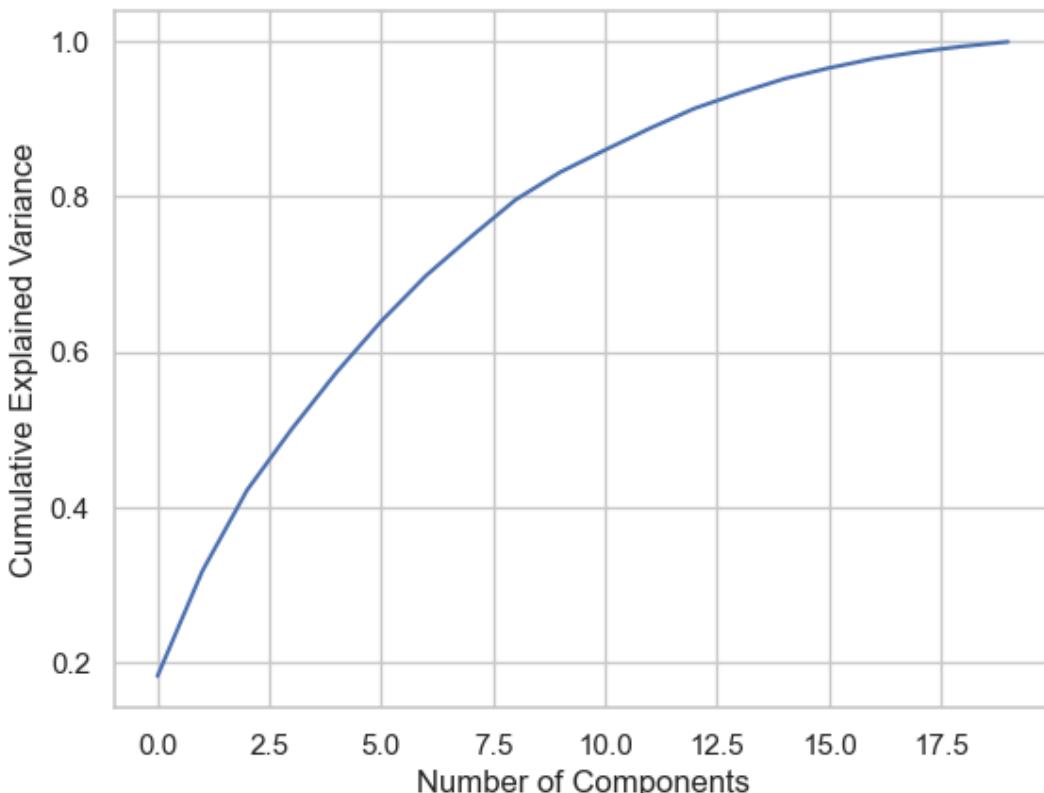
```
In [ ]: x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_val_scaled = scaler.transform(x_val)
```

```
In [ ]: pca = PCA()
pca.fit(x_train_scaled)

explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = explained_variance_ratio.cumsum()

plt.plot(cumulative_variance_ratio)
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```



```
In [ ]: rows, columns = x_train_scaled.shape
print(f"Number of rows: {rows}")
print(f"Number of columns: {columns}")
print("1 and 0 distribution:", y_val.value_counts())
```

```
Number of rows: 140800
Number of columns: 20
1 and 0 distribution: Label
1          17615
0          17585
Name: count, dtype: int64
```

## Code Metrics

Imports the libraries that will be used to evaluate the models later on

```
In [ ]:
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
from sklearn.metrics import confusion_matrix # will plot the confusion matrix
import time

model_performance = pd.DataFrame(columns=['Accuracy', 'Recall', 'Precision'])

def train_evaluate_model(model, x_train_scaled, y_train, x_val_scaled, y_val):
    start = time.time()
    model.fit(x_train_scaled, y_train)
    end_train = time.time()

    preds = model.predict(x_val_scaled)
    end_predict = time.time()

    accuracy = accuracy_score(y_val, preds)
    recall = recall_score(y_val, preds)
    precision = precision_score(y_val, preds)
    f1s = f1_score(y_val, preds)
    mae = mean_absolute_error(y_val, preds)

    print(f"Accuracy: {accuracy:.2%}")
    print(f"Recall: {recall:.2%}")
    print(f"Precision: {precision:.2%}")
    print(f"F1-Score: {f1s:.2%}")
    print(f"Mean Absolute Error: {mae:.2%}")
    print(f"time to train: {end_train-start:.2f} s")
    print(f"time to predict: {end_predict-end_train:.2f} s")
    print(f"total: {end_predict-start:.2f} s")

    model_performance.loc[type(model).__name__] = [accuracy, recall, precision, f1s, mae]

print(f"Classification Report:\n{classification_report(y_val, preds)}")
cm = confusion_matrix(y_val, preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

## 5 | Model Building

## Decision Tree Model

Before we make our Random Forest, we'll see if we can significantly improve our performance with merely a decision tree.

```
In [ ]:
import re
import graphviz
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import GridSearchCV
import joblib

# Train and evaluate Decision Tree
dt = DecisionTreeClassifier()
train_evaluate_model(dt, x_train_scaled, y_train, x_val_scaled, y_val)

# Check if the directory exists, if not, create it
model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

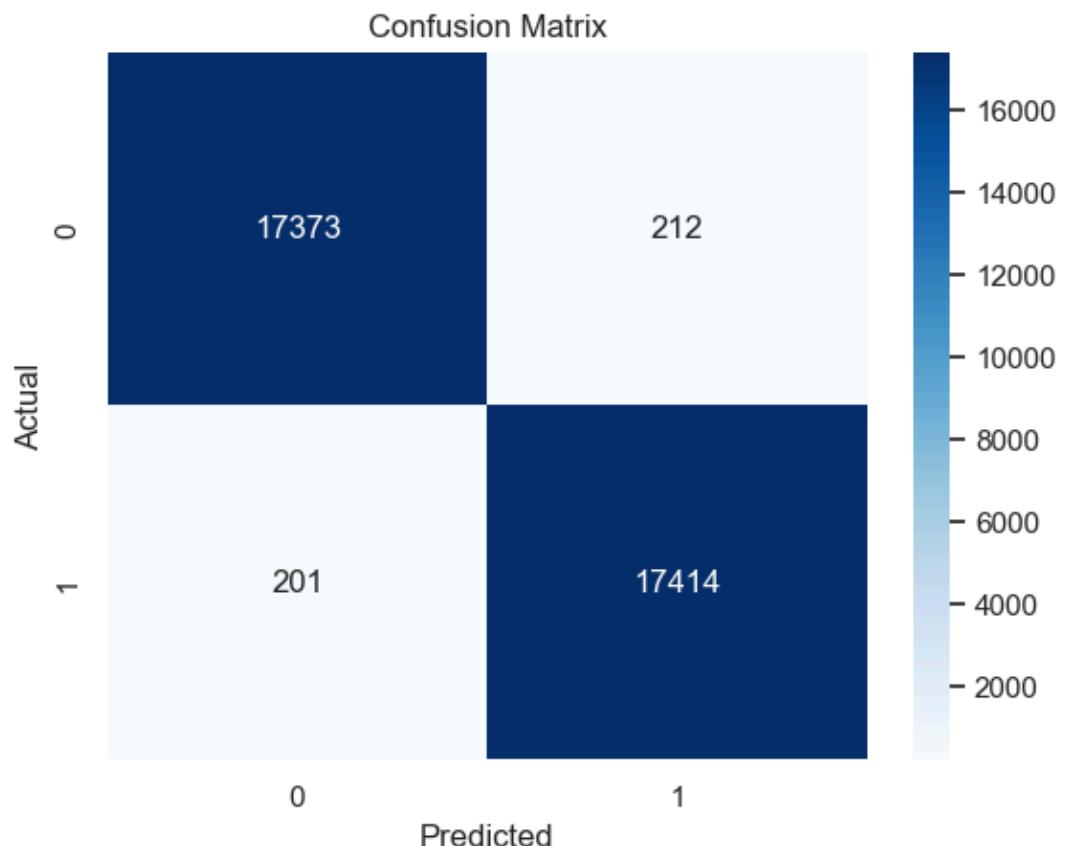
joblib.dump(dt, os.path.join(model_dir, 'decision_tree_model.joblib'))

# Function to draw the decision tree
def draw_tree(t, df, size=10, ratio=0.6, precision=2, **kwargs):
    feature_names = df.columns if hasattr(df, 'columns') else [f'feature_{i}' for i in range(df.shape[1])]
    s = export_graphviz(t, out_file=None, feature_names=feature_names, filled=True,
                        special_characters=True, precision=precision, **kwargs)
    return graphviz.Source(re.sub('Tree {', f'Tree {{ size={size}; ratio={ratio}; ', s))

# Draw the decision tree
draw_tree(dt, x_train_scaled, size=10)
```

Accuracy: 98.83%  
Recall: 98.86%  
Precision: 98.80%  
F1-Score: 98.83%  
Mean Absolute Error: 1.17%  
time to train: 1.09 s  
time to predict: 0.00 s  
total: 1.09 s  
Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	17585
1	0.99	0.99	0.99	17615
accuracy			0.99	35200
macro avg	0.99	0.99	0.99	35200
weighted avg	0.99	0.99	0.99	35200



```

-----
-
FileNotFoundException                                Traceback (most recent call last)
t)
File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\backend\execu
te.py:76, in run_check(cmd, input_lines, encoding, quiet, **kwargs)
    75         kwargs['stdout'] = kwargs['stderr'] = subprocess.PIPE
---> 76     proc = _run_input_lines(cmd, input_lines, kwargs=kwargs)
    77 else:

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\backend\execu
te.py:96, in _run_input_lines(cmd, input_lines, kwargs)
    95 def _run_input_lines(cmd, input_lines, *, kwargs):
---> 96     popen = subprocess.Popen(cmd, stdin=subprocess.PIPE, **kwargs)
    98     stdin_write = popen.stdin.write

File c:\DevTools\Python\Python312\Lib\subprocess.py:1026, in Popen.__init__
(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, clos
e_fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, re
store_signals, start_new_session, pass_fds, user, group, extra_groups, enc
oding, errors, text, umask, pipesize, process_group)
    1023             self.stderr = io.TextIOWrapper(self.stderr,
    1024                                         encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
    1027                                         pass_fds, cwd, env,
    1028                                         startupinfo, creationflags, shell,
    1029                                         p2cread, p2cwrite,
    1030                                         c2pread, c2pwrite,
    1031                                         errread, errwrite,
    1032                                         restore_signals,
    1033                                         gid, gids, uid, umask,
    1034                                         start_new_session, process_group)
1035 except:
    1036     # Cleanup if the child failed starting.

File c:\DevTools\Python\Python312\Lib\subprocess.py:1538, in Popen._execut
e_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,
startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, e
rrread, errwrite, unused_restore_signals, unused_gid, unused_gids, unused_
uid, unused_umask, unused_start_new_session, unused_process_group)
    1537 try:
-> 1538     hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
    1539                                         # no special security
    1540                                         None, None,
    1541                                         int(not close_fds),
    1542                                         creationflags,
    1543                                         env,
    1544                                         cwd,
    1545                                         startupinfo)
1546 finally:
    1547     # Child is launched. Close the parent's copy of those pipe
    1548     # handles that only the child should have open. You need
    (...)

1551     # pipe will not close when the child process exits and the
1552     # ReadFile will hang.

FileNotFoundException: [WinError 2] The system cannot find the file specified
The above exception was the direct cause of the following exception:
```

```

ExecutableNotFound                               Traceback (most recent call last)
t)
File c:\DevTools\Python\Python312\Lib\site-packages\IPython\core\formatter
s.py:977, in MimeBundleFormatter.__call__(self, obj, include, exclude)
    974     method = get_real_method(obj, self.print_method)
    976     if method is not None:
--> 977         return method(include=include, exclude=exclude)
    978     return None
    979 else:

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\jupyter_integ
ration.py:98, in JupyterIntegration._repr_mimebundle_(self, include, exclu
de, **_)
    96     include = set(include) if include is not None else {self._jupyter_
mimetype}
    97     include -= set(exclude or [])
--> 98     return {mimetype: getattr(self, method_name)()
    99         for mimetype, method_name in MIME_TYPES.items()
   100         if mimetype in include}

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\jupyter_integ
ration.py:112, in JupyterIntegration._repr_image_svg_xml(self)
   110 def _repr_image_svg_xml(self) -> str:
   111     """Return the rendered graph as SVG string."""
--> 112     return self.pipe(format='svg', encoding=SVG_ENCODING)

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\piping.py:10
4, in Pipe.pipe(self, format, renderer, formatter, neato_no_op, quiet, eng
ine, encoding)
    55     def pipe(self,
    56             format: typing.Optional[str] = None,
    57             renderer: typing.Optional[str] = None,
    (...),
    61             engine: typing.Optional[str] = None,
    62             encoding: typing.Optional[str] = None) -> typing.Union[by
tes, str]:
    63     """Return the source piped through the Graphviz layout comman
d.
    64
    65     Args:
    (...),
    102         '<?xml version='
    103         """
--> 104     return self._pipe_legacy(format,
    105                                         renderer=renderer,
    106                                         formatter=formatter,
    107                                         neato_no_op=neato_no_op,
    108                                         quiet=quiet,
    109                                         engine=engine,
    110                                         encoding=encoding)

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\_tools.py:17
1, in deprecate_positional_args.<locals>.decorator.<locals>.wrapper(*args,
**kwargs)
   162     wanted = ', '.join(f'{name}={value!r}'
   163                           for name, value in deprecated.items())
   164     warnings.warn(f'The signature of {func.__name__} will be reduc
ed'
   165                           f' to {supported_number} positional args'
   166                           f' {list(supported)}: pass {wanted}'
```

```

167             ' as keyword arg(s)',
168             stacklevel=stacklevel,
169             category=category)
--> 171 return func(*args, **kwargs)

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\piping.py:12
1, in Pipe._pipe_legacy(self, format, renderer, formatter, neato_no_op, quiet, engine, encoding)
    112 @_tools.deprecate_positional_args(supported_number=2)
    113 def _pipe_legacy(self,
    114                     format: typing.Optional[str] = None,
    (...),
    119                     engine: typing.Optional[str] = None,
    120                     encoding: typing.Optional[str] = None) -> typing.
Union[bytes, str]:
--> 121     return self._pipe_future(format,
    122                                         renderer=renderer,
    123                                         formatter=formatter,
    124                                         neato_no_op=neato_no_op,
    125                                         quiet=quiet,
    126                                         engine=engine,
    127                                         encoding=encoding)

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\piping.py:14
9, in Pipe._pipe_future(self, format, renderer, formatter, neato_no_op, quiet, engine, encoding)
    146 if encoding is not None:
    147     if codecs.lookup(encoding) is codecs.lookup(self.encoding):
    148         # common case: both stdin and stdout need the same encoding
--> 149     return self._pipe_lines_string(*args, encoding=encoding, **kwargs)
    150     try:
    151         raw = self._pipe_lines(*args, input_encoding=self.encoding,
    152                               **kwargs)

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\backend\piping.py:212, in pipe_lines_string(engine, format, input_lines, encoding, renderer, formatter, neato_no_op, quiet)
    206 cmd = dot_command.command(engine, format,
    207                             renderer=renderer,
    208                             formatter=formatter,
    209                             neato_no_op=neato_no_op)
    210 kwargs = {'input_lines': input_lines, 'encoding': encoding}
--> 212 proc = execute.run_check(cmd, capture_output=True, quiet=quiet, **kwargs)
    213 return proc.stdout

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\backend\execute.py:81, in run_check(cmd, input_lines, encoding, quiet, **kwargs)
    79 except OSError as e:
    80     if e.errno == errno.ENOENT:
--> 81         raise ExecutableNotFound(cmd) from e
    82     raise
    84 if not quiet and proc.stderr:

ExecutableNotFound: failed to execute WindowsPath('dot'), make sure the Graphviz executables are on your systems' PATH

```

Out[ ]: <graphviz.sources.Source at 0x28bb069bef0>

## Random Forest Model

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV

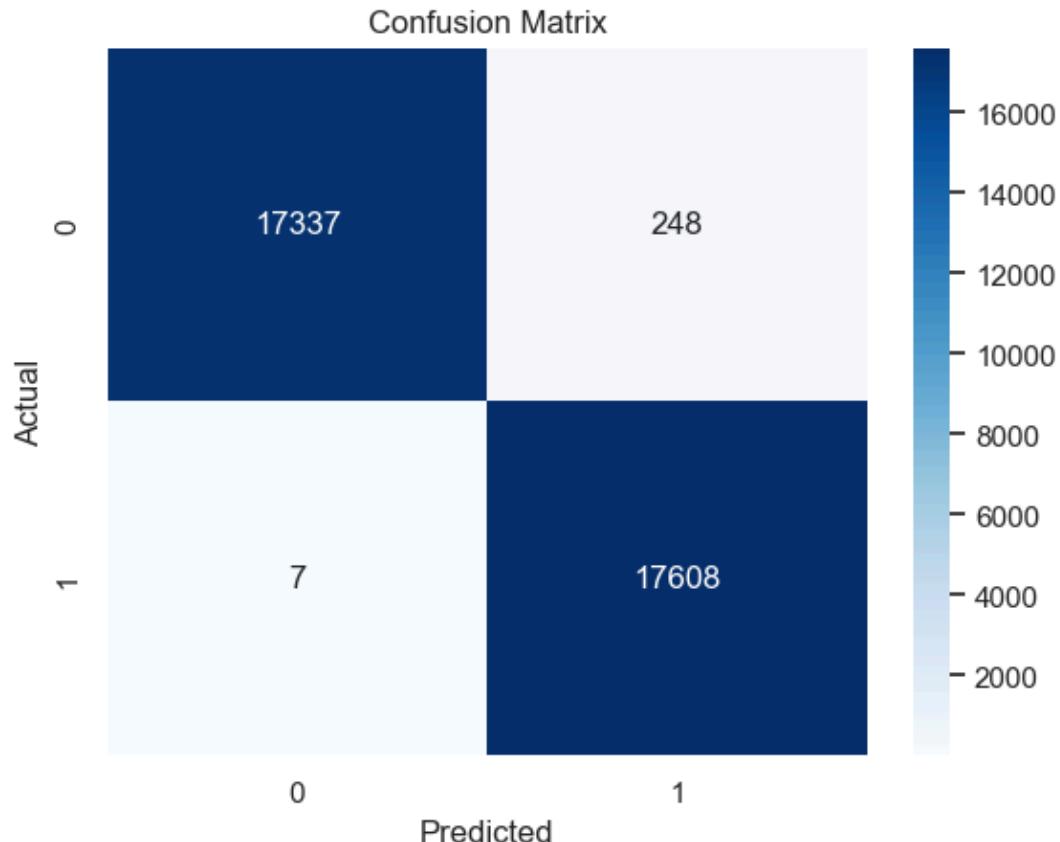
# Train and evaluate Random Forest
rf = RandomForestClassifier(n_estimators=100, n_jobs=-1, random_state=0,
train_evaluate_model(rf, x_train_scaled, y_train, x_val_scaled, y_val)

# Check if the directory exists, if not, create it
model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

joblib.dump(rf, os.path.join(model_dir, 'random_forest_model.joblib'))
```

Accuracy: 99.28%  
Recall: 99.96%  
Precision: 98.61%  
F1-Score: 99.28%  
Mean Absolute Error: 0.72%  
time to train: 5.48 s  
time to predict: 0.07 s  
total: 5.55 s  
Classification Report:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	17585
1	0.99	1.00	0.99	17615
accuracy			0.99	35200
macro avg	0.99	0.99	0.99	35200
weighted avg	0.99	0.99	0.99	35200



Out[ ]: ['model\\random\_forest\_model.joblib']

## Gradient Boosting Model

```
In [ ]: from sklearn.ensemble import GradientBoostingClassifier

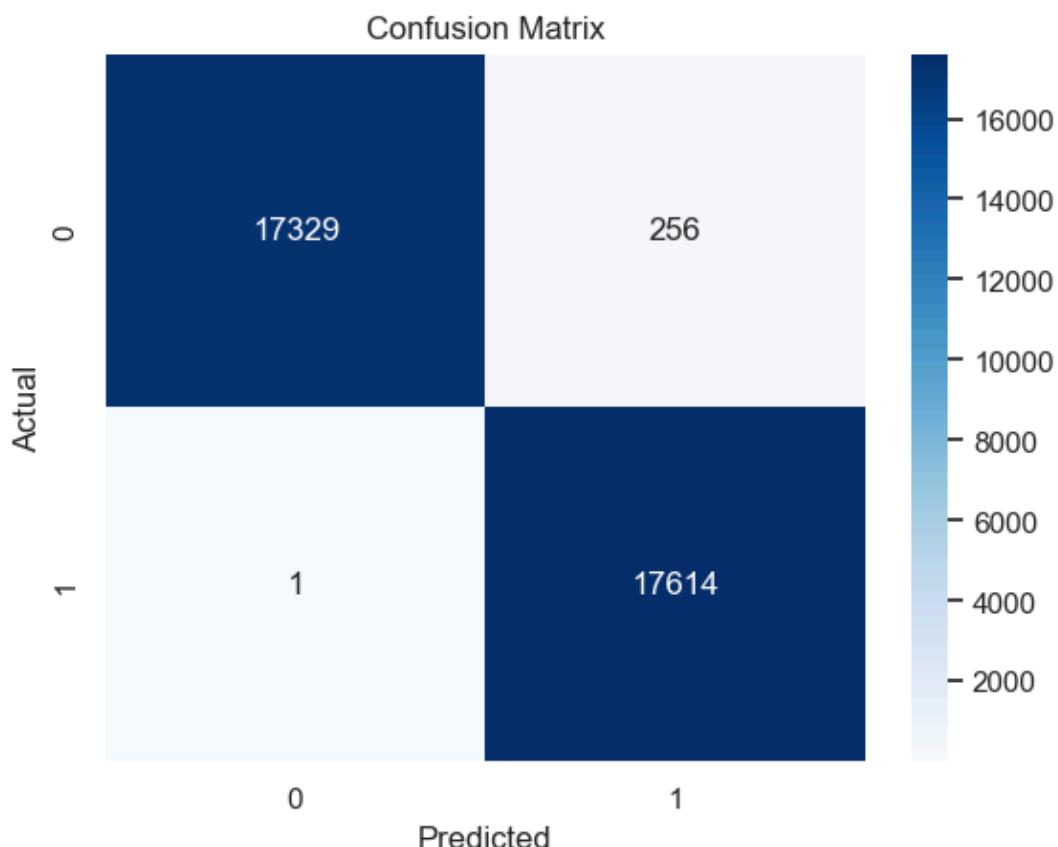
# Train and evaluate Gradient Boosting Classifier
gbm = GradientBoostingClassifier(random_state=42)
train_evaluate_model(gbm, x_train_scaled, y_train, x_val_scaled, y_val)

# Check if the directory exists, if not, create it
model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

joblib.dump(gbm, os.path.join(model_dir, 'gradient_boosting_model.joblib'))
```

Accuracy: 99.27%  
 Recall: 99.99%  
 Precision: 98.57%  
 F1-Score: 99.28%  
 Mean Absolute Error: 0.73%  
 time to train: 44.10 s  
 time to predict: 0.04 s  
 total: 44.14 s  
 Classification Report:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	17585
1	0.99	1.00	0.99	17615
accuracy			0.99	35200
macro avg	0.99	0.99	0.99	35200
weighted avg	0.99	0.99	0.99	35200



Out[ ]: ['model\\gradient\_boosting\_model.joblib']

## Logistic Regression Model

```
In [ ]: from sklearn.linear_model import LogisticRegression

# Train and evaluate Logistic Regression
lr = LogisticRegression()
train_evaluate_model(lr, x_train_scaled, y_train, x_val_scaled, y_val)

# Check if the directory exists, if not, create it
model_dir = 'model'
```

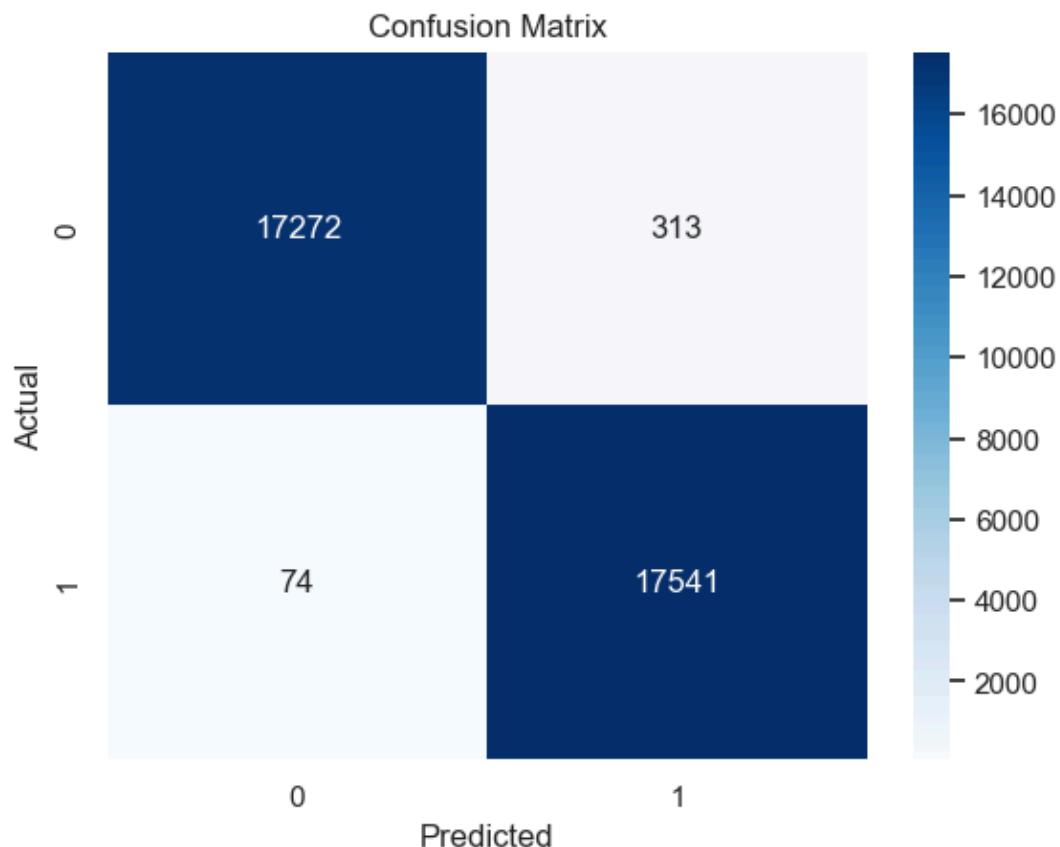
```

if not os.path.exists(model_dir):
    os.makedirs(model_dir)

joblib.dump(lr, os.path.join(model_dir, 'logistic_regression_model.joblib')

Accuracy: 98.90%
Recall: 99.58%
Precision: 98.25%
F1-Score: 98.91%
Mean Absolute Error: 1.10%
time to train: 0.29 s
time to predict: 0.00 s
total: 0.29 s
Classification Report:
      precision    recall   f1-score   support
          0         1.00     0.98     0.99    17585
          1         0.98     1.00     0.99    17615
          accuracy           0.99
          macro avg       0.99     0.99     0.99    35200
          weighted avg    0.99     0.99     0.99    35200

```



Out[ ]: ['model\\logistic\_regression\_model.joblib']

## KNeighbors Classifier model

```

In [ ]: from sklearn.neighbors import KNeighborsClassifier

# Train and evaluate KNeighbors Classifier

```

```

kn = KNeighborsClassifier(n_neighbors=3)
train_evaluate_model(kn, x_train_scaled, y_train, x_val_scaled, y_val)

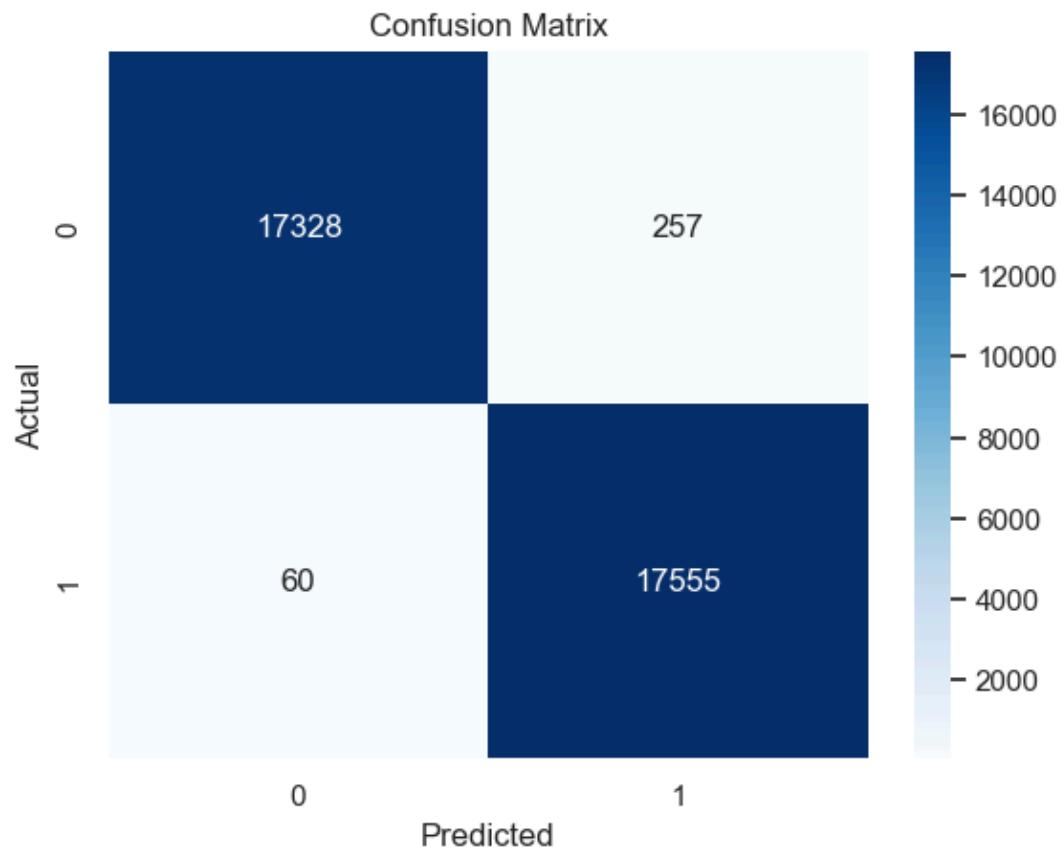
# Check if the directory exists, if not, create it
model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

joblib.dump(kn, os.path.join(model_dir, 'kneighbors_model.joblib'))

```

Accuracy: 99.10%  
 Recall: 99.66%  
 Precision: 98.56%  
 F1-Score: 99.11%  
 Mean Absolute Error: 0.90%  
 time to train: 0.03 s  
 time to predict: 7.60 s  
 total: 7.63 s  
 Classification Report:  

	precision	recall	f1-score	support
0	1.00	0.99	0.99	17585
1	0.99	1.00	0.99	17615
accuracy			0.99	35200
macro avg	0.99	0.99	0.99	35200
weighted avg	0.99	0.99	0.99	35200



Out[ ]: ['model\\kneighbors\_model.joblib']

## Extra Trees Model

```
In [ ]: from sklearn.ensemble import ExtraTreesClassifier

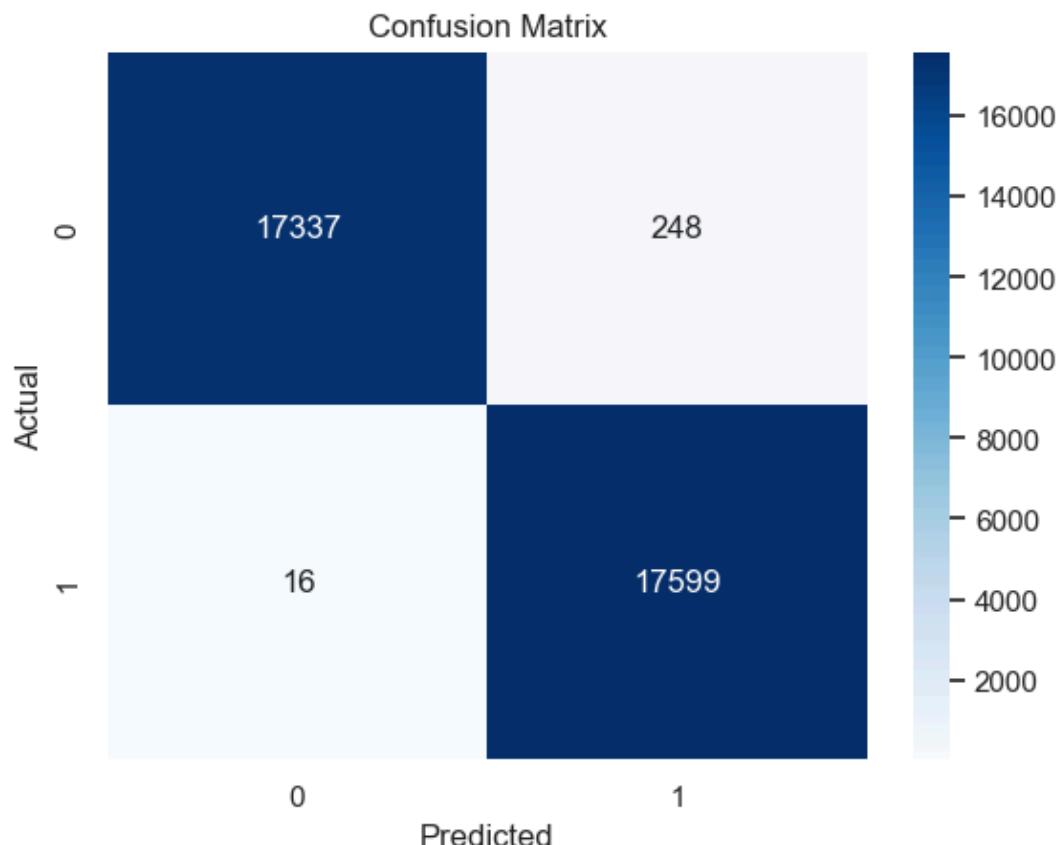
# Train and evaluate Extra Trees
et = ExtraTreesClassifier(random_state=42, n_jobs=-1)
train_evaluate_model(et, x_train_scaled, y_train, x_val_scaled, y_val)

# Check if the directory exists, if not, create it
model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

joblib.dump(et, os.path.join(model_dir, 'extra_trees_model.joblib'))
```

Accuracy: 99.25%  
 Recall: 99.91%  
 Precision: 98.61%  
 F1-Score: 99.26%  
 Mean Absolute Error: 0.75%  
 time to train: 3.45 s  
 time to predict: 0.11 s  
 total: 3.56 s  
 Classification Report:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	17585
1	0.99	1.00	0.99	17615
accuracy			0.99	35200
macro avg	0.99	0.99	0.99	35200
weighted avg	0.99	0.99	0.99	35200



```
Out[ ]: ['model\\extra_trees_model.joblib']
```

## SVM model (SGDClassifier)

```
In [ ]: from sklearn.linear_model import SGDClassifier

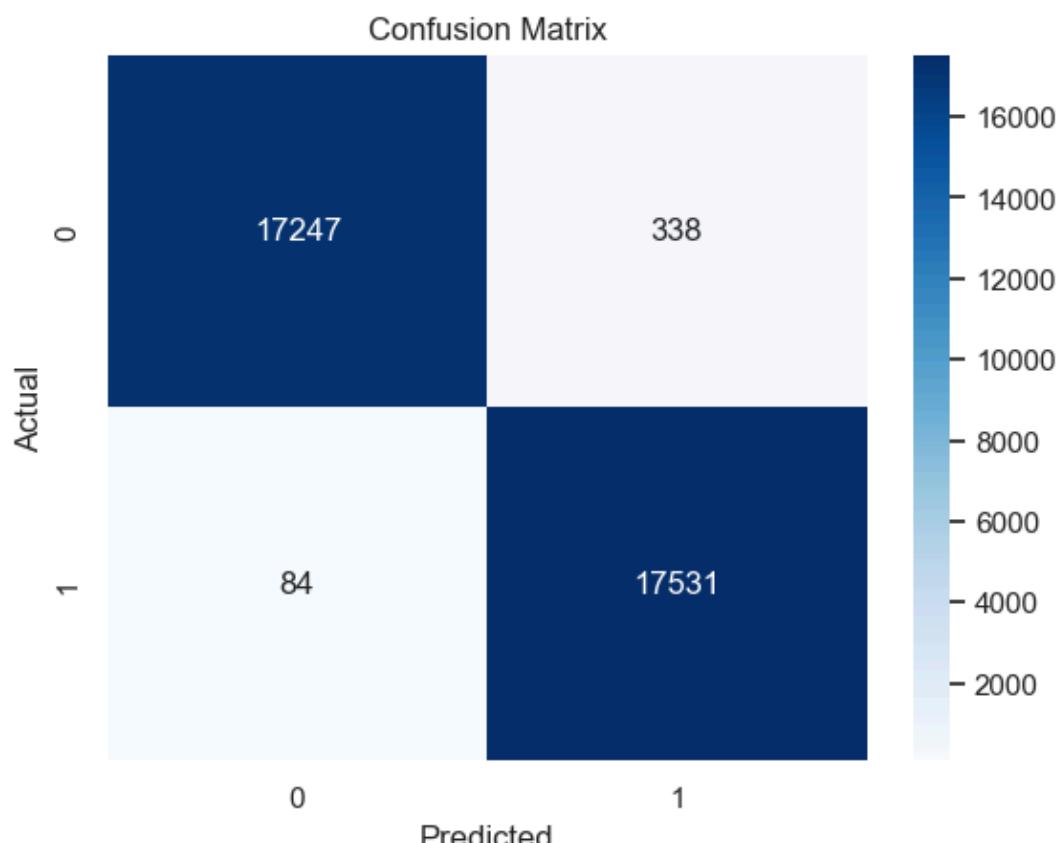
# Train and evaluate SVM with SGD Classifier
sgdc_svm = SGDClassifier(loss='hinge', random_state=42)
train_evaluate_model(sgdc_svm, x_train_scaled, y_train, x_val_scaled, y_val)

# Check if the directory exists, if not, create it
model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

joblib.dump(sgdc_svm, os.path.join(model_dir, 'sgdc_svm_model.joblib'))
```

Accuracy: 98.80%  
 Recall: 99.52%  
 Precision: 98.11%  
 F1-Score: 98.81%  
 Mean Absolute Error: 1.20%  
 time to train: 0.42 s  
 time to predict: 0.00 s  
 total: 0.42 s  
 Classification Report:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	17585
1	0.98	1.00	0.99	17615
accuracy			0.99	35200
macro avg	0.99	0.99	0.99	35200
weighted avg	0.99	0.99	0.99	35200



```
Out[ ]: ['model\\sgdc_svm_model.joblib']
```

## SVM model (SVC)

```
In [ ]: from sklearn.svm import SVC
from sklearn.svm import LinearSVC

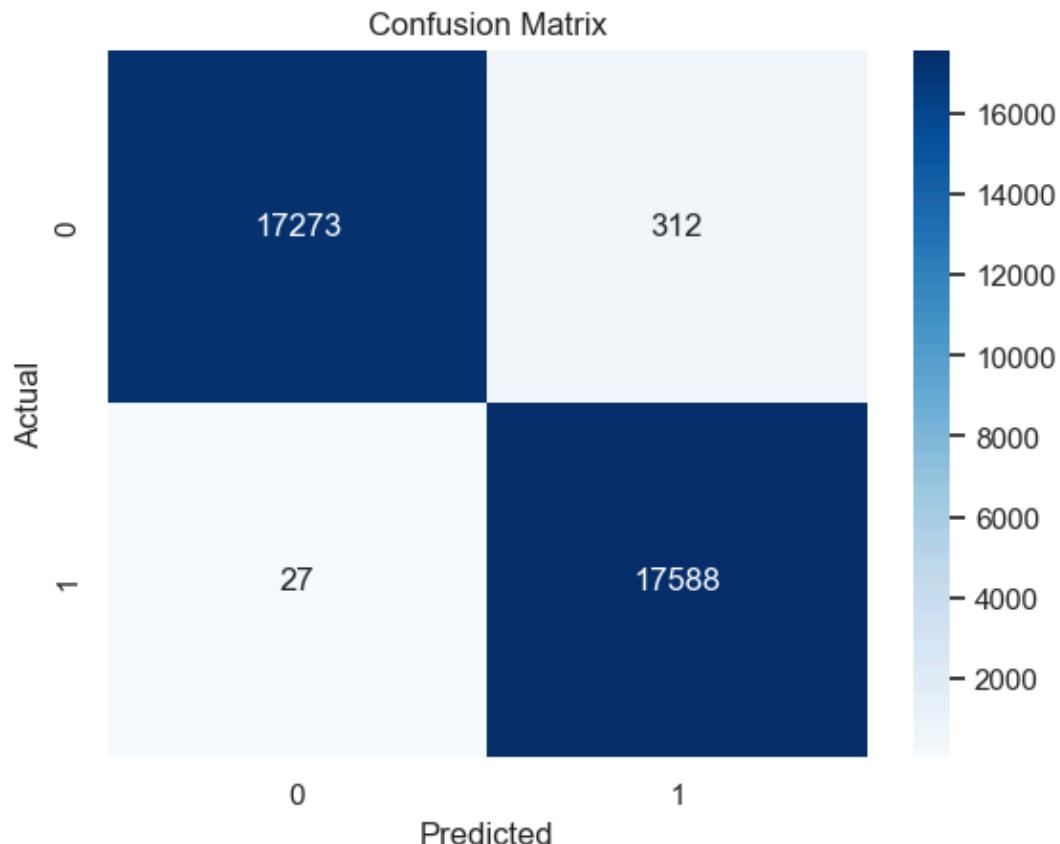
# Train and evaluate SVM with SVC Classifier
svc_svm = LinearSVC()
train_evaluate_model(svc_svm, x_train_scaled, y_train, x_val_scaled, y_val)

# Check if the directory exists, if not, create it
model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

joblib.dump(svc_svm, os.path.join(model_dir, 'svc_svm_model.joblib'))
```

Accuracy: 99.04%  
 Recall: 99.85%  
 Precision: 98.26%  
 F1-Score: 99.05%  
 Mean Absolute Error: 0.96%  
 time to train: 1.67 s  
 time to predict: 0.00 s  
 total: 1.67 s  
 Classification Report:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	17585
1	0.98	1.00	0.99	17615
accuracy			0.99	35200
macro avg	0.99	0.99	0.99	35200
weighted avg	0.99	0.99	0.99	35200



```
Out[ ]: ['model\\svc_svm_model.joblib']
```

## SVM model (OneClassSVM) - Unsupervised

An unsupervised model trained on normal packet data. Aimed to identify only what is considered normal. I will work in conjunction with the ensemble classifier model to identify normal or attack packets.

```
In [ ]: # x_test_ocsvm = x_val
# y_test_ocsvm = y_val

y_val.value_counts()
```

```
Out[ ]: Label
1      17615
0      17585
Name: count, dtype: int64
```

```
In [ ]: x_train_ocsvm.shape
```

```
Out[ ]: (391870, 20)
```

```
In [ ]: y_val.shape
```

```
Out[ ]: (35200, 1)
```

```
In [ ]: x_val.shape
```

```
Out[ ]: (35200, 20)
```

```
In [ ]: # Standardize the features
scaler = StandardScaler()
x_train_ocsvm_scaled = scaler.fit_transform(x_train_ocsvm)
x_test_ocsvm_scaled = scaler.transform(x_val)
```

Function to train and evaluate the One-Class SVM model

```
In [ ]: # Define the train and evaluate function for One-Class SVM
def train_evaluate_ocsvm(model, x_train_ocsvm_scaled, x_test_ocsvm_scaled):
    start = time.time()
    model.fit(x_train_ocsvm_scaled)
    end_train = time.time()

    # Predicting the anomalies
    preds = model.predict(x_test_ocsvm_scaled)
    end_predict = time.time()

    # Adjust predictions: +1 -> 0 (normal), -1 -> 1 (anomaly)
    preds = np.where(preds == 1, 0, 1)

    accuracy = accuracy_score(y_val, preds)
    recall = recall_score(y_val, preds)
    precision = precision_score(y_val, preds)
    f1s = f1_score(y_val, preds)
    mae = mean_absolute_error(y_val, preds)

    print(f"Accuracy: {accuracy:.2%}")
    print(f"Recall: {recall:.2%}")
    print(f"Precision: {precision:.2%}")
    print(f"F1-Score: {f1s:.2%}")
    print(f"Mean Absolute Error: {mae:.2%}")
    print(f"time to train: {end_train-start:.2f} s")
    print(f"time to predict: {end_predict-end_train:.2f} s")
    print(f"total: {end_predict-start:.2f} s")

    model_performance.loc['SVM model (OneClassSVM)'] = [accuracy, recall,]

    # Print classification report and plot confusion matrix
    print(f"Classification Report:\n{classification_report(y_val, preds)}")
    cm = confusion_matrix(y_val, preds)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()
```

The One-Class SVM model is trained on the normal data (label 0) only. The model is then evaluated on the validation set, and the performance metrics are calculated.

```
In [ ]: from sklearn.svm import OneClassSVM
from sklearn.model_selection import GridSearchCV
from skopt import BayesSearchCV
from skopt.space import Real, Categorical
from sklearn.metrics import make_scorer, accuracy_score
```

```

# # Ensure reproducibility
# np.random.seed(42)

# # Randomly sample rows from x_train_ocsvm_scaled to match the number of
# sample_indices = np.random.choice(x_train_ocsvm_scaled.shape[0], size=y
# # x_train_temp = x_train_ocsvm_scaled[sample_indices, :] # Maintain all

# # Define the parameter space for Bayesian search
# param_space = {
#     'nu': Real(0.01, 0.09, prior='uniform'),
#     'gamma': Categorical(['scale', 'auto']),
#     'kernel': Categorical(['linear', 'rbf', 'sigmoid'])
# }

# # Define a custom scoring function
# def custom_scorer(y_true, y_pred):
#     # Adjust predictions: +1 -> 0 (normal), -1 -> 1 (anomaly)
#     y_pred = np.where(y_pred == 1, 0, 1)
#     return accuracy_score(y_true, y_pred)

# # Wrap the custom_scorer to be compatible with BayesSearchCV
# def scoring_function(estimator, X, y):
#     y_pred = estimator.predict(X)
#     return custom_scorer(y, y_pred)

# # Initialize the One-Class SVM
# oc_svm = OneClassSVM()

# # Initialize Bayesian Search with the custom scoring function
# bayes_search = BayesSearchCV(
#     estimator=oc_svm,
#     search_spaces=param_space,
#     n_iter=20, # Number of parameter settings sampled
#     scoring=scoring_function, # Use the wrapped custom scorer
#     cv=[(slice(None), slice(None))], # Dummy cross-validation
#     n_jobs=-1,
#     random_state=42
# )

# # Perform Bayesian search using the temporary training set
# bayes_search.fit(x_train_temp, y_val)

# # Get the best parameters and the corresponding accuracy
# best_params = bayes_search.best_params_
# best_score = bayes_search.best_score_

# print(f"Best Parameters: {best_params}")
# print(f"Best Accuracy: {best_score:.2%}")

```

```

In [ ]: # Initialize the One-Class SVM
oc_svm = OneClassSVM(
    nu=0.01,
    gamma='auto',
    kernel='linear'
)

# Train and evaluate One Class SVM
train_evaluate_ocsvm(oc_svm, x_train_ocsvm_scaled, x_test_ocsvm_scaled, 

# Check if the directory exists, if not, create it

```

```

model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

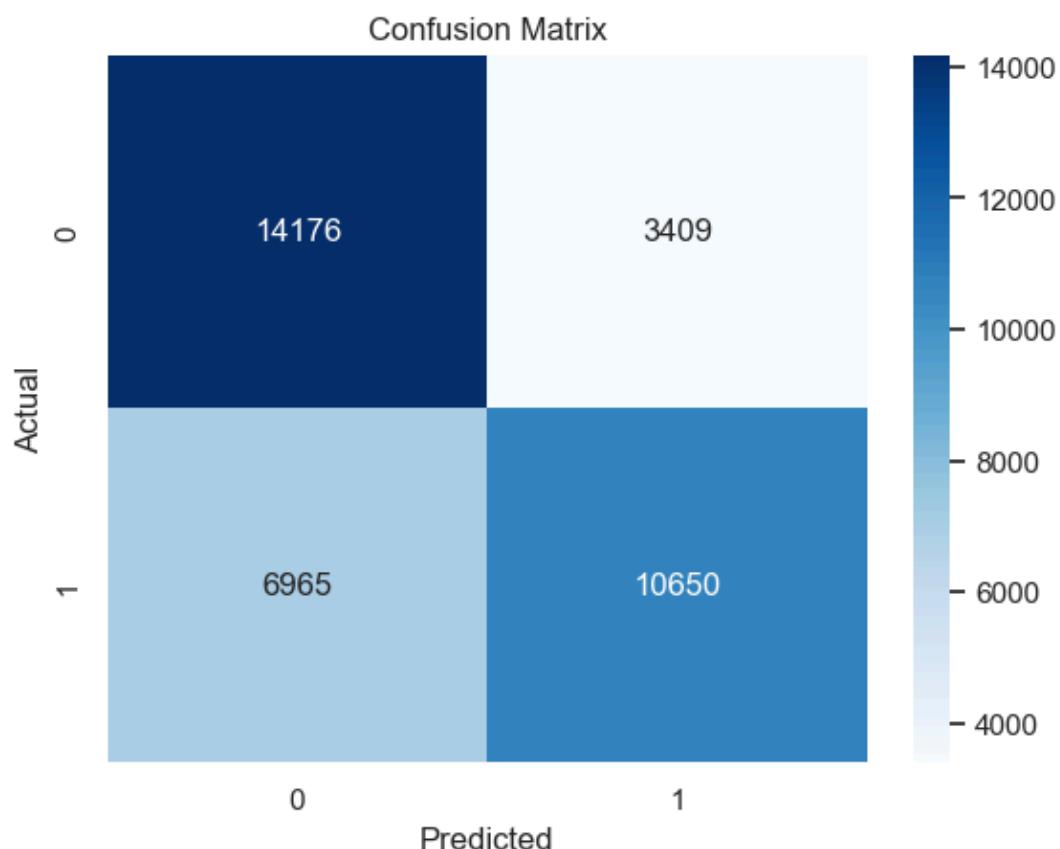
joblib.dump(oc_svm, os.path.join(model_dir, 'one_class_svm_model.joblib'))

```

Accuracy: 70.53%  
 Recall: 60.46%  
 Precision: 75.75%  
 F1-Score: 67.25%  
 Mean Absolute Error: 29.47%  
 time to train: 36.50 s  
 time to predict: 2.22 s  
 total: 38.72 s

Classification Report:

	precision	recall	f1-score	support
0	0.67	0.81	0.73	17585
1	0.76	0.60	0.67	17615
accuracy			0.71	35200
macro avg	0.71	0.71	0.70	35200
weighted avg	0.71	0.71	0.70	35200



Out[ ]: ['model\\one\_class\_svm\_model.joblib']

```

In [ ]: # # Cross-validation
# scores = cross_val_score(best_oc_svm, x_train_ocsvm_scaled, y_train_ocs
# print(f"Cross-validation scores: {scores}")
# print(f"Mean cross-validation score: {scores.mean()}")

```

## Summary and Classifier Ensemble

```
In [ ]: # Define the classifier ensemble
from sklearn.ensemble import VotingClassifier

ensemble_model = VotingClassifier(estimators=[
    ('decision_tree', dt),
    ('random_forest', rf),
    ('gradient_boosting', gbm),
    ('logistic_regression', lr),
    ('kneighbors', kn),
    ('extra_trees', et),
    ('sgdc_svm', sgdc_svm),
    ('svc_svm', svc_svm)
], voting='hard')

# Fit the ensemble model on the training data
start = time.time()
ensemble_model.fit(x_train_scaled, y_train)
end_train = time.time()

# Save the ensemble model
# Create a directory to save the models if it doesn't exist
model_dir = 'model'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)
ensemble_model_path = os.path.join(model_dir, 'classifier_ensemble_model')
joblib.dump(ensemble_model, ensemble_model_path)
print(f"Classifier ensemble model saved at {ensemble_model_path}")

# Make predictions with the ensemble model
ensemble_preds_val = ensemble_model.predict(x_val_scaled)
end_predict = time.time()
```

Classifier ensemble model saved at model\classifier\_ensemble\_model.joblib

## Ensemble Final Statistics

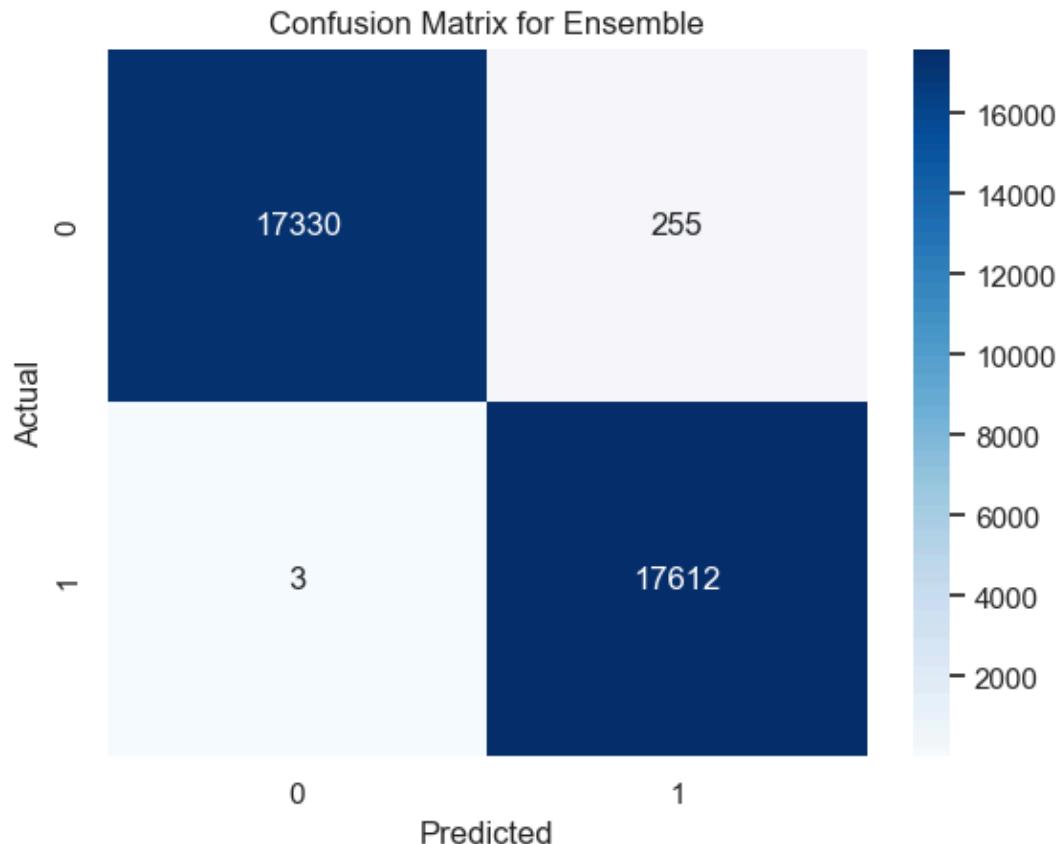
```
In [ ]: # Calculate metrics for the ensemble model
accuracy = accuracy_score(y_val, ensemble_preds_val)
recall = recall_score(y_val, ensemble_preds_val, average='weighted')
precision = precision_score(y_val, ensemble_preds_val, average='weighted')
f1s = f1_score(y_val, ensemble_preds_val, average='weighted')
mae = mean_absolute_error(y_val, ensemble_preds_val)

# Print metrics
print("Accuracy: " + "{:.2%}".format(accuracy))
print("Recall: " + "{:.2%}".format(recall))
print("Precision: " + "{:.2%}".format(precision))
print("F1-Score: " + "{:.2%}".format(f1s))
print("MAE: " + "{:.2%}".format(mae))
print("time to train: " + "{:.2f}" .format(end_train-start)+" s")
print("time to predict: " + "{:.2f}" .format(end_predict-end_train)+" s")
print("total: " + "{:.2f}" .format(end_predict-start)+" s")
model_performance.loc['Ensemble (without OC)'] = [accuracy, recall, preci
```

Accuracy: 99.27%  
 Recall: 99.27%  
 Precision: 99.28%  
 F1-Score: 99.27%  
 MAE: 0.73%  
 time to train: 56.02 s  
 time to predict: 7.43 s  
 total: 63.45 s

```
In [ ]: # Display classification report and confusion matrix
print(f"Classification Report:\n{classification_report(y_val, ensemble_preds_val)}
cm = confusion_matrix(y_val, ensemble_preds_val)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Ensemble')
plt.show()
```

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	0.99	0.99	17585	
1	0.99	1.00	0.99	17615	
accuracy			0.99	35200	
macro avg	0.99	0.99	0.99	35200	
weighted avg	0.99	0.99	0.99	35200	



-- Classifier models are best for well-defined, labeled datasets.

-- Anomaly detection models are best for identifying outliers in predominantly normal datasets.

-- Combining both can leverage the strengths of each, providing a robust solution for detecting both known and unknown attacks in packet traffic.

## 6 | Final Metrics

Finally, all models are compared alongside the ensemble in the following table

In [ ]:

```
model_performance.style.background_gradient(cmap='coolwarm').format({'Accuracy': '{:.2%}', 'Recall': '{:.2%}', 'Precision': '{:.2%}', 'F1-Score': '{:.2%}', 'Mean Absolute Error': '{:.4f}', 'time to train': '{:.1f}'})
```

Out[ ]:

	Accuracy	Recall	Precision	F1-Score	Mean Absolute Error	time to train
<b>DecisionTreeClassifier</b>	98.83%	98.86%	98.80%	98.83%	0.011733	1.1
<b>RandomForestClassifier</b>	99.28%	99.96%	98.61%	99.28%	0.007244	5.5
<b>GradientBoostingClassifier</b>	99.27%	99.99%	98.57%	99.28%	0.007301	44.1
<b>LogisticRegression</b>	98.90%	99.58%	98.25%	98.91%	0.010994	0.3
<b>KNeighborsClassifier</b>	99.10%	99.66%	98.56%	99.11%	0.009006	0.0
<b>ExtraTreesClassifier</b>	99.25%	99.91%	98.61%	99.26%	0.007500	3.5
<b>SGDClassifier</b>	98.80%	99.52%	98.11%	98.81%	0.011989	0.4
<b>LinearSVC</b>	99.04%	99.85%	98.26%	99.05%	0.009631	1.7
<b>SVM model (OneClassSVM)</b>	70.53%	60.46%	75.75%	67.25%	0.294716	36.5
<b>Ensemble (without OC)</b>	99.27%	99.27%	99.28%	99.27%	0.007330	56.0

## Cross-validated accuracy

In [ ]:

```
from sklearn.model_selection import cross_val_score

# Cross-validated scores
dt_scores = cross_val_score(dt, x_resampled, y_resampled.values.ravel(), cv=5)
print(f"Decision Tree cross-validated accuracy: {dt_scores.mean():.2%}")

rf_scores = cross_val_score(rf, x_resampled, y_resampled.values.ravel(), cv=5)
print(f"Random Forest cross-validated accuracy: {rf_scores.mean():.2%}")

gbm_scores = cross_val_score(gbm, x_resampled, y_resampled.values.ravel(), cv=5)
print(f"Gradient Boosting cross-validated accuracy: {gbm_scores.mean():.2%}")
```

```

lr_scores = cross_val_score(lr, x_resampled, y_resampled.values.ravel(),
print(f"Logistic Regression cross-validated accuracy: {lr_scores.mean():.2%}")

kn_scores = cross_val_score(kn, x_resampled, y_resampled.values.ravel(),
print(f"kNeighbours cross-validated accuracy: {kn_scores.mean():.2%}")

et_scores = cross_val_score(et, x_resampled, y_resampled.values.ravel(),
print(f"Extra Trees cross-validated accuracy: {et_scores.mean():.2%}")

sgdc_svm_scores = cross_val_score(sgdc_svm, x_resampled, y_resampled.values.ravel(),
print(f"SGD SVM cross-validated accuracy: {sgdc_svm_scores.mean():.2%}")

svc_svm_scores = cross_val_score(svc_svm, x_resampled, y_resampled.values.ravel(),
print(f"SVC SVM cross-validated accuracy: {svc_svm_scores.mean():.2%}")

```

Decision Tree cross-validated accuracy: 98.74%  
 Random Forest cross-validated accuracy: 99.26%  
 Gradient Boosting cross-validated accuracy: 99.26%  
 Logistic Regression cross-validated accuracy: 83.75%  
 kNeighbours cross-validated accuracy: 95.44%  
 Extra Trees cross-validated accuracy: 99.25%  
 SGD SVM cross-validated accuracy: 61.41%  
 SVC SVM cross-validated accuracy: 83.62%

## Keras Models

### Neural Network MLP (Keras)

```

In [ ]: #Import Libraries that will allow you to use keras
import tensorflow.keras.backend as K
import keras
from keras import layers
from keras import metrics
import numpy as np
from numpy import array
from sklearn.model_selection import train_test_split
import time

```

```

In [ ]: # Custom metric functions
def recall_m(y_true, y_pred):
    y_true = K.cast(y_true, 'float32') # Cast y_true to float32
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    y_true = K.cast(y_true, 'float32') # Cast y_true to float32
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):

```

```
precision = precision_m(y_true, y_pred)
recall = recall_m(y_true, y_pred)
return 2 * ((precision * recall) / (precision + recall + K.epsilon()))

def mae_m(y_true, y_pred):
    y_true = K.cast(y_true, 'float32') # Cast y_true to float32
    y_pred = K.cast(y_pred, 'float32') # Cast y_pred to float32
    return K.mean(K.abs(y_pred - y_true))
```

```
In [ ]: # Build the feed-forward neural network model
def build_model():
    model = keras.Sequential()
    model.add(layers.Dense(20, input_dim=20, activation='relu')) # Input
    model.add(layers.Dense(20, activation='relu')) # Hidden Layer
    model.add(layers.Dense(1, activation='sigmoid')) # Output layer for
    # Compile the model
    model.compile(loss='binary_crossentropy', optimizer='adam',
                  metrics=['accuracy', f1_m, precision_m, recall_m, mae_m])
    return model

# Instantiate the model
model = build_model()

# Fit the model
start = time.time()
model.fit(x_train_scaled, y_train, epochs=200, batch_size=2000, verbose=2)
end_train = time.time()
```

Epoch 1/200  
71/71 - 2s - 31ms/step - accuracy: 0.8870 - f1\_m: 0.8602 - loss: 0.4244 - mae\_m: 0.3215 - precision\_m: 0.9111 - recall\_m: 0.8458  
Epoch 2/200  
71/71 - 0s - 2ms/step - accuracy: 0.9841 - f1\_m: 0.9842 - loss: 0.1180 - mae\_m: 0.0847 - precision\_m: 0.9738 - recall\_m: 0.9949  
Epoch 3/200  
71/71 - 0s - 2ms/step - accuracy: 0.9875 - f1\_m: 0.9876 - loss: 0.0717 - mae\_m: 0.0391 - precision\_m: 0.9790 - recall\_m: 0.9964  
Epoch 4/200  
71/71 - 0s - 2ms/step - accuracy: 0.9885 - f1\_m: 0.9886 - loss: 0.0598 - mae\_m: 0.0292 - precision\_m: 0.9807 - recall\_m: 0.9966  
Epoch 5/200  
71/71 - 0s - 2ms/step - accuracy: 0.9891 - f1\_m: 0.9892 - loss: 0.0538 - mae\_m: 0.0253 - precision\_m: 0.9818 - recall\_m: 0.9966  
Epoch 6/200  
71/71 - 0s - 2ms/step - accuracy: 0.9897 - f1\_m: 0.9898 - loss: 0.0495 - mae\_m: 0.0226 - precision\_m: 0.9823 - recall\_m: 0.9974  
Epoch 7/200  
71/71 - 0s - 2ms/step - accuracy: 0.9905 - f1\_m: 0.9906 - loss: 0.0464 - mae\_m: 0.0211 - precision\_m: 0.9827 - recall\_m: 0.9986  
Epoch 8/200  
71/71 - 0s - 2ms/step - accuracy: 0.9912 - f1\_m: 0.9913 - loss: 0.0439 - mae\_m: 0.0202 - precision\_m: 0.9831 - recall\_m: 0.9996  
Epoch 9/200  
71/71 - 0s - 2ms/step - accuracy: 0.9913 - f1\_m: 0.9914 - loss: 0.0421 - mae\_m: 0.0190 - precision\_m: 0.9833 - recall\_m: 0.9997  
Epoch 10/200  
71/71 - 0s - 2ms/step - accuracy: 0.9914 - f1\_m: 0.9915 - loss: 0.0404 - mae\_m: 0.0186 - precision\_m: 0.9834 - recall\_m: 0.9997  
Epoch 11/200  
71/71 - 0s - 2ms/step - accuracy: 0.9915 - f1\_m: 0.9915 - loss: 0.0390 - mae\_m: 0.0179 - precision\_m: 0.9835 - recall\_m: 0.9997  
Epoch 12/200  
71/71 - 0s - 2ms/step - accuracy: 0.9916 - f1\_m: 0.9917 - loss: 0.0378 - mae\_m: 0.0173 - precision\_m: 0.9839 - recall\_m: 0.9996  
Epoch 13/200  
71/71 - 0s - 2ms/step - accuracy: 0.9919 - f1\_m: 0.9920 - loss: 0.0369 - mae\_m: 0.0169 - precision\_m: 0.9845 - recall\_m: 0.9996  
Epoch 14/200  
71/71 - 0s - 3ms/step - accuracy: 0.9921 - f1\_m: 0.9922 - loss: 0.0362 - mae\_m: 0.0164 - precision\_m: 0.9849 - recall\_m: 0.9996  
Epoch 15/200  
71/71 - 0s - 2ms/step - accuracy: 0.9922 - f1\_m: 0.9923 - loss: 0.0357 - mae\_m: 0.0161 - precision\_m: 0.9852 - recall\_m: 0.9995  
Epoch 16/200  
71/71 - 0s - 2ms/step - accuracy: 0.9923 - f1\_m: 0.9923 - loss: 0.0353 - mae\_m: 0.0159 - precision\_m: 0.9852 - recall\_m: 0.9996  
Epoch 17/200  
71/71 - 0s - 2ms/step - accuracy: 0.9923 - f1\_m: 0.9923 - loss: 0.0350 - mae\_m: 0.0155 - precision\_m: 0.9852 - recall\_m: 0.9995  
Epoch 18/200  
71/71 - 0s - 2ms/step - accuracy: 0.9924 - f1\_m: 0.9925 - loss: 0.0346 - mae\_m: 0.0155 - precision\_m: 0.9855 - recall\_m: 0.9995  
Epoch 19/200  
71/71 - 0s - 2ms/step - accuracy: 0.9924 - f1\_m: 0.9924 - loss: 0.0344 - mae\_m: 0.0152 - precision\_m: 0.9854 - recall\_m: 0.9995  
Epoch 20/200  
71/71 - 0s - 2ms/step - accuracy: 0.9924 - f1\_m: 0.9924 - loss: 0.0342 - mae\_m: 0.0152 - precision\_m: 0.9855 - recall\_m: 0.9995

Epoch 21/200  
71/71 - 0s - 2ms/step - accuracy: 0.9924 - f1\_m: 0.9924 - loss: 0.0340 - m  
ae\_m: 0.0150 - precision\_m: 0.9855 - recall\_m: 0.9996  
Epoch 22/200  
71/71 - 0s - 2ms/step - accuracy: 0.9924 - f1\_m: 0.9925 - loss: 0.0338 - m  
ae\_m: 0.0150 - precision\_m: 0.9855 - recall\_m: 0.9995  
Epoch 23/200  
71/71 - 0s - 2ms/step - accuracy: 0.9924 - f1\_m: 0.9925 - loss: 0.0337 - m  
ae\_m: 0.0148 - precision\_m: 0.9855 - recall\_m: 0.9996  
Epoch 24/200  
71/71 - 0s - 2ms/step - accuracy: 0.9924 - f1\_m: 0.9925 - loss: 0.0336 - m  
ae\_m: 0.0147 - precision\_m: 0.9855 - recall\_m: 0.9996  
Epoch 25/200  
71/71 - 0s - 2ms/step - accuracy: 0.9924 - f1\_m: 0.9925 - loss: 0.0334 - m  
ae\_m: 0.0148 - precision\_m: 0.9855 - recall\_m: 0.9996  
Epoch 26/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0333 - m  
ae\_m: 0.0147 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 27/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0332 - m  
ae\_m: 0.0148 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 28/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0332 - m  
ae\_m: 0.0145 - precision\_m: 0.9855 - recall\_m: 0.9996  
Epoch 29/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0331 - m  
ae\_m: 0.0147 - precision\_m: 0.9856 - recall\_m: 0.9995  
Epoch 30/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0330 - m  
ae\_m: 0.0146 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 31/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0329 - m  
ae\_m: 0.0145 - precision\_m: 0.9855 - recall\_m: 0.9995  
Epoch 32/200  
71/71 - 0s - 3ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0329 - m  
ae\_m: 0.0146 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 33/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0328 - m  
ae\_m: 0.0144 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 34/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0327 - m  
ae\_m: 0.0144 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 35/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0327 - m  
ae\_m: 0.0144 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 36/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0326 - m  
ae\_m: 0.0144 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 37/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0325 - m  
ae\_m: 0.0144 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 38/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0324 - m  
ae\_m: 0.0144 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 39/200  
71/71 - 0s - 3ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0324 - m  
ae\_m: 0.0144 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 40/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0324 - m  
ae\_m: 0.0143 - precision\_m: 0.9857 - recall\_m: 0.9996

Epoch 41/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0323 - m  
ae\_m: 0.0143 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 42/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0322 - m  
ae\_m: 0.0142 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 43/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0322 - m  
ae\_m: 0.0142 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 44/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0321 - m  
ae\_m: 0.0144 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 45/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0321 - m  
ae\_m: 0.0142 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 46/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0321 - m  
ae\_m: 0.0142 - precision\_m: 0.9856 - recall\_m: 0.9997  
Epoch 47/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0319 - m  
ae\_m: 0.0143 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 48/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9925 - loss: 0.0319 - m  
ae\_m: 0.0142 - precision\_m: 0.9856 - recall\_m: 0.9996  
Epoch 49/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0319 - m  
ae\_m: 0.0142 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 50/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0319 - m  
ae\_m: 0.0141 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 51/200  
71/71 - 0s - 2ms/step - accuracy: 0.9925 - f1\_m: 0.9926 - loss: 0.0319 - m  
ae\_m: 0.0140 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 52/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0318 - m  
ae\_m: 0.0143 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 53/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0317 - m  
ae\_m: 0.0141 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 54/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0317 - m  
ae\_m: 0.0142 - precision\_m: 0.9856 - recall\_m: 0.9997  
Epoch 55/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0317 - m  
ae\_m: 0.0140 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 56/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0316 - m  
ae\_m: 0.0141 - precision\_m: 0.9858 - recall\_m: 0.9996  
Epoch 57/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0316 - m  
ae\_m: 0.0140 - precision\_m: 0.9858 - recall\_m: 0.9996  
Epoch 58/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0315 - m  
ae\_m: 0.0142 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 59/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0315 - m  
ae\_m: 0.0141 - precision\_m: 0.9857 - recall\_m: 0.9996  
Epoch 60/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0315 - m  
ae\_m: 0.0140 - precision\_m: 0.9858 - recall\_m: 0.9997

Epoch 61/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0314 - m  
ae\_m: 0.0141 - precision\_m: 0.9856 - recall\_m: 0.9997  
Epoch 62/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0314 - m  
ae\_m: 0.0140 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 63/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0314 - m  
ae\_m: 0.0141 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 64/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0313 - m  
ae\_m: 0.0140 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 65/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0313 - m  
ae\_m: 0.0140 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 66/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0312 - m  
ae\_m: 0.0141 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 67/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0312 - m  
ae\_m: 0.0141 - precision\_m: 0.9858 - recall\_m: 0.9996  
Epoch 68/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0312 - m  
ae\_m: 0.0138 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 69/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0311 - m  
ae\_m: 0.0141 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 70/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0311 - m  
ae\_m: 0.0140 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 71/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0311 - m  
ae\_m: 0.0139 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 72/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0310 - m  
ae\_m: 0.0141 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 73/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0310 - m  
ae\_m: 0.0138 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 74/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0310 - m  
ae\_m: 0.0140 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 75/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0310 - m  
ae\_m: 0.0140 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 76/200  
71/71 - 0s - 3ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0309 - m  
ae\_m: 0.0138 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 77/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0310 - m  
ae\_m: 0.0139 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 78/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0309 - m  
ae\_m: 0.0141 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 79/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0308 - m  
ae\_m: 0.0139 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 80/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0308 - m  
ae\_m: 0.0138 - precision\_m: 0.9858 - recall\_m: 0.9997

Epoch 81/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0308 - m  
ae\_m: 0.0140 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 82/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0309 - m  
ae\_m: 0.0140 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 83/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0307 - m  
ae\_m: 0.0138 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 84/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0307 - m  
ae\_m: 0.0140 - precision\_m: 0.9859 - recall\_m: 0.9997  
Epoch 85/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0307 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 86/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9926 - loss: 0.0307 - m  
ae\_m: 0.0139 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 87/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0306 - m  
ae\_m: 0.0139 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 88/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0306 - m  
ae\_m: 0.0139 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 89/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0307 - m  
ae\_m: 0.0139 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 90/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0306 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 91/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0306 - m  
ae\_m: 0.0139 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 92/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0305 - m  
ae\_m: 0.0138 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 93/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0305 - m  
ae\_m: 0.0138 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 94/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0305 - m  
ae\_m: 0.0139 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 95/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0305 - m  
ae\_m: 0.0138 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 96/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9926 - loss: 0.0305 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 97/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0305 - m  
ae\_m: 0.0140 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 98/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0304 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 99/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0305 - m  
ae\_m: 0.0139 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 100/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9926 - loss: 0.0304 - m  
ae\_m: 0.0138 - precision\_m: 0.9857 - recall\_m: 0.9997

Epoch 101/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0304 - m  
ae\_m: 0.0140 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 102/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0304 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 103/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0303 - m  
ae\_m: 0.0139 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 104/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0303 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 105/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0304 - m  
ae\_m: 0.0138 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 106/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0303 - m  
ae\_m: 0.0139 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 107/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0303 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 108/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0302 - m  
ae\_m: 0.0138 - precision\_m: 0.9857 - recall\_m: 0.9998  
Epoch 109/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0302 - m  
ae\_m: 0.0140 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 110/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0302 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 111/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0302 - m  
ae\_m: 0.0139 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 112/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0302 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 113/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0302 - m  
ae\_m: 0.0139 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 114/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0302 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 115/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0301 - m  
ae\_m: 0.0138 - precision\_m: 0.9859 - recall\_m: 0.9997  
Epoch 116/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0302 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 117/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0301 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 118/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0301 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 119/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0300 - m  
ae\_m: 0.0138 - precision\_m: 0.9859 - recall\_m: 0.9997  
Epoch 120/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0301 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9997

Epoch 121/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0300 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 122/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0300 - m  
ae\_m: 0.0137 - precision\_m: 0.9859 - recall\_m: 0.9997  
Epoch 123/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0301 - m  
ae\_m: 0.0137 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 124/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0301 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 125/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0300 - m  
ae\_m: 0.0139 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 126/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0299 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9998  
Epoch 127/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0300 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 128/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0300 - m  
ae\_m: 0.0138 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 129/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0299 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9997  
Epoch 130/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0300 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 131/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0299 - m  
ae\_m: 0.0137 - precision\_m: 0.9859 - recall\_m: 0.9997  
Epoch 132/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0299 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 133/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0298 - m  
ae\_m: 0.0137 - precision\_m: 0.9859 - recall\_m: 0.9997  
Epoch 134/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0299 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 135/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0298 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 136/200  
71/71 - 0s - 2ms/step - accuracy: 0.9926 - f1\_m: 0.9927 - loss: 0.0299 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 137/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0299 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 138/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0298 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 139/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0298 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 140/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0298 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997

Epoch 141/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0299 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 142/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0298 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 143/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0298 - m  
ae\_m: 0.0138 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 144/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0298 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 145/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0298 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 146/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0298 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 147/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0299 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 148/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0297 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 149/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0297 - m  
ae\_m: 0.0135 - precision\_m: 0.9859 - recall\_m: 0.9997  
Epoch 150/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0297 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 151/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0297 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 152/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0297 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9999  
Epoch 153/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0296 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 154/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0296 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 155/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0296 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 156/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0296 - m  
ae\_m: 0.0136 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 157/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0295 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9997  
Epoch 158/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0295 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 159/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0296 - m  
ae\_m: 0.0135 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 160/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0296 - m  
ae\_m: 0.0136 - precision\_m: 0.9857 - recall\_m: 0.9997

Epoch 161/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0295 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 162/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0295 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 163/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0295 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 164/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0296 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 165/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0295 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9999  
Epoch 166/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 167/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0295 - m  
ae\_m: 0.0136 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 168/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0295 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 169/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0295 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 170/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 171/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0294 - m  
ae\_m: 0.0136 - precision\_m: 0.9857 - recall\_m: 0.9998  
Epoch 172/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0137 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 173/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0136 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 174/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9999  
Epoch 175/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 176/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0295 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 177/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0293 - m  
ae\_m: 0.0136 - precision\_m: 0.9857 - recall\_m: 0.9999  
Epoch 178/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0294 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 179/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0295 - m  
ae\_m: 0.0137 - precision\_m: 0.9857 - recall\_m: 0.9998  
Epoch 180/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0293 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998

Epoch 181/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0295 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 182/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0293 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 183/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0294 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 184/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0293 - m  
ae\_m: 0.0135 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 185/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0135 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 186/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0136 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 187/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0293 - m  
ae\_m: 0.0136 - precision\_m: 0.9860 - recall\_m: 0.9998  
Epoch 188/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0293 - m  
ae\_m: 0.0134 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 189/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0293 - m  
ae\_m: 0.0136 - precision\_m: 0.9859 - recall\_m: 0.9999  
Epoch 190/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0292 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9999  
Epoch 191/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0294 - m  
ae\_m: 0.0135 - precision\_m: 0.9859 - recall\_m: 0.9998  
Epoch 192/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0292 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9999  
Epoch 193/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0292 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 194/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0292 - m  
ae\_m: 0.0136 - precision\_m: 0.9859 - recall\_m: 0.9999  
Epoch 195/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0293 - m  
ae\_m: 0.0134 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 196/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0293 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998  
Epoch 197/200  
71/71 - 0s - 2ms/step - accuracy: 0.9928 - f1\_m: 0.9928 - loss: 0.0292 - m  
ae\_m: 0.0135 - precision\_m: 0.9859 - recall\_m: 0.9999  
Epoch 198/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9927 - loss: 0.0292 - m  
ae\_m: 0.0136 - precision\_m: 0.9857 - recall\_m: 0.9998  
Epoch 199/200  
71/71 - 0s - 2ms/step - accuracy: 0.9928 - f1\_m: 0.9928 - loss: 0.0292 - m  
ae\_m: 0.0136 - precision\_m: 0.9859 - recall\_m: 0.9999  
Epoch 200/200  
71/71 - 0s - 2ms/step - accuracy: 0.9927 - f1\_m: 0.9928 - loss: 0.0292 - m  
ae\_m: 0.0135 - precision\_m: 0.9858 - recall\_m: 0.9998

```
In [ ]: # Evaluate the model
loss, accuracy, f1s, precision, recall, mae = model.evaluate(x_val_scaled)
end_predict = time.time()

# Record the performance in the DataFrame
model_performance.loc['MLP (Keras)'] = [accuracy, recall, precision, f1s,
```

**1100/1100** ————— 1s 940us/step - accuracy: 0.9923 - f1\_m: 0.9921 - loss: 0.0326 - mae\_m: 0.0126 - precision\_m: 0.9851 - recall\_m: 0.9927

```
In [ ]: import os
model_directory = "model"
if not os.path.exists(model_directory):
    os.makedirs(model_directory)
model_path = os.path.join(model_directory, "nnkeras.keras")
model.save(model_path)
```

## GRU (Keras)

```
In [ ]: from keras.layers import GRU, Dense
from keras.models import Sequential
import numpy as np

# Build the neural network model
def build_model():
    model = Sequential()
    model.add(GRU(20, return_sequences=True, input_shape=(1, 20)))
    model.add(GRU(20, return_sequences=False)) # Set return_sequences=False
    model.add(Dense(1, activation='sigmoid')) # For binary classification
    # Compile the model
    model.compile(loss='binary_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model

# The GRU input Layer must be 3D.
# The meaning of the 3 input dimensions are: samples, time steps, and features

# Reshape input data
X_train_array = np.array(x_train_scaled)
X_train_reshaped = X_train_array.reshape(X_train_array.shape[0], 1, 20)

# Reshape validation data
X_test_array = np.array(x_val_scaled)
X_test_reshaped = X_test_array.reshape(X_test_array.shape[0], 1, 20)

# Instantiate the model
model = build_model()

start = time.time()
# Fit the model
model.fit(X_train_reshaped, y_train, epochs=200, batch_size=2000, verbose=0)
end_train = time.time()
```

Epoch 1/200  
71/71 - 4s - 51ms/step - accuracy: 0.8980 - loss: 0.5017  
Epoch 2/200  
71/71 - 0s - 5ms/step - accuracy: 0.9785 - loss: 0.1522  
Epoch 3/200  
71/71 - 0s - 5ms/step - accuracy: 0.9859 - loss: 0.0736  
Epoch 4/200  
71/71 - 0s - 5ms/step - accuracy: 0.9875 - loss: 0.0582  
Epoch 5/200  
71/71 - 0s - 5ms/step - accuracy: 0.9883 - loss: 0.0517  
Epoch 6/200  
71/71 - 0s - 5ms/step - accuracy: 0.9895 - loss: 0.0480  
Epoch 7/200  
71/71 - 0s - 5ms/step - accuracy: 0.9903 - loss: 0.0455  
Epoch 8/200  
71/71 - 0s - 5ms/step - accuracy: 0.9906 - loss: 0.0436  
Epoch 9/200  
71/71 - 0s - 4ms/step - accuracy: 0.9908 - loss: 0.0420  
Epoch 10/200  
71/71 - 0s - 4ms/step - accuracy: 0.9911 - loss: 0.0406  
Epoch 11/200  
71/71 - 0s - 5ms/step - accuracy: 0.9913 - loss: 0.0393  
Epoch 12/200  
71/71 - 0s - 4ms/step - accuracy: 0.9914 - loss: 0.0383  
Epoch 13/200  
71/71 - 0s - 4ms/step - accuracy: 0.9916 - loss: 0.0375  
Epoch 14/200  
71/71 - 0s - 5ms/step - accuracy: 0.9919 - loss: 0.0368  
Epoch 15/200  
71/71 - 0s - 5ms/step - accuracy: 0.9922 - loss: 0.0363  
Epoch 16/200  
71/71 - 0s - 4ms/step - accuracy: 0.9923 - loss: 0.0358  
Epoch 17/200  
71/71 - 0s - 5ms/step - accuracy: 0.9923 - loss: 0.0354  
Epoch 18/200  
71/71 - 0s - 4ms/step - accuracy: 0.9923 - loss: 0.0351  
Epoch 19/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0349  
Epoch 20/200  
71/71 - 0s - 5ms/step - accuracy: 0.9923 - loss: 0.0346  
Epoch 21/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0344  
Epoch 22/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0342  
Epoch 23/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0340  
Epoch 24/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0339  
Epoch 25/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0337  
Epoch 26/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0336  
Epoch 27/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0334  
Epoch 28/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0333  
Epoch 29/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0332  
Epoch 30/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0331

Epoch 31/200  
71/71 - 0s - 4ms/step - accuracy: 0.9924 - loss: 0.0330  
Epoch 32/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0329  
Epoch 33/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0328  
Epoch 34/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0327  
Epoch 35/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0326  
Epoch 36/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0325  
Epoch 37/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0325  
Epoch 38/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0324  
Epoch 39/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0323  
Epoch 40/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0323  
Epoch 41/200  
71/71 - 0s - 4ms/step - accuracy: 0.9924 - loss: 0.0322  
Epoch 42/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0321  
Epoch 43/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0321  
Epoch 44/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0320  
Epoch 45/200  
71/71 - 0s - 4ms/step - accuracy: 0.9924 - loss: 0.0320  
Epoch 46/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0319  
Epoch 47/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0319  
Epoch 48/200  
71/71 - 1s - 8ms/step - accuracy: 0.9924 - loss: 0.0318  
Epoch 49/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0317  
Epoch 50/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0317  
Epoch 51/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0316  
Epoch 52/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0316  
Epoch 53/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0315  
Epoch 54/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0315  
Epoch 55/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0314  
Epoch 56/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0314  
Epoch 57/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0313  
Epoch 58/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0313  
Epoch 59/200  
71/71 - 0s - 5ms/step - accuracy: 0.9925 - loss: 0.0313  
Epoch 60/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0312

Epoch 61/200  
71/71 - 0s - 4ms/step - accuracy: 0.9924 - loss: 0.0312  
Epoch 62/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0312  
Epoch 63/200  
71/71 - 0s - 5ms/step - accuracy: 0.9924 - loss: 0.0311  
Epoch 64/200  
71/71 - 0s - 5ms/step - accuracy: 0.9925 - loss: 0.0310  
Epoch 65/200  
71/71 - 0s - 5ms/step - accuracy: 0.9925 - loss: 0.0310  
Epoch 66/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0310  
Epoch 67/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0310  
Epoch 68/200  
71/71 - 0s - 5ms/step - accuracy: 0.9925 - loss: 0.0310  
Epoch 69/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0309  
Epoch 70/200  
71/71 - 0s - 5ms/step - accuracy: 0.9925 - loss: 0.0308  
Epoch 71/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0308  
Epoch 72/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0308  
Epoch 73/200  
71/71 - 0s - 5ms/step - accuracy: 0.9925 - loss: 0.0307  
Epoch 74/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0307  
Epoch 75/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0306  
Epoch 76/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0306  
Epoch 77/200  
71/71 - 0s - 5ms/step - accuracy: 0.9926 - loss: 0.0306  
Epoch 78/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0306  
Epoch 79/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0306  
Epoch 80/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0305  
Epoch 81/200  
71/71 - 0s - 5ms/step - accuracy: 0.9926 - loss: 0.0305  
Epoch 82/200  
71/71 - 0s - 5ms/step - accuracy: 0.9926 - loss: 0.0304  
Epoch 83/200  
71/71 - 0s - 5ms/step - accuracy: 0.9926 - loss: 0.0304  
Epoch 84/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0304  
Epoch 85/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0304  
Epoch 86/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0304  
Epoch 87/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0303  
Epoch 88/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0302  
Epoch 89/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0303  
Epoch 90/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0302

Epoch 91/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0302  
Epoch 92/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0302  
Epoch 93/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0301  
Epoch 94/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0301  
Epoch 95/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0301  
Epoch 96/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0301  
Epoch 97/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0301  
Epoch 98/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 99/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 100/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 101/200  
71/71 - 0s - 6ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 102/200  
71/71 - 0s - 6ms/step - accuracy: 0.9927 - loss: 0.0299  
Epoch 103/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0299  
Epoch 104/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 105/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0299  
Epoch 106/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0299  
Epoch 107/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 108/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0299  
Epoch 109/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0299  
Epoch 110/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 111/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 112/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 113/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 114/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 115/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 116/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 117/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 118/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 119/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 120/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0297

Epoch 121/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 122/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0297  
Epoch 123/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 124/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 125/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 126/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 127/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 128/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 129/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0296  
Epoch 130/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 131/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 132/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0295  
Epoch 133/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0295  
Epoch 134/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0295  
Epoch 135/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0295  
Epoch 136/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0295  
Epoch 137/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0295  
Epoch 138/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0295  
Epoch 139/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0295  
Epoch 140/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0294  
Epoch 141/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 142/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0294  
Epoch 143/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 144/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 145/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 146/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 147/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0294  
Epoch 148/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 149/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 150/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0293

Epoch 151/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 152/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 153/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 154/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 155/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 156/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 157/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 158/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 159/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 160/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 161/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 162/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 163/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 164/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 165/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 166/200  
71/71 - 1s - 9ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 167/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0292  
Epoch 168/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 169/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 170/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 171/200  
71/71 - 0s - 6ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 172/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0291  
Epoch 173/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 174/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 175/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 176/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 177/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 178/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 179/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 180/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290

```

Epoch 181/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0290
Epoch 182/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291
Epoch 183/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291
Epoch 184/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290
Epoch 185/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0290
Epoch 186/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290
Epoch 187/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291
Epoch 188/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290
Epoch 189/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 190/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 191/200
71/71 - 1s - 8ms/step - accuracy: 0.9928 - loss: 0.0290
Epoch 192/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 193/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 194/200
71/71 - 0s - 6ms/step - accuracy: 0.9928 - loss: 0.0290
Epoch 195/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 196/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 197/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 198/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 199/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 200/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289

```

```
In [ ]: loss, accuracy = model.evaluate(X_test_reshaped, y_val)
# Loss, accuracy, f1s, precision, recall = model.evaluate(X_test_reshaped
end_predict = time.time()
model_performance.loc['GRU (Keras)'] = [accuracy, accuracy, accuracy, acc
```

```
1100/1100 ━━━━━━━━ 1s 922us/step - accuracy: 0.9924 - loss: 0.0308
```

```
In [ ]: np.shape(X_train_reshaped)
```

```
Out[ ]: (140800, 1, 20)
```

```
In [ ]: import os
model_directory = "model"
if not os.path.exists(model_directory):
    os.makedirs(model_directory)
model_path = os.path.join(model_directory, "grukeras.keras")
model.save(model_path)
```

## LSTM (Keras)

```
In [ ]: from keras.layers import LSTM, Dense
from keras.models import Sequential
import numpy as np

def build_model():
    model = Sequential()
    model.add(LSTM(20, return_sequences=True, input_shape=(1, 20)))
    model.add(LSTM(20, return_sequences=False)) # Set return_sequences=False
    model.add(Dense(1, activation='sigmoid')) # Use 1 neuron and sigmoid
    # Compile the model
    model.compile(loss='binary_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model

# The LSTM input Layer must be 3D.
# The meaning of the 3 input dimensions are: samples, time steps, and features

# Reshape input data
X_train_array = np.array(x_train_scaled)
X_train_reshaped = X_train_array.reshape(X_train_array.shape[0], 1, 20)

# Reshape validation data
X_test_array = np.array(x_val_scaled)
X_test_reshaped = X_test_array.reshape(X_test_array.shape[0], 1, 20)

# Instantiate the model
model = build_model()

# Fit the model
start = time.time()
model.fit(X_train_reshaped, y_train, epochs=200, batch_size=2000, verbose=1)
end_train = time.time()
```

Epoch 1/200  
71/71 - 3s - 37ms/step - accuracy: 0.8491 - loss: 0.6363  
Epoch 2/200  
71/71 - 0s - 4ms/step - accuracy: 0.9781 - loss: 0.3242  
Epoch 3/200  
71/71 - 0s - 4ms/step - accuracy: 0.9829 - loss: 0.1060  
Epoch 4/200  
71/71 - 0s - 4ms/step - accuracy: 0.9866 - loss: 0.0673  
Epoch 5/200  
71/71 - 0s - 4ms/step - accuracy: 0.9890 - loss: 0.0570  
Epoch 6/200  
71/71 - 0s - 5ms/step - accuracy: 0.9894 - loss: 0.0519  
Epoch 7/200  
71/71 - 0s - 4ms/step - accuracy: 0.9900 - loss: 0.0485  
Epoch 8/200  
71/71 - 0s - 5ms/step - accuracy: 0.9909 - loss: 0.0460  
Epoch 9/200  
71/71 - 0s - 5ms/step - accuracy: 0.9912 - loss: 0.0443  
Epoch 10/200  
71/71 - 0s - 5ms/step - accuracy: 0.9914 - loss: 0.0430  
Epoch 11/200  
71/71 - 0s - 5ms/step - accuracy: 0.9915 - loss: 0.0419  
Epoch 12/200  
71/71 - 0s - 5ms/step - accuracy: 0.9915 - loss: 0.0411  
Epoch 13/200  
71/71 - 0s - 4ms/step - accuracy: 0.9916 - loss: 0.0402  
Epoch 14/200  
71/71 - 0s - 4ms/step - accuracy: 0.9916 - loss: 0.0393  
Epoch 15/200  
71/71 - 0s - 6ms/step - accuracy: 0.9916 - loss: 0.0384  
Epoch 16/200  
71/71 - 0s - 5ms/step - accuracy: 0.9916 - loss: 0.0376  
Epoch 17/200  
71/71 - 0s - 5ms/step - accuracy: 0.9917 - loss: 0.0369  
Epoch 18/200  
71/71 - 0s - 5ms/step - accuracy: 0.9917 - loss: 0.0362  
Epoch 19/200  
71/71 - 0s - 5ms/step - accuracy: 0.9917 - loss: 0.0357  
Epoch 20/200  
71/71 - 0s - 5ms/step - accuracy: 0.9917 - loss: 0.0353  
Epoch 21/200  
71/71 - 0s - 6ms/step - accuracy: 0.9917 - loss: 0.0350  
Epoch 22/200  
71/71 - 0s - 5ms/step - accuracy: 0.9917 - loss: 0.0347  
Epoch 23/200  
71/71 - 0s - 5ms/step - accuracy: 0.9917 - loss: 0.0345  
Epoch 24/200  
71/71 - 0s - 4ms/step - accuracy: 0.9917 - loss: 0.0343  
Epoch 25/200  
71/71 - 0s - 5ms/step - accuracy: 0.9917 - loss: 0.0341  
Epoch 26/200  
71/71 - 0s - 5ms/step - accuracy: 0.9917 - loss: 0.0339  
Epoch 27/200  
71/71 - 0s - 4ms/step - accuracy: 0.9918 - loss: 0.0337  
Epoch 28/200  
71/71 - 0s - 4ms/step - accuracy: 0.9918 - loss: 0.0335  
Epoch 29/200  
71/71 - 0s - 5ms/step - accuracy: 0.9918 - loss: 0.0334  
Epoch 30/200  
71/71 - 0s - 4ms/step - accuracy: 0.9918 - loss: 0.0332

Epoch 31/200  
71/71 - 0s - 4ms/step - accuracy: 0.9920 - loss: 0.0331  
Epoch 32/200  
71/71 - 0s - 4ms/step - accuracy: 0.9921 - loss: 0.0330  
Epoch 33/200  
71/71 - 0s - 4ms/step - accuracy: 0.9922 - loss: 0.0329  
Epoch 34/200  
71/71 - 0s - 4ms/step - accuracy: 0.9923 - loss: 0.0328  
Epoch 35/200  
71/71 - 0s - 4ms/step - accuracy: 0.9924 - loss: 0.0327  
Epoch 36/200  
71/71 - 0s - 4ms/step - accuracy: 0.9924 - loss: 0.0326  
Epoch 37/200  
71/71 - 0s - 4ms/step - accuracy: 0.9924 - loss: 0.0325  
Epoch 38/200  
71/71 - 0s - 5ms/step - accuracy: 0.9925 - loss: 0.0325  
Epoch 39/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0324  
Epoch 40/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0323  
Epoch 41/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0322  
Epoch 42/200  
71/71 - 0s - 5ms/step - accuracy: 0.9925 - loss: 0.0322  
Epoch 43/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0321  
Epoch 44/200  
71/71 - 0s - 5ms/step - accuracy: 0.9926 - loss: 0.0320  
Epoch 45/200  
71/71 - 0s - 4ms/step - accuracy: 0.9925 - loss: 0.0320  
Epoch 46/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0319  
Epoch 47/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0318  
Epoch 48/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0318  
Epoch 49/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0317  
Epoch 50/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0317  
Epoch 51/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0316  
Epoch 52/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0316  
Epoch 53/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0315  
Epoch 54/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0315  
Epoch 55/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0314  
Epoch 56/200  
71/71 - 0s - 5ms/step - accuracy: 0.9926 - loss: 0.0313  
Epoch 57/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0313  
Epoch 58/200  
71/71 - 0s - 5ms/step - accuracy: 0.9926 - loss: 0.0312  
Epoch 59/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0312  
Epoch 60/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0311

Epoch 61/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0311  
Epoch 62/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0310  
Epoch 63/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0310  
Epoch 64/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0310  
Epoch 65/200  
71/71 - 0s - 4ms/step - accuracy: 0.9926 - loss: 0.0309  
Epoch 66/200  
71/71 - 0s - 5ms/step - accuracy: 0.9926 - loss: 0.0309  
Epoch 67/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0308  
Epoch 68/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0308  
Epoch 69/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0307  
Epoch 70/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0307  
Epoch 71/200  
71/71 - 0s - 7ms/step - accuracy: 0.9927 - loss: 0.0306  
Epoch 72/200  
71/71 - 0s - 6ms/step - accuracy: 0.9927 - loss: 0.0306  
Epoch 73/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0305  
Epoch 74/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0305  
Epoch 75/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0304  
Epoch 76/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0304  
Epoch 77/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0304  
Epoch 78/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0303  
Epoch 79/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0302  
Epoch 80/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0302  
Epoch 81/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0302  
Epoch 82/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0301  
Epoch 83/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0301  
Epoch 84/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0301  
Epoch 85/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 86/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 87/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 88/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 89/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0300  
Epoch 90/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0299

Epoch 91/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0299  
Epoch 92/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0299  
Epoch 93/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 94/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 95/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 96/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0298  
Epoch 97/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 98/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 99/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 100/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 101/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0297  
Epoch 102/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 103/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 104/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 105/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 106/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 107/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0295  
Epoch 108/200  
71/71 - 1s - 9ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 109/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0296  
Epoch 110/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0295  
Epoch 111/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 112/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0295  
Epoch 113/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0295  
Epoch 114/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 115/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 116/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 117/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 118/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0294  
Epoch 119/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 120/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0294

Epoch 121/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 122/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 123/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 124/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 125/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0293  
Epoch 126/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 127/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0293  
Epoch 128/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 129/200  
71/71 - 0s - 4ms/step - accuracy: 0.9927 - loss: 0.0293  
Epoch 130/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0293  
Epoch 131/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 132/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0292  
Epoch 133/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 134/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 135/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 136/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 137/200  
71/71 - 0s - 5ms/step - accuracy: 0.9927 - loss: 0.0292  
Epoch 138/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 139/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0292  
Epoch 140/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 141/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 142/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 143/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 144/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 145/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 146/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 147/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 148/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 149/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 150/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291

Epoch 151/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 152/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 153/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 154/200  
71/71 - 1s - 9ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 155/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 156/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0291  
Epoch 157/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 158/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 159/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 160/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 161/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 162/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 163/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 164/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 165/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 166/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0290  
Epoch 167/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 168/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 169/200  
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0288  
Epoch 170/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 171/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 172/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288  
Epoch 173/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288  
Epoch 174/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 175/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0289  
Epoch 176/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288  
Epoch 177/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288  
Epoch 178/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288  
Epoch 179/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288  
Epoch 180/200  
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288

```

Epoch 181/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 182/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 183/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 184/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 185/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 186/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 187/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 188/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 189/200
71/71 - 0s - 6ms/step - accuracy: 0.9928 - loss: 0.0289
Epoch 190/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 191/200
71/71 - 1s - 9ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 192/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 193/200
71/71 - 1s - 10ms/step - accuracy: 0.9928 - loss: 0.0288
Epoch 194/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 195/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 196/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 197/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 198/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 199/200
71/71 - 0s - 4ms/step - accuracy: 0.9928 - loss: 0.0287
Epoch 200/200
71/71 - 0s - 5ms/step - accuracy: 0.9928 - loss: 0.0288

```

```

In [ ]: #Evaluate the neural network
loss, accuracy = model.evaluate(X_test_reshaped, y_val)
# loss, accuracy, f1s, precision, recall = model.evaluate(X_test_reshaped
end_predict = time.time()
model_performance.loc['LSTM (Keras)'] = [accuracy, accuracy, accuracy, ac
1100/1100 ━━━━━━━━ 1s 910us/step - accuracy: 0.9923 - loss: 0.
0326

```

```

In [ ]: import os
model_directory = "model"
if not os.path.exists(model_directory):
    os.makedirs(model_directory)
model_path = os.path.join(model_directory, "lstmkeras.keras")
model.save(model_path)

```

## 8 | Evaluation with Keras Models

In [ ]:

```
model_performance.fillna(.90,inplace=True)
model_performance.style.background_gradient(cmap='coolwarm').format({'Acc': '{:.2%}', 'Pre': '{:.2%}', 'Rec': '{:.2%}', 'F1': '{:.2%}', 'MAE': '{:.2f}', 'time': '{:.2f}', 'time_norm': '{:.2f}', 'total_time': '{:.2f}'})
```

Out[ ]:

	Accuracy	Recall	Precision	F1-Score	Mean Absolute Error	time to train
<b>DecisionTreeClassifier</b>	98.83%	98.86%	98.80%	98.83%	0.011733	1.1
<b>RandomForestClassifier</b>	99.28%	99.96%	98.61%	99.28%	0.007244	5.5
<b>GradientBoostingClassifier</b>	99.27%	99.99%	98.57%	99.28%	0.007301	44.1
<b>LogisticRegression</b>	98.90%	99.58%	98.25%	98.91%	0.010994	0.3
<b>KNeighborsClassifier</b>	99.10%	99.66%	98.56%	99.11%	0.009006	0.0
<b>ExtraTreesClassifier</b>	99.25%	99.91%	98.61%	99.26%	0.007500	3.5
<b>SGDClassifier</b>	98.80%	99.52%	98.11%	98.81%	0.011989	0.4
<b>LinearSVC</b>	99.04%	99.85%	98.26%	99.05%	0.009631	1.7
<b>SVM model (OneClassSVM)</b>	70.53%	60.46%	75.75%	67.25%	0.294716	36.5
<b>Ensemble (without OC)</b>	99.27%	99.27%	99.28%	99.27%	0.007330	56.0
<b>MLP (Keras)</b>	99.25%	99.98%	98.52%	99.22%	0.012375	31.5
<b>GRU (Keras)</b>	99.26%	99.26%	99.26%	99.26%	0.992585	71.7
<b>LSTM (Keras)</b>	99.26%	99.26%	99.26%	99.26%	0.992557	69.5

◀ ▶

## 9 | End of Performance Monitoring

In [ ]:

```
# Record the end time and final system statistics
import json

notebook_end_time = time.time()

end_cpu_times = psutil.cpu_times()
end_memory_info = process.memory_info()
end_disk_io = psutil.disk_io_counters()
end_net_io = psutil.net_io_counters()

# Calculate the total run time
total_run_time = notebook_end_time - notebook_start_time
```

```

# Calculate CPU usage
cpu_usage = {
    'user': end_cpu_times.user - start_cpu_times.user,
    'system': end_cpu_times.system - start_cpu_times.system,
    'idle': end_cpu_times.idle - start_cpu_times.idle
}

# Calculate memory usage
memory_usage = {
    'rss': end_memory_info.rss - start_memory_info.rss,
    'vms': end_memory_info.vms - start_memory_info.vms
}

# Calculate disk I/O
disk_io = {
    'read_count': end_disk_io.read_count - start_disk_io.read_count,
    'write_count': end_disk_io.write_count - start_disk_io.write_count,
    'read_bytes': end_disk_io.read_bytes - start_disk_io.read_bytes,
    'write_bytes': end_disk_io.write_bytes - start_disk_io.write_bytes
}

# Calculate network I/O
net_io = {
    'bytes_sent': end_net_io.bytes_sent - start_net_io.bytes_sent,
    'bytes_recv': end_net_io.bytes_recv - start_net_io.bytes_recv
}

# Save metrics to a text file
metrics = {
    "total_run_time": total_run_time,
    "cpu_usage": cpu_usage,
    "memory_usage": memory_usage,
    "disk_io": disk_io,
    "network_io": net_io
}

with open('system_metrics.txt', 'w') as file:
    file.write(json.dumps(metrics, indent=4))
# Print the statistics
print("Total run time: {:.2f} seconds".format(total_run_time))
print("CPU usage (seconds):", cpu_usage)
print("Memory usage (bytes):", memory_usage)
print("Disk I/O:", disk_io)
print("Network I/O:", net_io)

```

```

Total run time: 926.92 seconds
CPU usage (seconds): {'user': 2239.515625, 'system': 616.6093750000073, 'idle': 2705.40625}
Memory usage (bytes): {'rss': 1094676480, 'vms': 6892978176}
Disk I/O: {'read_count': 225853, 'write_count': 27007, 'read_bytes': 6794255872, 'write_bytes': 5688184832}
Network I/O: {'bytes_sent': 69634802, 'bytes_recv': 7092419}

```