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

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

In [ ]:
```
import time
import psutil
import json
import threading

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

start_cpu_times = process.cpu_times()
```

```
start_memory_info = process.memory_info()
start_disk_io = process.io_counters()
start_net_io = psutil.net_io_counters()

# Initialize lists to store CPU usage data
cpu_percentages = []

# Function to sample CPU usage periodically
def sample_cpu_usage():
    while True:
        cpu_percentages.append(psutil.cpu_percent(interval=None))
        time.sleep(1)  # Sample every 1 second

# Start a background thread to sample CPU usage
cpu_thread = threading.Thread(target=sample_cpu_usage)
cpu_thread.daemon = True  # Daemonize thread to exit when the main program exits
cpu_thread.start()
```

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":"#FFFAF0", "figure.facecolor":"#FFFAF0"})
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[ ]:

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

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

Out[ ]:

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

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

Out[ ]:

	Attack category	Attack subcategory	Number of events
0	normal	NaN	2218761
1	Fuzzers	FTP	558
2	Fuzzers	HTTP	1497
3	Fuzzers	RIP	3550
4	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
0	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	29	0
1	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	29	0
2	59.166.0.5	3593	149.171.126.5	53	udp	CON	0.001209	132	164	31	29	0
3	59.166.0.3	49664	149.171.126.0	53	udp	CON	0.001169	146	178	31	29	0
4	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
0	59.166.0.0	7832	149.171.126.3	5607	tcp	FIN	0.144951	5174	91072	31	29
1	59.166.0.8	11397	149.171.126.6	21	tcp	FIN	0.116107	2934	3742	31	29
2	59.166.0.0	3804	149.171.126.3	53	udp	CON	0.000986	146	178	31	29
3	59.166.0.8	14339	149.171.126.6	14724	tcp	FIN	0.038480	8928	320	31	29
4	59.166.0.8	39094	149.171.126.3	53	udp	CON	0.001026	130	162	31	29

In [ ]:
```
NB15_3.head()
```

Out[ ]:

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

In [ ]:
```
NB15_4.head()
```

Out[ ]:

	59.166.0.9	7045	149.171.126.7	25	tcp	FIN	0.201886	37552	3380	31
0	59.166.0.9	9685	149.171.126.2	80	tcp	FIN	5.864748	19410	1087890	31
1	59.166.0.2	1421	149.171.126.4	53	udp	CON	0.001391	146	178	31
2	59.166.0.2	21553	149.171.126.2	25	tcp	FIN	0.053948	37812	3380	31
3	59.166.0.8	45212	149.171.126.4	53	udp	CON	0.000953	146	178	31
4	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	1	srcip	nominal	Source IP address
1	2	sport	integer	Source port number
2	3	dstip	nominal	Destination IP address
3	4	dsport	integer	Destination port number
4	5	proto	nominal	Transaction protocol
5	6	state	nominal	Indicates to the state and its dependent proto...
6	7	dur	Float	Record total duration
7	8	sbytes	Integer	Source to destination transaction bytes
8	9	dbytes	Integer	Destination to source transaction bytes
9	10	sttl	Integer	Source to destination time to live value
10	11	dttl	Integer	Destination to source time to live value
11	12	sloss	Integer	Source packets retransmitted or dropped
12	13	dloss	Integer	Destination packets retransmitted or dropped
13	14	service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and ...
14	15	Sload	Float	Source bits per second
15	16	Dload	Float	Destination bits per second
16	17	Spkts	integer	Source to destination packet count
17	18	Dpkts	integer	Destination to source packet count
18	19	swin	integer	Source TCP window advertisement value
19	20	dwin	integer	Destination TCP window advertisement value
20	21	stcpb	integer	Source TCP base sequence number
21	22	dtcpb	integer	Destination TCP base sequence number
22	23	smeansz	integer	Mean of the ?ow packet size transmitted by the...
23	24	dmeansz	integer	Mean of the ?ow packet size transmitted by the...
24	25	trans_depth	integer	Represents the pipelined depth into the connec...
25	26	res_bdy_len	integer	Actual uncompressed content size of the data t...
26	27	Sjit	Float	Source jitter (mSec)
27	28	Djit	Float	Destination jitter (mSec)
28	29	Stime	Timestamp	record start time
29	30	Ltime	Timestamp	record last time
30	31	Sintpkt	Float	Source interpacket arrival time (mSec)
31	32	Dintpkt	Float	Destination interpacket arrival time (mSec)
32	33	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

Concating 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
...		...	...	...	...	...	...	...	...
2540038		59.166.0.8	34415	149.171.126.0	5190	tcp	FIN	0.008119	1920
2540039		59.166.0.8	56352	149.171.126.2	53	udp	CON	0.001047	130
2540040		59.166.0.9	25527	149.171.126.4	6881	tcp	FIN	0.013106	1540
2540041		175.45.176.1	47439	149.171.126.14	53	udp	INT	0.000003	114
2540042		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
...		...	...	...	...	...	...	...	...	...	...	...
2540038		34415	5190	tcp	FIN	0.008119	1920	4312	31	29	6	6
2540039		56352	53	udp	CON	0.001047	130	162	31	29	0	0
2540040		25527	6881	tcp	FIN	0.013106	1540	1644	31	29	4	4
2540041		47439	53	udp	INT	0.000003	114	0	254	0	0	0
2540042		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** availabe 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   dttl              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
sport	0	0.00
dsport	0	0.00
proto	0	0.00
state	0	0.00
dur	0	0.00
sbytes	0	0.00
dbytes	0	0.00
sttl	0	0.00
dttl	0	0.00
sloss	0	0.00
dloss	0	0.00
service	0	0.00
Sload	0	0.00
Dload	0	0.00
Spkts	0	0.00
Dpkts	0	0.00
swin	0	0.00
dwin	0	0.00
stcpb	0	0.00
dtcpb	0	0.00
smeansz	0	0.00
dmeansz	0	0.00
trans_depth	0	0.00
res_bdy_len	0	0.00
Sjit	0	0.00
Djit	0	0.00
Stime	0	0.00
Ltime	0	0.00
Sintpkt	0	0.00
Dintpkt	0	0.00
tcprtt	0	0.00
synack	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[ ]:

Name	count	mean	std	min	25
dur	2048194.0	6.400000e-01	1.105000e+01	0.000000e+00	0.000000e+
sbytes	2048194.0	5.115540e+03	6.227923e+04	0.000000e+00	4.240000e+
dbytes	2048194.0	4.214292e+04	1.700893e+05	0.000000e+00	3.040000e+
sttl	2048194.0	4.252000e+01	4.834000e+01	0.000000e+00	3.100000e+
dttl	2048194.0	3.666000e+01	4.367000e+01	0.000000e+00	2.900000e+
sloss	2048194.0	6.150000e+00	2.463000e+01	0.000000e+00	0.000000e+
dloss	2048194.0	1.903000e+01	5.959000e+01	0.000000e+00	0.000000e+
Sload	2048194.0	8.335933e+06	7.207406e+07	0.000000e+00	8.036232e+
Dload	2048194.0	2.970461e+06	4.487254e+06	0.000000e+00	7.638325e+
Spkts	2048194.0	3.882000e+01	8.063000e+01	0.000000e+00	4.000000e+
Dpkts	2048194.0	5.004000e+01	1.273500e+02	0.000000e+00	4.000000e+
swin	2048194.0	1.803400e+02	1.160300e+02	0.000000e+00	0.000000e+
dwin	2048194.0	1.799700e+02	1.162000e+02	0.000000e+00	0.000000e+
stcpb	2048194.0	1.516590e+09	1.429920e+09	0.000000e+00	0.000000e+
dtcpb	2048194.0	1.516382e+09	1.429779e+09	0.000000e+00	0.000000e+
smeansz	2048194.0	1.301100e+02	1.649300e+02	0.000000e+00	6.200000e+
dmeansz	2048194.0	3.319400e+02	3.397700e+02	0.000000e+00	8.100000e+
trans_depth	2048194.0	1.000000e-01	3.800000e-01	0.000000e+00	0.000000e+
res_bdy_len	2048194.0	5.180000e+03	5.225101e+04	0.000000e+00	0.000000e+
Sjit	2048194.0	1.654010e+03	1.359472e+04	0.000000e+00	0.000000e+
Djit	2048194.0	8.457600e+02	3.258080e+03	0.000000e+00	3.300000e-
Stime	2048194.0	1.423098e+09	1.145972e+06	1.421927e+09	1.421949e+
Ltime	2048194.0	1.423098e+09	1.145972e+06	1.421927e+09	1.421949e+
Sintpkt	2048194.0	1.534600e+02	2.420990e+03	0.000000e+00	2.800000e-
Dintpkt	2048194.0	6.746000e+01	1.283740e+03	0.000000e+00	2.300000e-
tcprtt	2048194.0	1.000000e-02	5.000000e-02	0.000000e+00	0.000000e+
synack	2048194.0	0.000000e+00	3.000000e-02	0.000000e+00	0.000000e+
ackdat	2048194.0	0.000000e+00	3.000000e-02	0.000000e+00	0.000000e+
is_sm_ips_ports	2048194.0	0.000000e+00	3.000000e-02	0.000000e+00	0.000000e+
ct_state_ttl	2048194.0	9.000000e-02	4.200000e-01	0.000000e+00	0.000000e+
ct_flw_http_mthd	2048194.0	1.200000e-01	5.000000e-01	0.000000e+00	0.000000e+
is_ftp_login	2048194.0	2.000000e-02	1.400000e-01	0.000000e+00	0.000000e+

Name	count	mean	std	min	2!
ct_srv_src	2048194.0	5.410000e+00	5.190000e+00	1.000000e+00	2.000000e+
ct_srv_dst	2048194.0	5.160000e+00	5.010000e+00	1.000000e+00	2.000000e+
ct_dst_ltm	2048194.0	3.680000e+00	3.270000e+00	1.000000e+00	2.000000e+
ct_src_ltm	2048194.0	4.160000e+00	3.590000e+00	1.000000e+00	2.000000e+
ct_src_dport_ltm	2048194.0	1.620000e+00	2.660000e+00	1.000000e+00	1.000000e+
ct_dst_sport_ltm	2048194.0	1.250000e+00	1.750000e+00	1.000000e+00	1.000000e+
ct_dst_src_ltm	2048194.0	2.640000e+00	4.140000e+00	1.000000e+00	1.000000e+
Label	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[ ]:

Name	count	unique	top	freq
sport	2048194	100343	1043	18503
dsport	2048194	128297	53	233820
proto	2048194	135	tcp	1448579
state	2048194	16	FIN	1433230
service	2048194	13	-	1158268
ct_ftp_cmd	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' 'narp' 'ippc' 'vines' 'skip' 'rvd' 'emcon' 'ipv6-opts' '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'
 'sccopmce' 'il' 'igp' 'sat-expak' 'pgm' 'rdp' 'srp' 'iplt' 'trunk-1'
 'ggp' 'ax.25' 'netblt' 'ipcomp' 'crtp' 'smp' 'mfe-nsp' 'mux' 'l2tp'
 'irtp' '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  dttl 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 -> ['-' 'dn
s' '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
000000e-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).asty
    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
```

```
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).asty
    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).asty
    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'].uni
print(f"Unique values of 'dsport' after processing:", train_df['dsport'].u
```

```
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']).c
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', 'tcprtt', 'synack',
'ackdat', 'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_mthd', 'is_ftp_lo
gin', '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']).c
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', 'dm
eansz', 'trans_depth', 'res_bdy_len', 'Sjit', 'Djit', 'Stime', 'Ltime', 'Si
ntpkt', 'Dintpkt', 'tcprtt', 'synack', 'ackdat', 'is_sm_ips_ports', 'ct_sta
te_ttl', 'ct_flw_http_mthd', 'is_ftp_login', 'ct_ftp_cmd', 'ct_srv_src', 'c
t_srv_dst', 'ct_dst_ltm', 'ct_src_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_l
tm', 'ct_dst_src_ltm', 'Label']
Numerical columns in your notebook: ['sport', 'dsport', 'dur', 'sbytes', 'd
bytes', 'sttl', 'dttl', 'sloss', 'dloss', 'Sload', 'Dload', 'Spkts', 'Dpkt
s', '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=
# plt.tight_layout(rect=[0, 0, 1, 0.95])
# plt.show()
```

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

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

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

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].qu
#     upper_bound = train_df[col].quantile(0.75) + 1.5 * (train_df[col].qu
#     train_df[col] = train_df[col].apply(lambda x: median_value if x < lo
```

In [ ]:
```
train_df
```

Out[ ]:

	Name	sport	dsport	proto	state	dur	sbytes	dbytes	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 × 46 columns

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

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

Visualising Data Distribution and Skewness

In [ ]:
```
# # 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}', horizontalalignm

# # 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()
```

- |Skewness| < 1: The distribution is approximately symmetric.
- |Skewness| > 1: The distribution is highly skewed.
- |Skewness| between 1 and 2: Moderately skewed distribution.

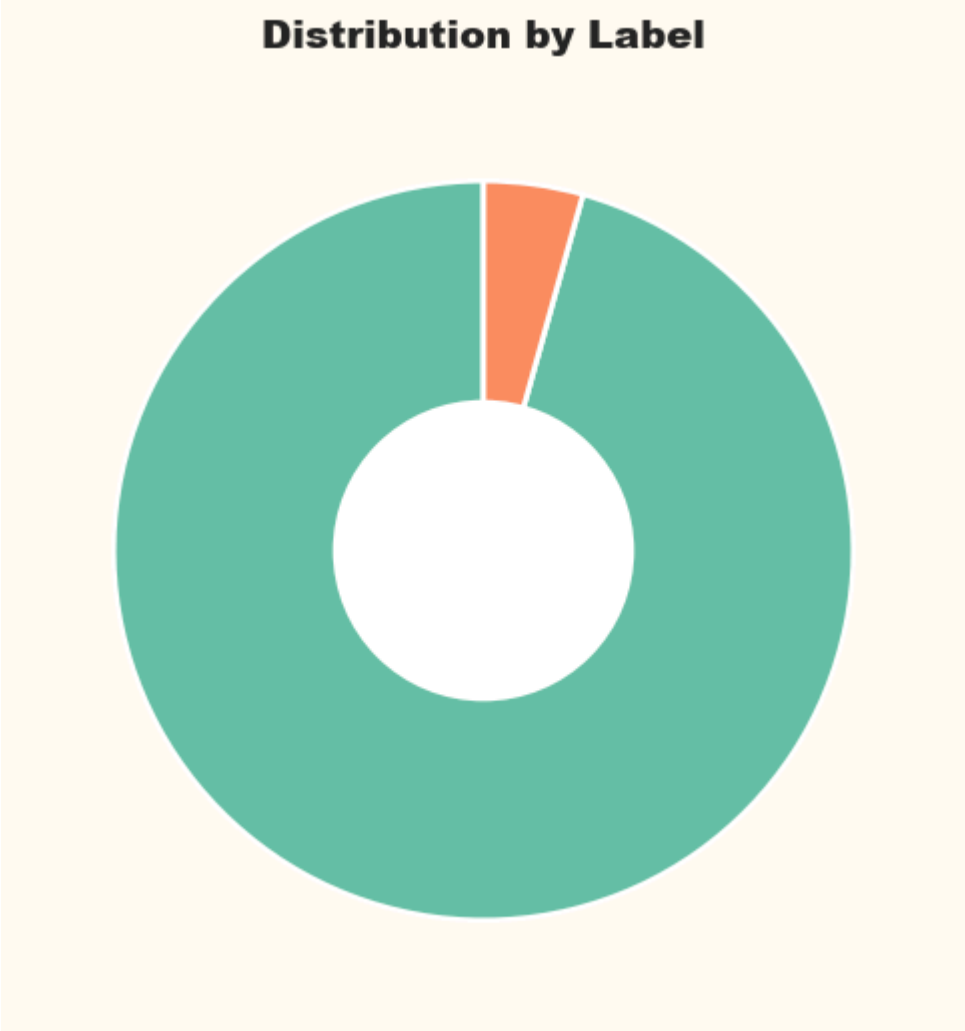
In [ ]:
```
def transform(x):
    eps = 1e-5
    numerical_columns = x.select_dtypes(include=['float64', 'int64']).colu

    # 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=col
    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')
```



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

Featutre Engineering

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

train_df.head()
```

Out[ ]:

	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['tcprtt'] / (df['synack'] + df['ackdat'] +

    # 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['tcprtt'] + 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	dbytes	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['servic

# Create the label mapping
proto_mapping = dict(zip(label_encoder_proto.classes_, label_encoder_proto
state_mapping = dict(zip(label_encoder_state.classes_, label_encoder_state
service_mapping = dict(zip(label_encoder_service.classes_, label_encoder_s

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, 'ar
p': 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': 1
7, '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, 'gmtp': 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, 'i
pip': 49, 'iplt': 50, 'ipnip': 51, 'ippc': 52, 'ipv6': 53, 'ipv6-frag': 54,
'ipv6-no': 55, 'ipv6-opts': 56, 'ipv6-route': 57, 'ipx-n-ip': 58, 'irtp': 5
9, 'isis': 60, 'iso-ip': 61, 'iso-tp4': 62, 'kryptolan': 63, 'l2tp': 64, 'l
arp': 65, 'leaf-1': 66, 'leaf-2': 67, 'merit-inp': 68, 'mfe-nsp': 69, 'mhr
p': 70, 'micp': 71, 'mobile': 72, 'mtp': 73, 'mux': 74, 'narp': 75, 'netbl
t': 76, 'nsfnet-igp': 77, 'nvp': 78, 'ospf': 79, 'pgm': 80, 'pim': 81, 'pip
e': 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': 9
4, 'sat-mon': 95, 'sccopmce': 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, 't
runk-1': 117, 'trunk-2': 118, 'ttp': 119, 'udp': 120, 'udt': 121, 'unas': 1
22, 'uti': 123, 'vines': 124, 'visa': 125, 'vmtp': 126, 'vrrp': 127, 'wb-ex
pak': 128, 'wb-mon': 129, 'wsn': 130, 'xnet': 131, 'xns-idp': 132, 'xtp': 1
33, 'zero': 134}
State Mapping:
{'ACC': 0, 'CLO': 1, 'CON': 2, 'ECO': 3, 'ECR': 4, 'FIN': 5, 'INT': 6, 'MA
S': 7, 'PAR': 8, 'REQ': 9, 'RST': 10, 'TST': 11, 'TXD': 12, 'URH': 13, 'UR
N': 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}
```

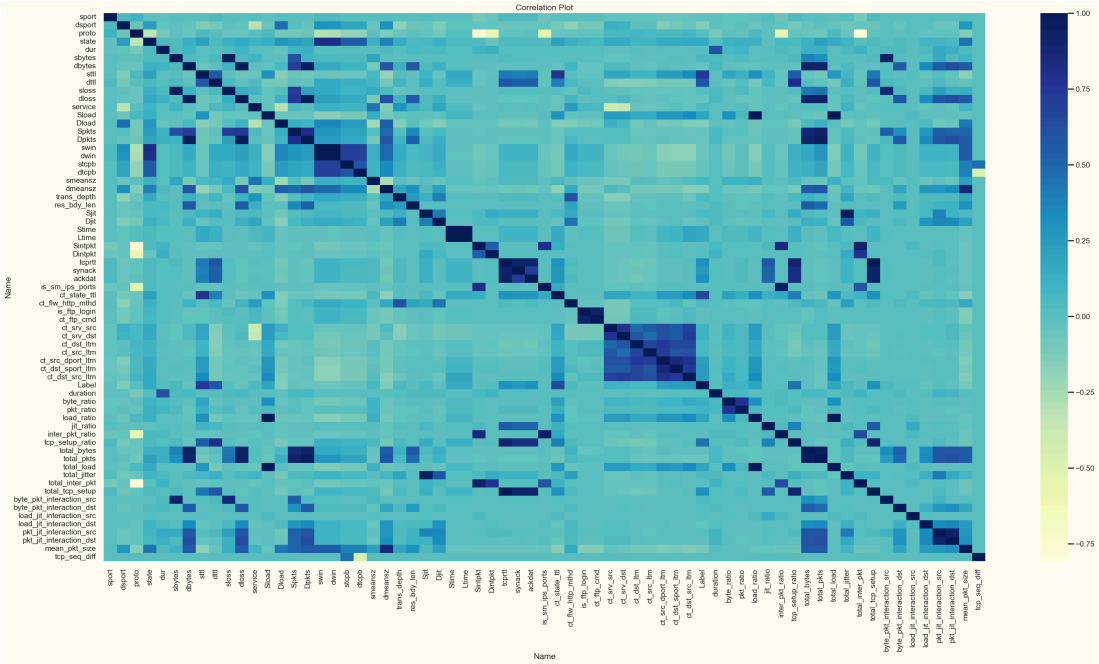
In [ ]:
```
#Checking the categorical columns
cat_columns = train_df.select_dtypes(include=['O']).columns.tolist()
cat_columns
```

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='Nam
e'>
```



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

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_feat
        # Choose the feature with higher correlation to the label for rete
        if abs(correlation_matrix.loc[feature1, 'Label']) > abs(correlatio
            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', 'dtt
l',
       '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_lt
m',
       'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'Label', 'duration', 'byte_rat
io',
       '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
f'],
      dtype='object', name='Name')
```

Splitting the features in dependent and independent features

Creating a new train_df and a test_df

In [ ]:
```
# Separate 8000 samples of Label = 1
label_1_samples = train_df[train_df['Label'] == 1].sample(n=8000, random_s

# Separate 8000 samples of Label = 0
label_0_samples = train_df[train_df['Label'] == 0].sample(n=8000, random_s

# Combine them into a new test_df
test_df = pd.concat([label_1_samples, label_0_samples])

# Remove the selected samples from train_df
train_df = train_df.drop(test_df.index)


# 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())
```

In [ ]:
```
test_df.shape
```

Out[ ]:
```
(16000, 44)
```

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

Out[ ]:
```
Label
1    8000
0    8000
Name: count, dtype: int64
```

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

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

Out[ ]:
```
(2032194, 44)
```

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

Out[ ]:
```
Label
0    1951348
1      80846
Name: count, dtype: int64
```

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

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

# Randomly sample 20% of these rows
train_ocsvm = label_0_df.sample(frac=0.15, random_state=42)
```

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

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

x.shape
```

Out[ ]:
```
(2032194, 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 = 80000

# oversample_strategy = {key: desired_count for key in y.value_counts().ind
# undersample_strategy = {key: desired_count for key in y.value_counts().i

# 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        1951348
1          80846
Name: count, dtype: int64

After resampling: Label
0        80000
1        80000
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 Infromation 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 importanc
    in predicting the target.

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

    Returns:
    DataFrame: A DataFrame containing features and their corresponding mut
```

```
            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_featu

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

# Display the scores
mi_scores
```

Out[ ]:

	Features	Scores
0	sttl	0.660585
1	ct_state_ttl	0.642202
2	dttl	0.630747
3	sbytes	0.594687
4	byte_ratio	0.589945
5	byte_pkt_interaction_dst	0.584336
6	pkt_ratio	0.502273
7	Dintpkt	0.465101
8	dmeansz	0.464084
9	Dload	0.447874
10	tcp_setup_ratio	0.424921
11	load_ratio	0.412068
12	dur	0.411287
13	smeansz	0.406340
14	Dpkts	0.399804
15	tcp_seq_diff	0.329653
16	Sjit	0.283850
17	dsport	0.248295
18	Djit	0.243206
19	load_jit_interaction_dst	0.205966
20	state	0.189691
21	jit_ratio	0.182617
22	Ltime	0.182082
23	ct_dst_ltm	0.175228
24	pkt_jit_interaction_dst	0.139060
25	ct_src_ltm	0.137435
26	ct_srv_dst	0.130972
27	load_jit_interaction_src	0.096858
28	ct_srv_src	0.094337
29	sport	0.079316
30	ct_dst_src_ltm	0.070679
31	ct_dst_sport_ltm	0.062865
32	res_bdy_len	0.059676

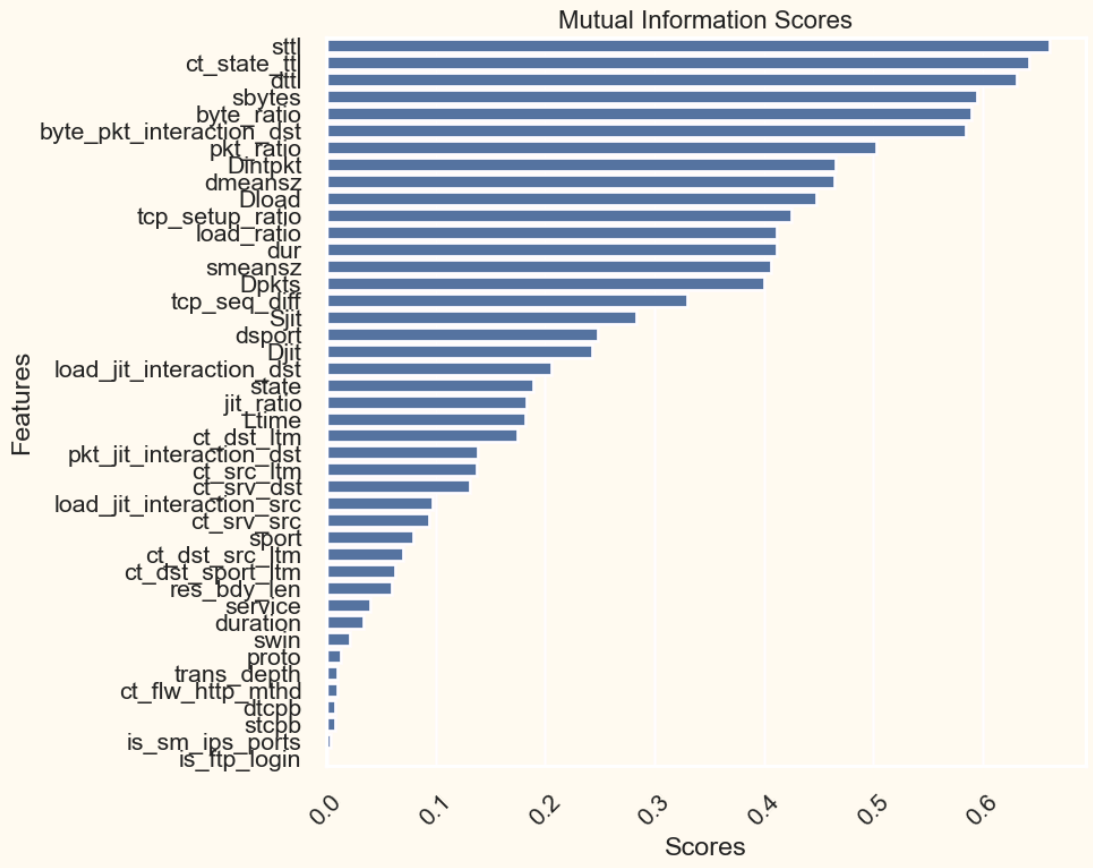
	Features	Scores
33	service	0.040197
34	duration	0.034099
35	swin	0.022032
36	proto	0.013007
37	trans_depth	0.010375
38	ct_flw_http_mthd	0.009993
39	dtcpb	0.008656
40	stcpb	0.007929
41	is_sm_ips_ports	0.004509
42	is_ftp_login	0.001028

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_sm_ips_ports',
 'is_ftp_login']
```

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 o
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', 'Dpkt
s',
       'smeansz', 'dmeansz', 'Sjit', 'Djit', 'Ltime', 'Dintpkt',
       'ct_state_ttl', 'ct_srv_dst', 'ct_dst_ltm', 'ct_src_ltm', 'byte_rati
o',
       '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 ran
    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	1.92
3	dur	0.01	1.98
4	sbytes	0.00	1.39
5	sttl	0.00	4.31
6	dttl	0.00	4.98
7	Dload	0.00	2.07
8	Dpkts	0.00	5.37
9	smeansz	0.00	1.33
10	dmeansz	0.00	3.38
11	Sjit	0.00	1.57
12	Djit	0.00	1.86
13	Ltime	0.00	1.19
14	Dintpkt	0.00	1.35
15	ct_state_ttl	0.00	4.46
16	ct_srv_dst	0.00	3.08
17	ct_dst_ltm	0.19	4.48
18	ct_src_ltm	0.00	3.48
19	byte_ratio	0.00	3.15
20	pkt_ratio	0.93	2.80
21	load_ratio	0.00	1.53
22	jit_ratio	0.00	1.07
23	tcp_setup_ratio	0.00	4.37
24	byte_pkt_interaction_dst	0.00	2.91
25	load_jit_interaction_dst	0.00	1.46
26	pkt_jit_interaction_dst	0.13	2.12
27	tcp_seq_diff	0.38	1.00

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

Out[ ]: (160000, 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_thr

    # 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_thres

        if not high_vif_features.empty:
            x.drop(columns=high_vif_features['feature'], axis=1, inplace=T
        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', 'dtt
l', 'Dload', 'Dpkts', 'smeansz', 'dmeansz', 'Sjit', 'Djit', 'Ltime', 'Dintp
kt', 'ct_state_ttl', 'ct_srv_dst', 'ct_src_ltm', 'byte_ratio', 'load_rati
o', 'jit_ratio', 'tcp_setup_ratio', 'byte_pkt_interaction_dst', 'load_jit_i
nteraction_dst']
```

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

In [ ]:
```
x
```

Out[ ]:

	dsport	state	dur	sbytes	sttl	dttl	Dload	Dpkts	smean
444161	14221	5	0.052849	4550	31	29	1.014150e+07	80	
648240	6881	5	0.017917	1540	31	29	6.934197e+05	18	
255725	39733	5	0.024878	3078	31	29	9.854168e+06	48	
969023	49565	5	0.076358	4550	31	29	7.624820e+06	80	
1766672	6881	5	0.094773	3618	31	29	8.627351e+06	96	
...	...	...	...	...	...	...	...	...	
1953432	111	5	0.762201	564	254	252	3.253735e+03	8	
1335675	1210	5	0.234814	1482	254	252	7.631572e+03	6	
923813	1723	5	1.436163	2516	254	252	1.726824e+03	8	
957480	54721	5	0.657109	364	62	252	7.122106e+03	8	
229598	53	6	0.000008	114	254	0	0.000000e+00	0	

160000 rows × 23 columns

Method 3: Recursive Feature Elimination (RFE)

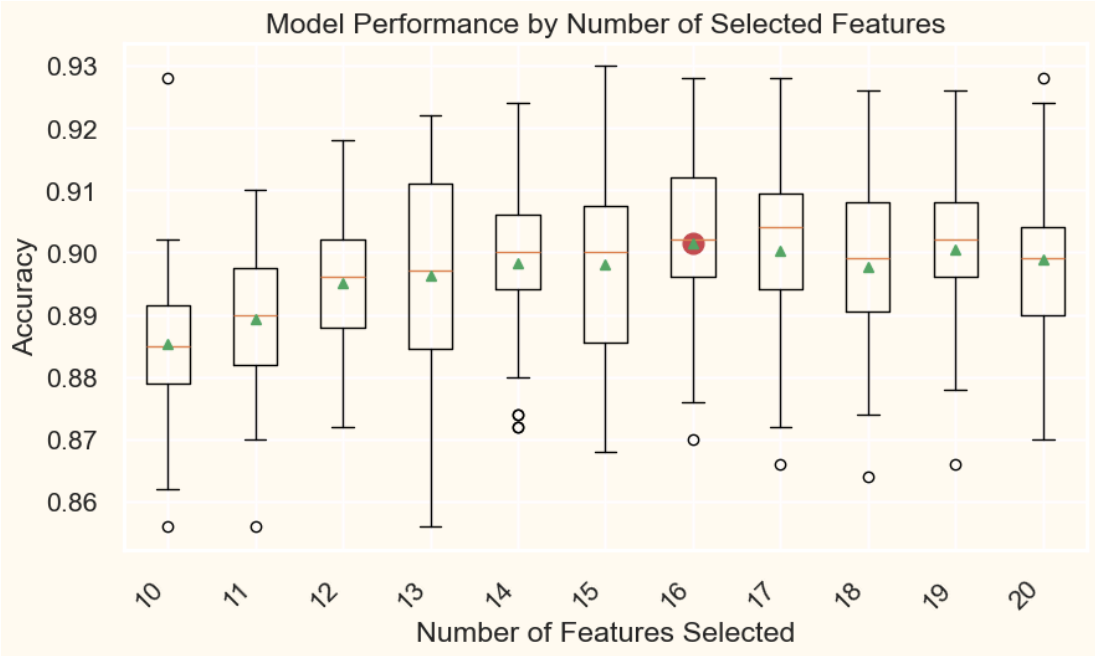
The example below demonstrates selecting different numbers of features from 2 to 10 on the synthetic binary classification dataset.

In [ ]:
```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from numpy import mean
from numpy import std
from sklearn.calibration import LinearSVC
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from matplotlib import pyplot
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression, Perceptron, SGDClassi
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClass
from sklearn.feature_selection import RFE
from sklearn.model_selection import cross_val_score, RepeatedStratifiedKFo
from numpy import mean, std
```

In [ ]:
```
# explore the number of selected features for RFE

# get the dataset
def get_dataset():
    X, Y = make_classification(n_samples=5000, n_features=20, n_informativ
    return X, Y
```

```
# get a list of models to evaluate
def get_models():
    models = dict()
    for i in range(10, 21):
        rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select
        model = DecisionTreeClassifier()
        models[str(i)] = Pipeline(steps=[('s',rfe),('m',model)])
    return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, Y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42
    scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_job
    return scores

# define dataset
X, Y = get_dataset()
# get the models to evaluate
models = get_models()

# Evaluate the models and store results
results, names = list(), list()
optimal_features = None
highest_mean_accuracy = 0

for name, model in models.items():
    scores = evaluate_model(model, X, Y)
    mean_score = mean(scores)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean_score, std(scores)))

    # Check if the current model has the highest mean accuracy
    if mean_score > highest_mean_accuracy:
        highest_mean_accuracy = mean_score
        optimal_features = int(name)

# Plot model performance for comparison
pyplot.figure(figsize=(10, 6))  # Increase figure size to fit labels
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.title('Model Performance by Number of Selected Features')
pyplot.xlabel('Number of Features Selected')
pyplot.ylabel('Accuracy')
pyplot.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
pyplot.subplots_adjust(bottom=0.2)  # Adjust the bottom margin to prevent
pyplot.grid(True)  # Adding grid lines for clarity

# Highlight the optimal number of features
optimal_index = names.index(str(optimal_features))
pyplot.plot(optimal_index + 1, highest_mean_accuracy, 'ro')  # 'ro' makes

pyplot.show()
```

```
>10 0.885 (0.013)
>11 0.889 (0.011)
>12 0.895 (0.012)
>13 0.896 (0.016)
>14 0.898 (0.013)
>15 0.898 (0.016)
>16 0.901 (0.015)
>17 0.900 (0.014)
>18 0.898 (0.014)
>19 0.900 (0.013)
>20 0.899 (0.014)
```



Identifying which model to use for RFE

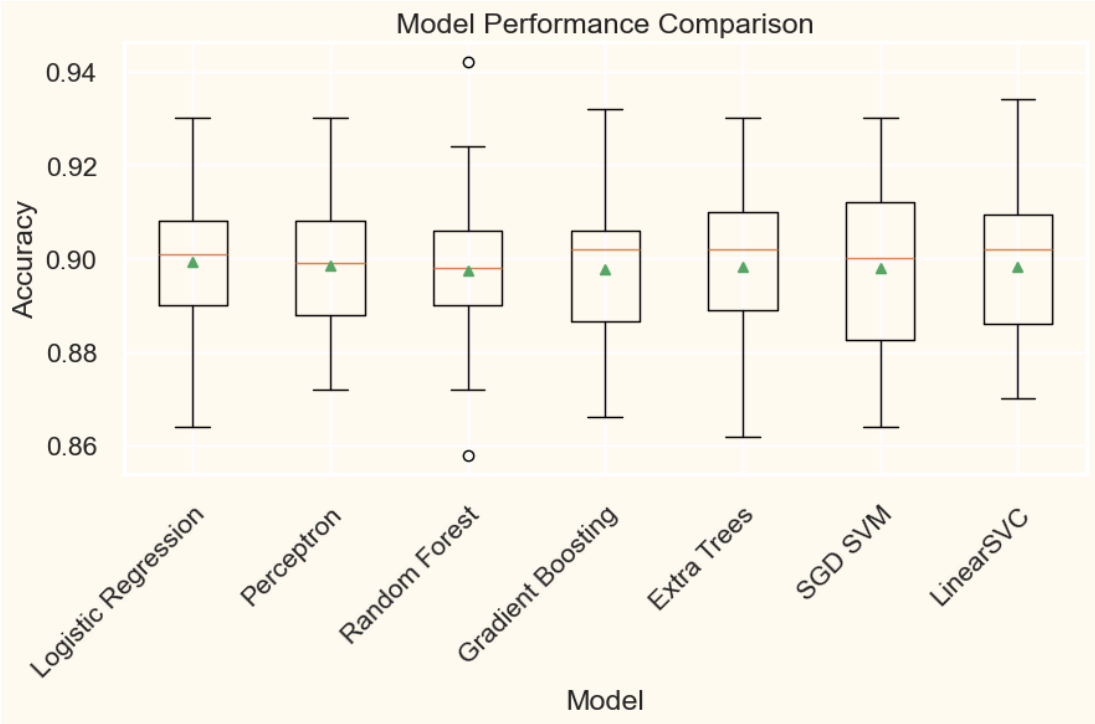
In [ ]:
```
optimal_features = 20
```

In [ ]:
```
# explore the algorithm wrapped by RFE


# get dataset
def get_dataset():
 X, Y = make_classification(n_samples=5000, n_features=20, n_informative=1
 return X, Y

# get a list of models to evaluate
def get_models(optimal_features):
    models = dict()
    # lr
    rfe = RFE(estimator=LogisticRegression(), n_features_to_select=optimal_
    models['Logistic Regression'] = Pipeline(steps=[('s',rfe),('m',Decisio
    # perceptron
    rfe = RFE(estimator=Perceptron(), n_features_to_select=optimal_feature
    models['Perceptron'] = Pipeline(steps=[('s',rfe),('m',DecisionTreeClas
    # rf
    rfe = RFE(estimator=RandomForestClassifier(), n_features_to_select=opt
    models['Random Forest'] = Pipeline(steps=[('s',rfe),('m',DecisionTreeC
    # gbm
    rfe = RFE(estimator=GradientBoostingClassifier(), n_features_to_select
    models['Gradient Boosting'] = Pipeline(steps=[('s',rfe),('m',DecisionT
    # Extra Trees
```

```
    rfe = RFE(estimator=ExtraTreesClassifier(), n_features_to_select=optim
    models['Extra Trees'] = Pipeline(steps=[('s',rfe),('m',DecisionTreeCla
    # SGD
    rfe = RFE(estimator=SGDClassifier(), n_features_to_select=optimal_feat
    models['SGD SVM'] = Pipeline(steps=[('s',rfe),('m',DecisionTreeClassif
    # Linear SVC
    rfe = RFE(estimator=LinearSVC(), n_features_to_select=optimal_features
    models['LinearSVC'] = Pipeline(steps=[('s',rfe),('m',DecisionTreeClass
    return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, Y):
 cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
 scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_jobs=5
 return scores

X, Y = get_dataset()
# get the models to evaluate
models = get_models(optimal_features)
# evaluate the models and store results
results, names = list(), list()

highest_mean_accuracy = 0
optimal_model_name = None

for name, model in models.items():
    scores = evaluate_model(model, X, Y)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
    # Check if the current model has the highest mean accuracy
    if mean_score > highest_mean_accuracy:
        highest_mean_accuracy = mean_score
        optimal_model_name = name

# Plot model performance for comparison
pyplot.figure(figsize=(10, 6))  # Increase figure size to fit labels
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.xticks(rotation=45, ha='right', wrap=True)  # Rotate and wrap x-axi
pyplot.subplots_adjust(bottom=0.3)  # Adjust the bottom margin to prevent

# Adding plot lines for better visualization
pyplot.plot()  # Optional: add a plot line if you want lines connecting me
pyplot.title('Model Performance Comparison')
pyplot.xlabel('Model')
pyplot.ylabel('Accuracy')
pyplot.grid(True)  # Adding grid lines for clarity
pyplot.show()

# Set the optimal_model based on the optimal_model_name
if optimal_model_name == 'Logistic Regression':
    optimal_model = LogisticRegression(max_iter=1000)
elif optimal_model_name == 'Perceptron':
    optimal_model = Perceptron(max_iter=1000)
elif optimal_model_name == 'Random Forest':
    optimal_model = RandomForestClassifier()
elif optimal_model_name == 'Gradient Boosting':
    optimal_model = GradientBoostingClassifier()
elif optimal_model_name == 'Extra Trees':
    optimal_model = ExtraTreesClassifier()
```

```
elif optimal_model_name == 'SGD SVM':
    optimal_model = SGDClassifier(max_iter=1000)
elif optimal_model_name == 'LinearSVC':
    optimal_model = LinearSVC(max_iter=1000)
```

```
>Logistic Regression 0.899 (0.014)
>Perceptron 0.898 (0.013)
>Random Forest 0.897 (0.016)
>Gradient Boosting 0.898 (0.014)
>Extra Trees 0.898 (0.016)
>SGD SVM 0.898 (0.017)
>LinearSVC 0.898 (0.016)
```



In [ ]:
```
# Initialize RFE with the model and select the top 'n' features
rfe = RFE(estimator=optimal_model, n_features_to_select=optimal_features)
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', 'Djit', 'Dintpkt',
'ct_state_ttl', 'ct_srv_dst', 'ct_src_ltm', 'byte_ratio', 'load_ratio', 'ji
t_ratio', 'tcp_setup_ratio']
```

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

Out[ ]:
```
(160000, 20)
```

In [ ]:
```
# Giving the final features of classifier x train, to the one class x trai
# Identify columns that are both in train_ocsvm and x, excluding "Label"
common_columns = [col for col in x.columns if col in train_ocsvm.columns]
```

```
# Include "Label" even if it's not in common_columns
if 'Label' in train_ocsvm.columns and 'Label' not in common_columns:
    common_columns.append('Label')

# Filter train_ocsvm to keep these columns
train_ocsvm = train_ocsvm[common_columns]
```

In [ ]:
```
train_ocsvm.shape
```

Out[ ]:
```
(292702, 21)
```

In [ ]:
```
train_ocsvm
```

Out[ ]:

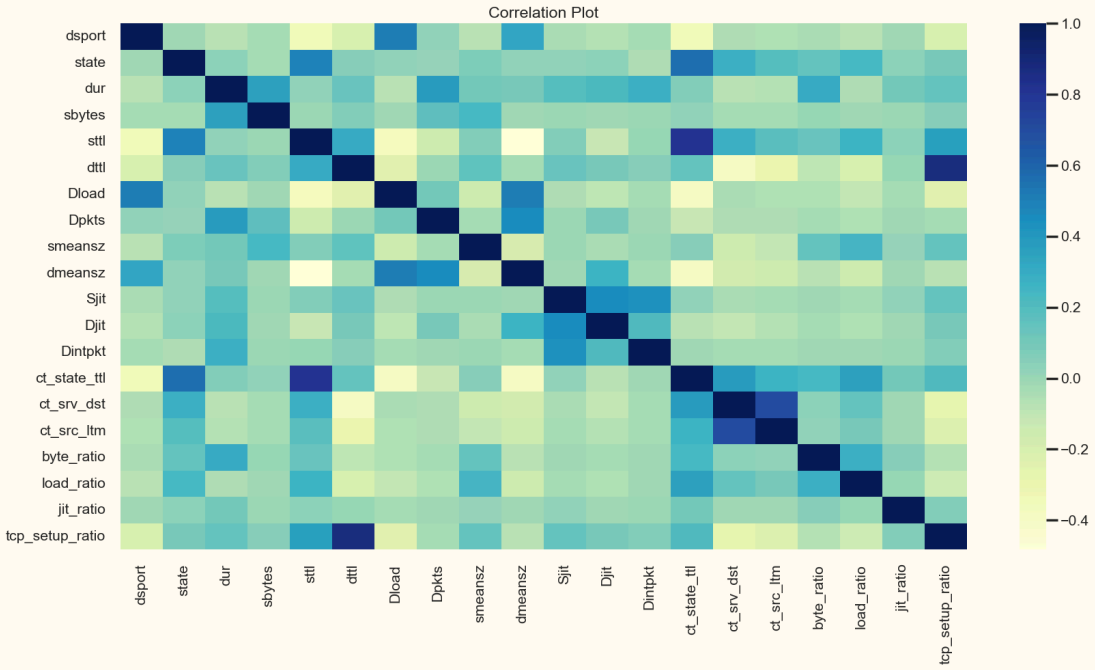
	Name	dsport	state	dur	sbytes	sttl	dttl	Dload	Dpkts	smeans
2305270	25	5	0.028857	37146	31	29	8.857469e+05	40	71
537588	53	2	0.000983	130	31	29	6.592065e+05	2	6
2488166	80	5	1.109621	820	62	252	6.914072e+03	8	8
922933	53	2	0.001074	146	31	29	6.629422e+05	2	7
1983731	53	2	0.000999	146	31	29	7.127127e+05	2	7
...	...	...	...	...	...	...	...	...	
1761664	21	5	0.024617	2934	31	29	1.193647e+06	54	5
2119139	53	2	0.001016	130	31	29	6.377952e+05	2	6
107755	5389	5	0.102718	3806	31	29	3.911602e+06	66	5
852246	15352	5	0.043966	3302	31	29	6.225538e+06	56	6
1362457	5190	5	0.008344	2048	31	29	2.325024e+06	28	8

292702 rows × 21 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:

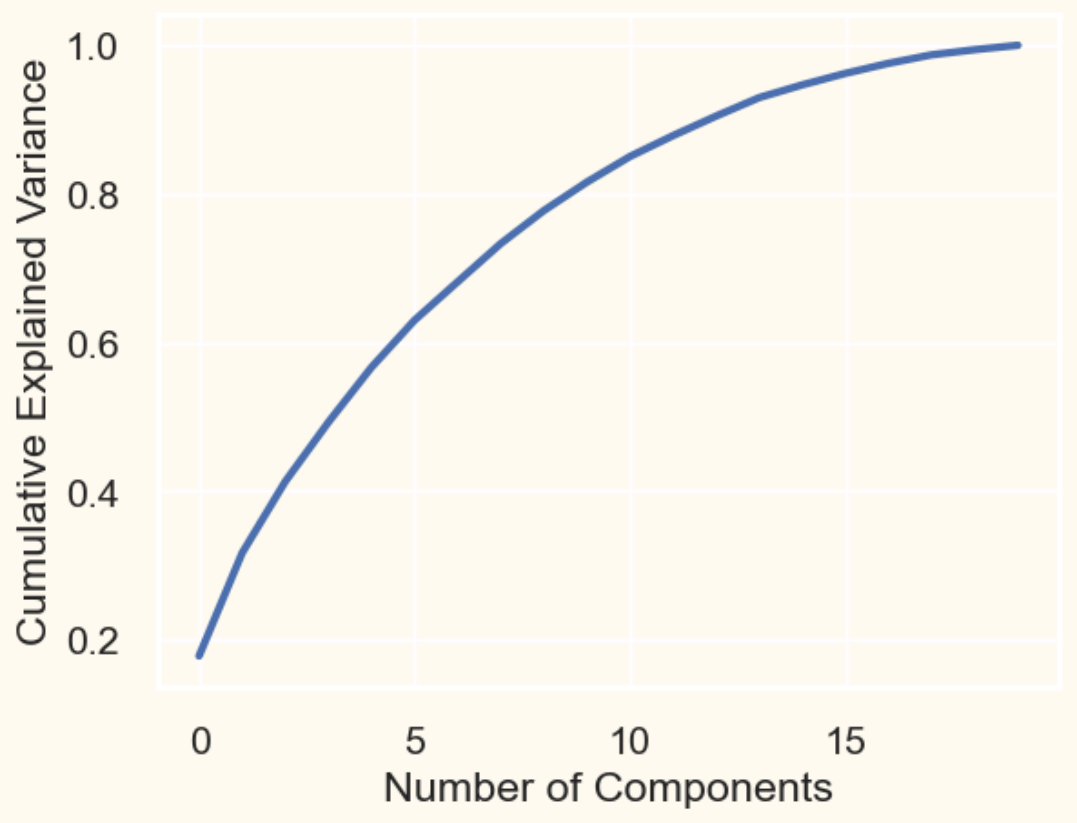
- No need to apply pca as all thw features are capturing full variance

Dividing in train-test split

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

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

# Save the scaler to a file
scaler_path = "scaler.joblib"
joblib.dump(scaler, scaler_path)
```

Out[ ]:
```
['scaler.joblib']
```

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: 128000
Number of columns: 20
1 and 0 distribution: Label
1        16126
0        15874
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, f1_score, precision_sc
import time
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.metrics import mean_absolute_error

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

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

    preds = model.predict(x_val_scaled)
    preds_proba = model.predict_proba(x_val_scaled)[:, 1]  # Use probabil
    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, prec

    print(f"Classification Report:\n{classification_report(y_val, preds)}

    cm = confusion_matrix(y_val, preds)
    print(f"Confusion Matrix:\n{cm}")

    # Plot confusion matrix
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
```

```
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

    # Calculate ROC curve
    fpr, tpr, thresholds = roc_curve(y_val, preds_proba)
    auc_score = roc_auc_score(y_val, preds_proba)

    # Plot ROC curve
    plt.figure()
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {auc
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()
```

5 | Model Building

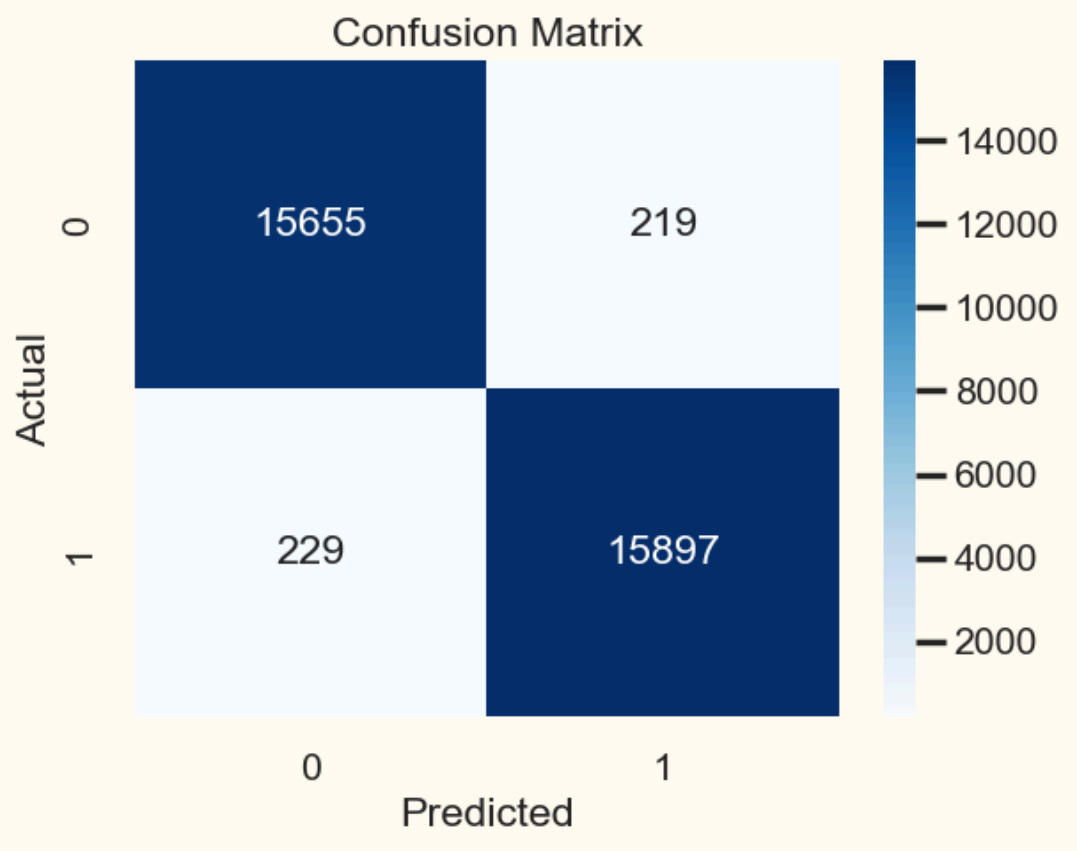
Decision Tree Model

Before we make our Random Forest, we'll see if we can signficantly 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_
    s = export_graphviz(t, out_file=None, feature_names=feature_names, fi
                        special_characters=True, precision=precision, **k
    return graphviz.Source(re.sub('Tree {', f'Tree {{ size={size}; ratio=
```

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

```
Accuracy: 98.60%
Recall: 98.58%
Precision: 98.64%
F1-Score: 98.61%
Mean Absolute Error: 1.40%
time to train: 1.04 s
time to predict: 0.00 s
total: 1.04 s
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     15874
           1       0.99      0.99      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000

Confusion Matrix:
[[15655   219]
 [  229 15897]]
```



```
---------------------------------------------------------------------------
-
FileNotFoundError                         Traceback (most recent call las
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.

FileNotFoundError: [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 las
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}'
```

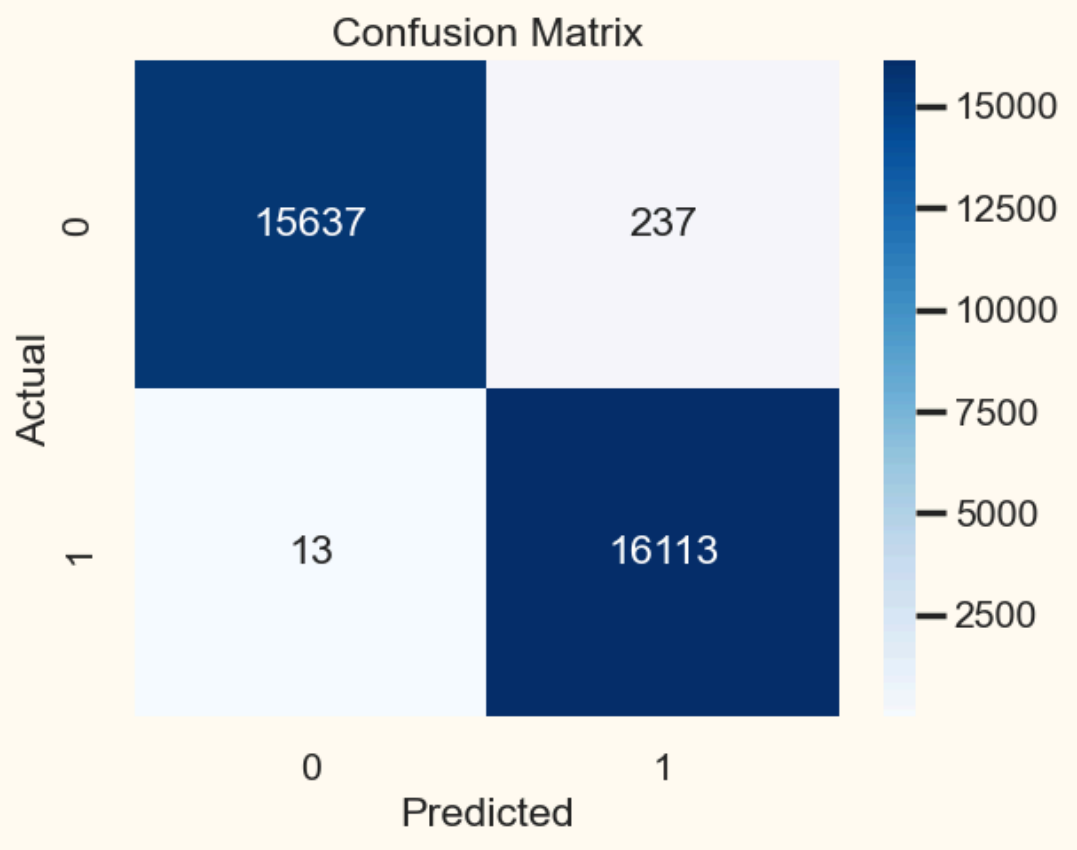
```
    167                   ' as keyword arg(s)',
    168                   stacklevel=stacklevel,
    169                   category=category)
--> 171 return func(*args, **kwargs)

File c:\DevTools\Python\Python312\Lib\site-packages\graphviz\piping.py:121, 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:149, 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, **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 0x1c9e3cf93d0>

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=42,
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: 99.22%
Recall: 99.92%
Precision: 98.55%
F1-Score: 99.23%
Mean Absolute Error: 0.78%
time to train: 4.40 s
time to predict: 0.05 s
total: 4.45 s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     15874
           1       0.99      1.00      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000

Confusion Matrix:
[[15637   237]
 [   13 16113]]
```

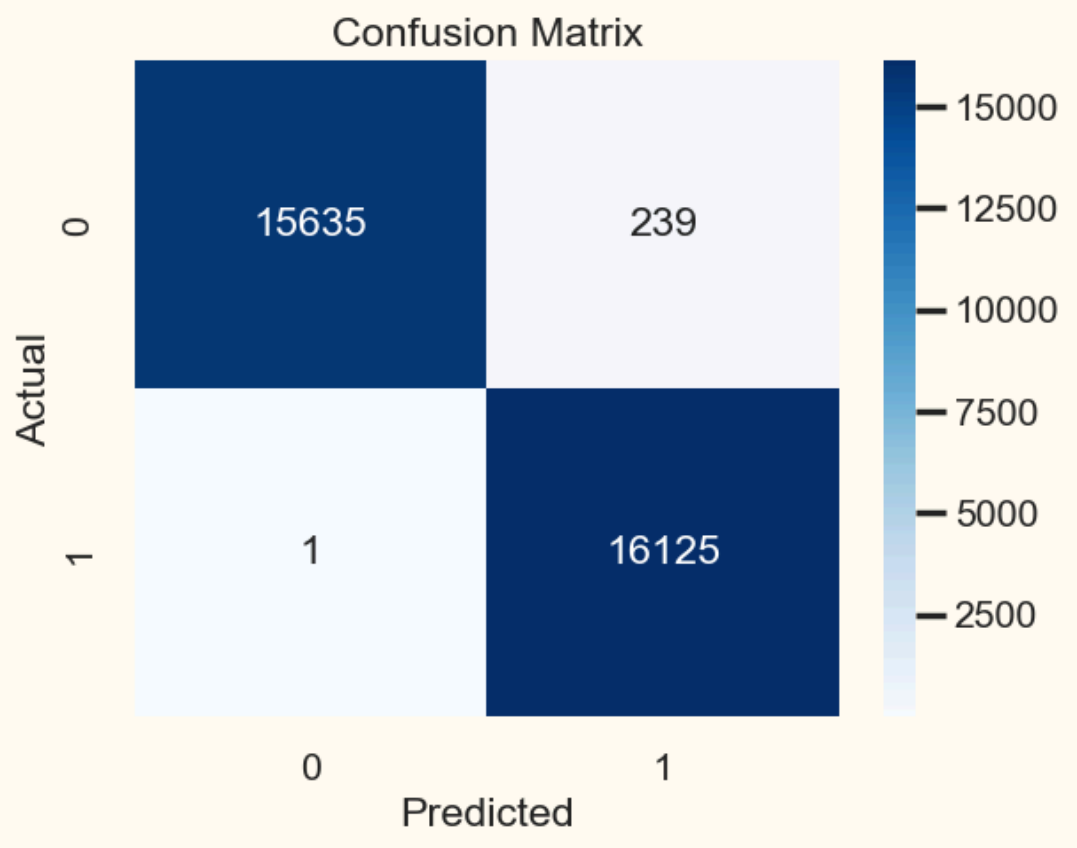


Out[ ]: ['model\\random_forest_model.joblib']

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: 99.25%
Recall: 99.99%
Precision: 98.54%
F1-Score: 99.26%
Mean Absolute Error: 0.75%
time to train: 38.85 s
time to predict: 0.04 s
total: 38.89 s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15874
           1       0.99      1.00      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000

Confusion Matrix:
[[15635   239]
 [    1 16125]]
```



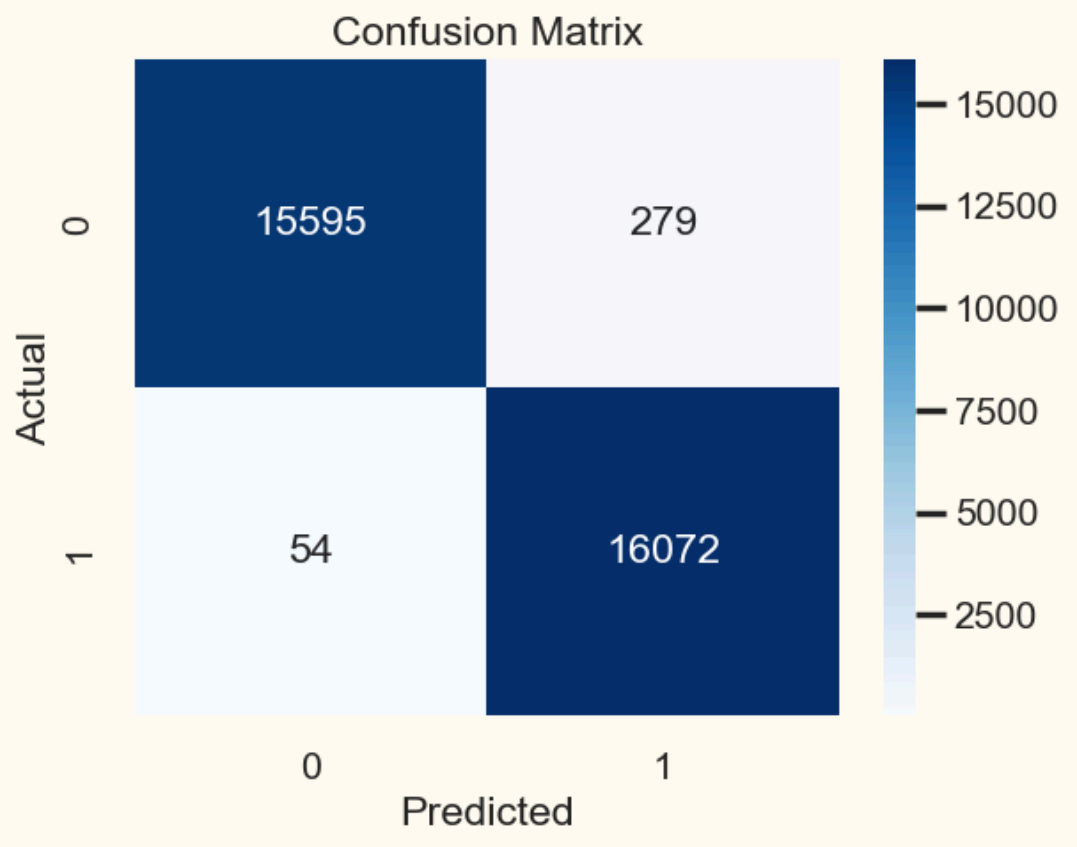
Out[ ]: ['model\\gradient_boosting_model.joblib']

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: 98.96%
Recall: 99.67%
Precision: 98.29%
F1-Score: 98.97%
Mean Absolute Error: 1.04%
time to train: 0.20 s
time to predict: 0.00 s
total: 0.20 s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15874
           1       0.98      1.00      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000

Confusion Matrix:
[[15595   279]
 [   54 16072]]
```

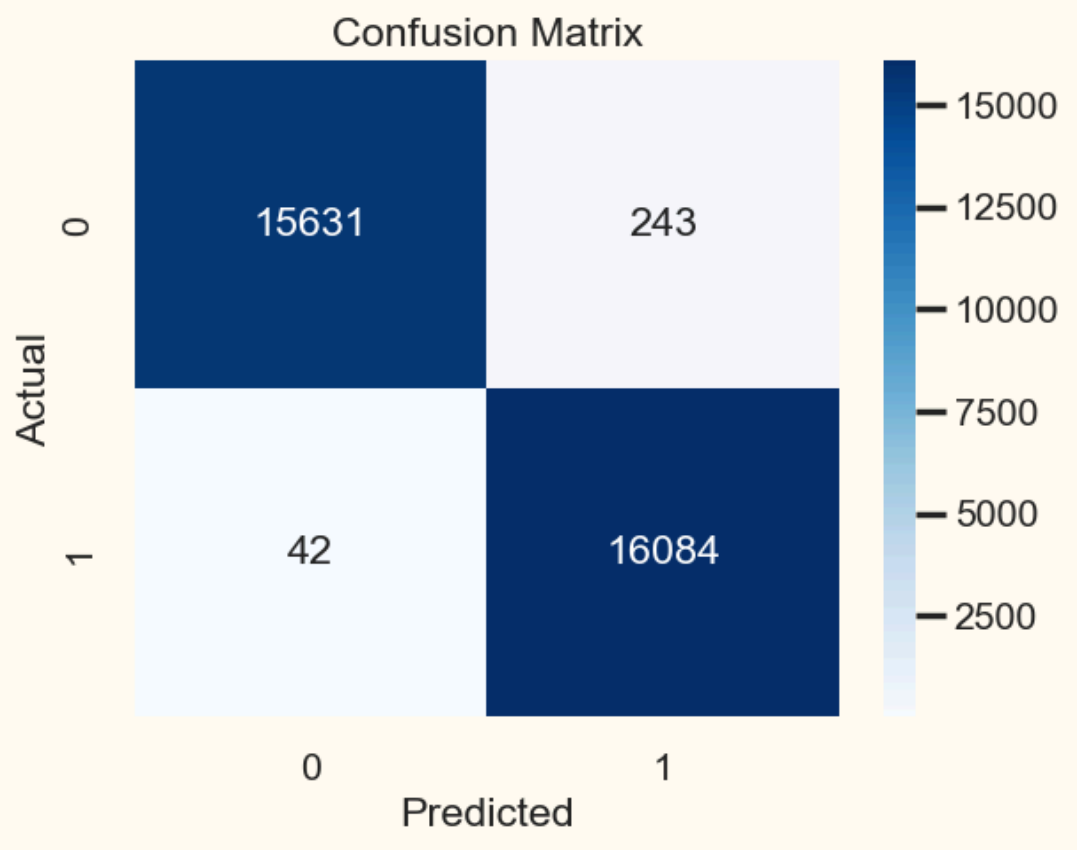


Out[ ]: ['model\\logistic_regression_model.joblib']

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: 99.11%
Recall: 99.74%
Precision: 98.51%
F1-Score: 99.12%
Mean Absolute Error: 0.89%
time to train: 0.02 s
time to predict: 5.51 s
total: 5.52 s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15874
           1       0.99      1.00      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000

Confusion Matrix:
[[15631   243]
 [   42 16084]]
```

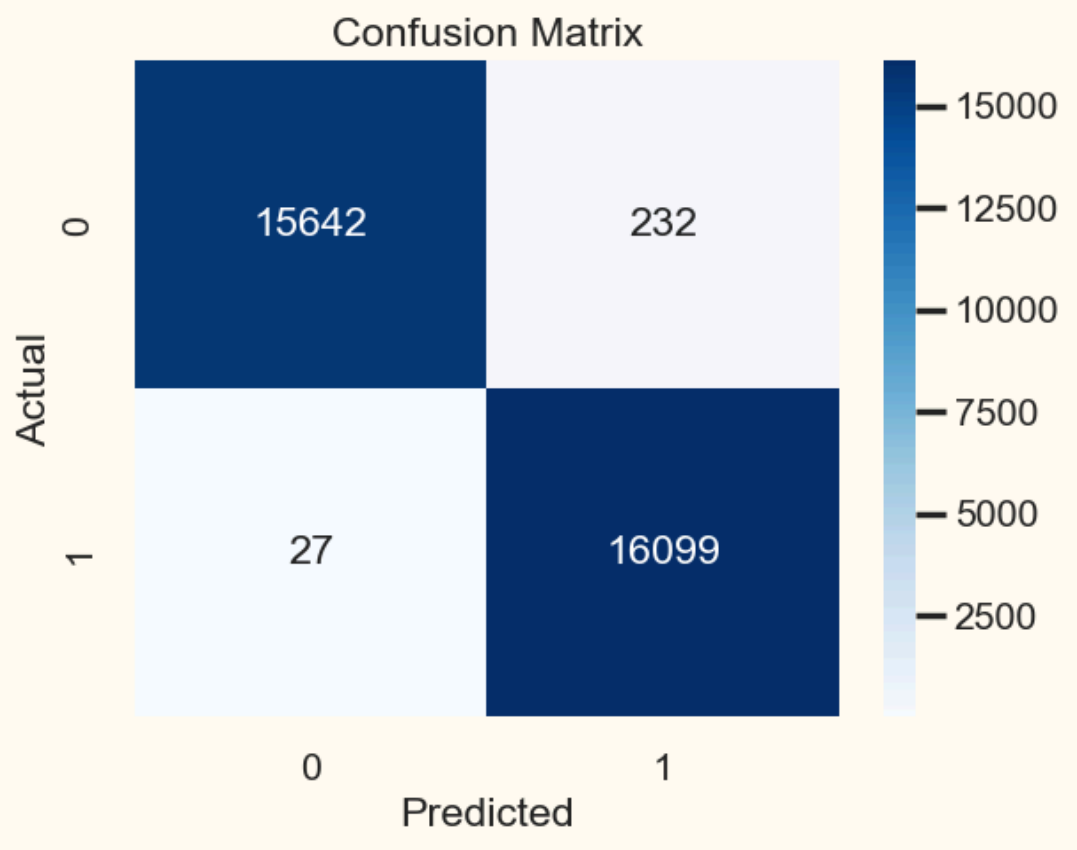


Out[ ]: ['model\\kneighbors_model.joblib']

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: 99.19%
Recall: 99.83%
Precision: 98.58%
F1-Score: 99.20%
Mean Absolute Error: 0.81%
time to train: 1.96 s
time to predict: 0.11 s
total: 2.07 s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     15874
           1       0.99      1.00      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000

Confusion Matrix:
[[15642   232]
 [   27 16099]]
```



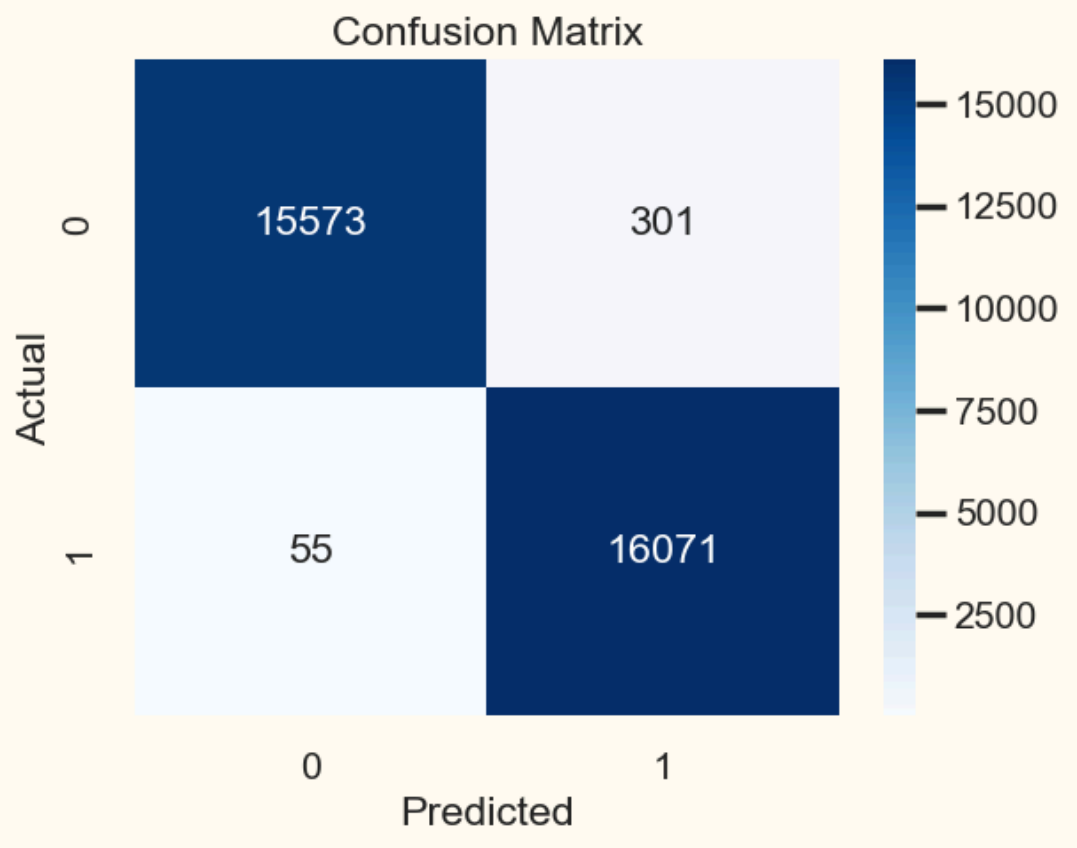
Out[ ]: ['model\\extra_trees_model.joblib']

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_v

# 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: 98.89%
Recall: 99.66%
Precision: 98.16%
F1-Score: 98.90%
Mean Absolute Error: 1.11%
time to train: 0.38 s
time to predict: 0.00 s
total: 0.38 s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15874
           1       0.98      1.00      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000

Confusion Matrix:
[[15573   301]
 [   55 16071]]
```

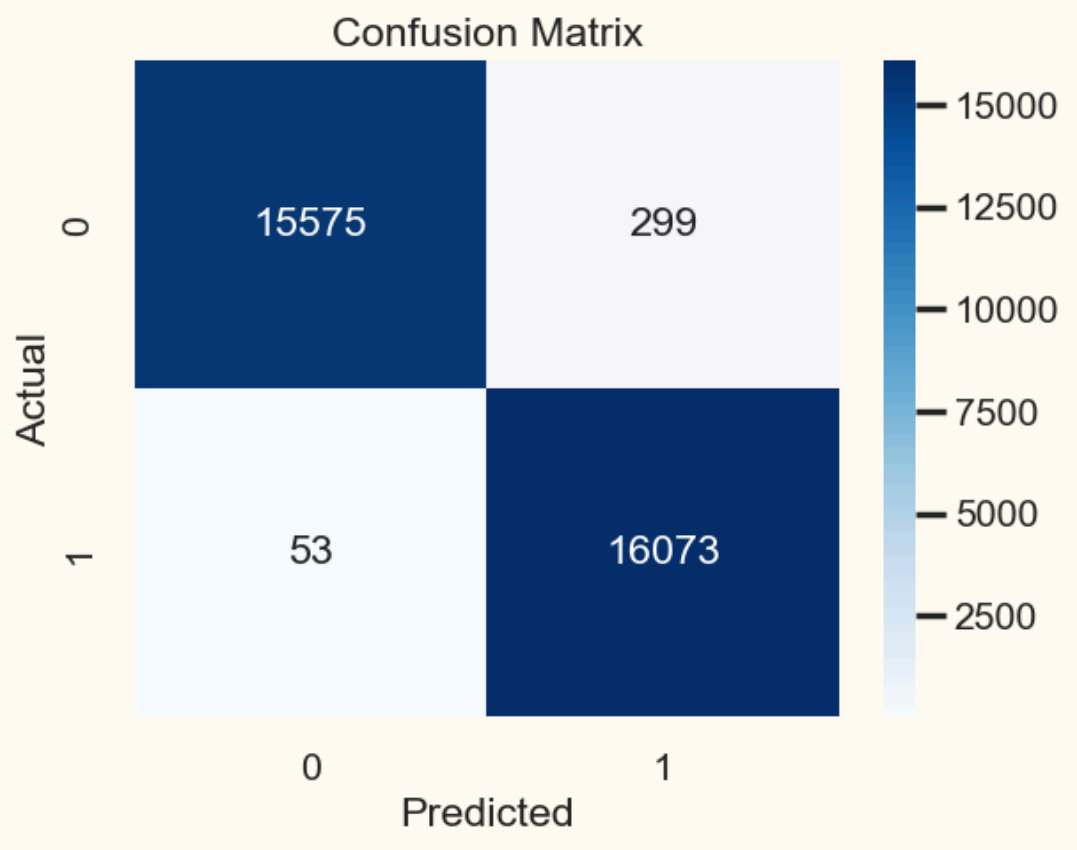


Out[ ]: ['model\\sgdc_svm_model.joblib']

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_va

# 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: 98.90%
Recall: 99.67%
Precision: 98.17%
F1-Score: 98.92%
Mean Absolute Error: 1.10%
time to train: 1.45 s
time to predict: 0.00 s
total: 1.45 s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15874
           1       0.98      1.00      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000

Confusion Matrix:
[[15575   299]
 [   53 16073]]
```



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 [ ]:
```
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
```

In [ ]:
```
train_ocsvm.shape
```

Out[ ]: (292702, 21)

In [ ]:
```
# Drop the 'Label' column
x_ocsvm = train_ocsvm.drop(['Label'], axis=1)
y_ocsvm = train_ocsvm[['Label']]

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

Out[ ]: (292702, 20)

In [ ]:
```
# ocx_train, ocx_test, ocy_train , ocy_val = train_test_split(x_ocsvm, y_
```

In [ ]:
```
# ocy_val.value_counts()
```

In [ ]:
```
# ocy_val.shape
```

In [ ]:
```
# ocx_test.shape
```

In [ ]:
```
# ocx_train.shape
```

In [ ]:
```
# # Standardize the features
# scaler = StandardScaler()
# ocx_train_scaled = scaler.fit_transform(ocx_train)
# ocx_test_scaled = scaler.transform(ocx_test)
```

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, ocx_train_scaled, ocx_test_scaled, ocy_va
    start = time.time()
    model.fit(ocx_train_scaled)
    end_train = time.time()

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

    preds = np.where(preds == -1, 1, 0)

    accuracy = accuracy_score(ocy_val, preds)
    recall = recall_score(ocy_val, preds, pos_label=0)
    precision = precision_score(ocy_val, preds, pos_label=0)
    f1s = f1_score(ocy_val, preds, pos_label=0)
    mae = mean_absolute_error(ocy_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 for Label=0:\n{classification_report(oc

    cm = confusion_matrix(ocy_val, preds)
    # Display only the part of the matrix corresponding to actual data (0
    cm_display = cm[0:1, 0:2]  # Take only the first row, all columns

    print(f"Confusion Matrix:\n{cm_display}")  # Print the confusion matr

    sns.heatmap(cm_display, annot=True, fmt='d', cmap='Blues', xticklabel
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
```

```
    plt.show()

    return accuracy
```

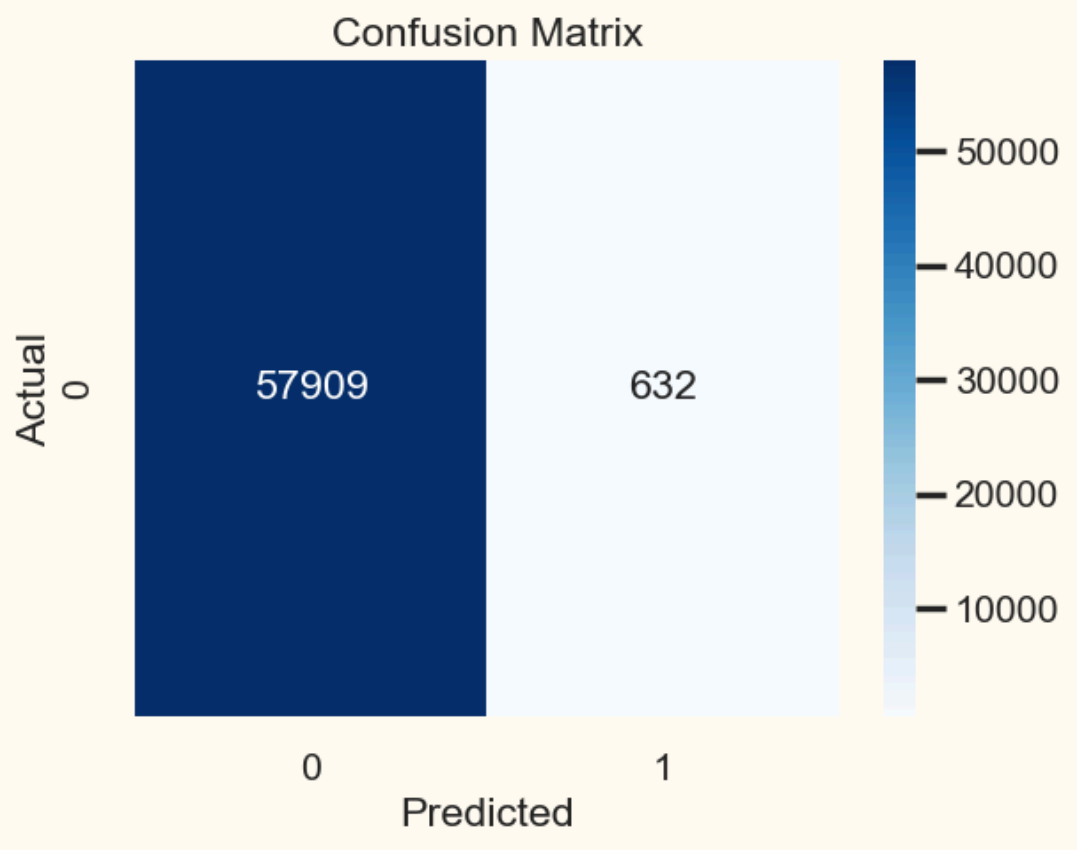
Using Bayesian Search to determine the best parameters for the One Class SVM model

In [ ]:
```
# # 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 [ ]:
```
# Loop until accuracy is greater than or equal to 65%
random_state = 1
accuracy_threshold = 0.95
```

```
accuracy = 0

while accuracy < accuracy_threshold:
    print(f"\nRunning with random_state = {random_state}")

    # Split the data with the current random_state
    ocx_train, ocx_test, ocy_train, ocy_val = train_test_split(x_ocsvm, y

    # Scale the data
    scaler = StandardScaler()
    ocx_train_scaled = scaler.fit_transform(ocx_train)
    ocx_test_scaled = scaler.transform(ocx_test)

    # Define the One-Class SVM model
    oc_svm = OneClassSVM(
        nu=0.01,
        gamma='auto',
        kernel='linear'
    )

    # Train and evaluate One-Class SVM
    accuracy = train_evaluate_ocsvm(oc_svm, ocx_train_scaled, ocx_test_sc

    # If accuracy is below threshold, increase random_state for next iter
    if accuracy < accuracy_threshold:
        print(f"Accuracy ({accuracy:.2%}) is below threshold. Changing ra
        random_state += 1
    else:
        print(f"Achieved accuracy ({accuracy:.2%}) meets the threshold.")
```

```
Running with random_state = 1
Accuracy: 98.92%
Recall: 0.00%
Precision: 0.00%
F1-Score: 0.00%
Mean Absolute Error: 1.08%
time to train: 11.39 s
time to predict: 1.97 s
total: 13.35 s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     58541
           1       0.00      0.00      0.00         0

    accuracy                           0.99     58541
   macro avg       0.50      0.49      0.50     58541
weighted avg       1.00      0.99      0.99     58541

Confusion Matrix:
[[57909   632]]
```




```
Achieved accuracy (98.92%) meets the threshold.
```

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, ocx_train_scaled , ocx_test_scaled , ocy_v

# 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')
```

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

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
```

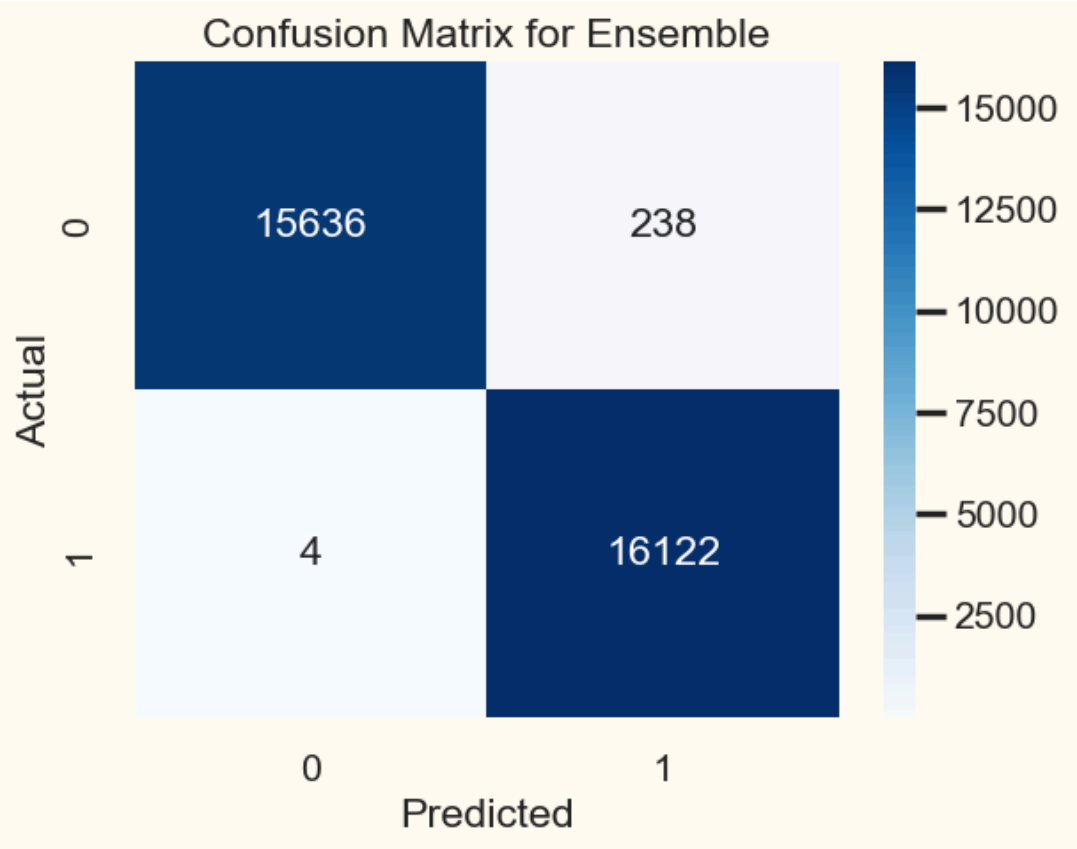
```
Accuracy: 99.24%
Recall: 99.24%
Precision: 99.25%
F1-Score: 99.24%
MAE: 0.76%
time to train: 48.03 s
time to predict: 5.82 s
total: 53.85 s
```

In [ ]:

```
# Display classification report and confusion matrix
print(f"Classification Report:\n{classification_report(y_val, ensemble_pr
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     15874
           1       0.99      1.00      0.99     16126

    accuracy                           0.99     32000
   macro avg       0.99      0.99      0.99     32000
weighted avg       0.99      0.99      0.99     32000
```



-- 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({'Acc
                                                                     'Pre
                                                                     'Rec
```

```
                                                                'F1-
                                                                'MAE
                                                                'tim
                                                                'tim
                                                                'tot
                                                                })
```

Out[ ]:

	Accuracy	Recall	Precision	F1-Score	Mean Absolute Error	time to train
DecisionTreeClassifier	98.60%	98.58%	98.64%	98.61%	0.014000	1.0
RandomForestClassifier	99.22%	99.92%	98.55%	99.23%	0.007812	4.4
GradientBoostingClassifier	99.25%	99.99%	98.54%	99.26%	0.007500	38.9
LogisticRegression	98.96%	99.67%	98.29%	98.97%	0.010406	0.2
KNeighborsClassifier	99.11%	99.74%	98.51%	99.12%	0.008906	0.0
ExtraTreesClassifier	99.19%	99.83%	98.58%	99.20%	0.008094	2.0
SGDClassifier	98.89%	99.66%	98.16%	98.90%	0.011125	0.4
LinearSVC	98.90%	99.67%	98.17%	98.92%	0.011000	1.5
SVM model (OneClassSVM)	98.92%	0.00%	0.00%	0.00%	0.010796	11.4
Ensemble (without OC)	99.24%	99.24%	99.25%	99.24%	0.007562	48.0

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(),
print(f"Decision Tree cross-validated accuracy: {dt_scores.mean():.2%}")

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

gbm_scores = cross_val_score(gbm, x_resampled, y_resampled.values.ravel()
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():.

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.valu
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
print(f"SVC SVM cross-validated accuracy: {svc_svm_scores.mean():.2%}")
```

```
Decision Tree cross-validated accuracy: 98.58%
Random Forest cross-validated accuracy: 99.21%
Gradient Boosting cross-validated accuracy: 99.22%
Logistic Regression cross-validated accuracy: 83.83%
kNeighbours cross-validated accuracy: 95.45%
Extra Trees cross-validated accuracy: 99.21%
SGD SVM cross-validated accuracy: 74.98%
SVC SVM cross-validated accuracy: 83.86%
```

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=optimal_features, activation='re
    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=200, verbose=2)
end_train = time.time()
```

```
Epoch 1/200
640/640 - 2s - 3ms/step - accuracy: 0.9756 - f1_m: 0.9737 - loss: 0.1156 -
mae_m: 0.0731 - precision_m: 0.9703 - recall_m: 0.9784
Epoch 2/200
640/640 - 1s - 911us/step - accuracy: 0.9909 - f1_m: 0.9909 - loss: 0.0443
- mae_m: 0.0192 - precision_m: 0.9824 - recall_m: 0.9997
Epoch 3/200
640/640 - 1s - 886us/step - accuracy: 0.9913 - f1_m: 0.9913 - loss: 0.0396
- mae_m: 0.0174 - precision_m: 0.9832 - recall_m: 0.9996
Epoch 4/200
640/640 - 1s - 916us/step - accuracy: 0.9917 - f1_m: 0.9916 - loss: 0.0366
- mae_m: 0.0164 - precision_m: 0.9840 - recall_m: 0.9995
Epoch 5/200
640/640 - 1s - 939us/step - accuracy: 0.9919 - f1_m: 0.9919 - loss: 0.0355
- mae_m: 0.0160 - precision_m: 0.9845 - recall_m: 0.9995
Epoch 6/200
640/640 - 1s - 916us/step - accuracy: 0.9919 - f1_m: 0.9919 - loss: 0.0350
- mae_m: 0.0156 - precision_m: 0.9846 - recall_m: 0.9995
Epoch 7/200
640/640 - 1s - 934us/step - accuracy: 0.9920 - f1_m: 0.9921 - loss: 0.0343
- mae_m: 0.0153 - precision_m: 0.9848 - recall_m: 0.9995
Epoch 8/200
640/640 - 1s - 921us/step - accuracy: 0.9920 - f1_m: 0.9920 - loss: 0.0340
- mae_m: 0.0153 - precision_m: 0.9848 - recall_m: 0.9994
Epoch 9/200
640/640 - 1s - 915us/step - accuracy: 0.9921 - f1_m: 0.9921 - loss: 0.0338
- mae_m: 0.0151 - precision_m: 0.9848 - recall_m: 0.9996
Epoch 10/200
640/640 - 1s - 916us/step - accuracy: 0.9921 - f1_m: 0.9920 - loss: 0.0336
- mae_m: 0.0151 - precision_m: 0.9847 - recall_m: 0.9996
Epoch 11/200
640/640 - 1s - 915us/step - accuracy: 0.9920 - f1_m: 0.9920 - loss: 0.0334
- mae_m: 0.0151 - precision_m: 0.9848 - recall_m: 0.9995
Epoch 12/200
640/640 - 1s - 916us/step - accuracy: 0.9921 - f1_m: 0.9920 - loss: 0.0334
- mae_m: 0.0149 - precision_m: 0.9847 - recall_m: 0.9996
Epoch 13/200
640/640 - 1s - 906us/step - accuracy: 0.9921 - f1_m: 0.9921 - loss: 0.0330
- mae_m: 0.0150 - precision_m: 0.9848 - recall_m: 0.9996
Epoch 14/200
640/640 - 1s - 916us/step - accuracy: 0.9921 - f1_m: 0.9921 - loss: 0.0329
- mae_m: 0.0149 - precision_m: 0.9848 - recall_m: 0.9997
Epoch 15/200
640/640 - 1s - 940us/step - accuracy: 0.9921 - f1_m: 0.9922 - loss: 0.0328
- mae_m: 0.0149 - precision_m: 0.9848 - recall_m: 0.9997
Epoch 16/200
640/640 - 1s - 916us/step - accuracy: 0.9921 - f1_m: 0.9922 - loss: 0.0327
- mae_m: 0.0148 - precision_m: 0.9849 - recall_m: 0.9996
Epoch 17/200
640/640 - 1s - 916us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0328
- mae_m: 0.0148 - precision_m: 0.9849 - recall_m: 0.9997
Epoch 18/200
640/640 - 1s - 1ms/step - accuracy: 0.9921 - f1_m: 0.9921 - loss: 0.0326 -
mae_m: 0.0148 - precision_m: 0.9848 - recall_m: 0.9997
Epoch 19/200
640/640 - 1s - 915us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0324
- mae_m: 0.0149 - precision_m: 0.9848 - recall_m: 0.9998
Epoch 20/200
640/640 - 1s - 915us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0324
- mae_m: 0.0147 - precision_m: 0.9848 - recall_m: 0.9997
```

```
Epoch 21/200
640/640 - 1s - 916us/step - accuracy: 0.9922 - f1_m: 0.9921 - loss: 0.0323
- mae_m: 0.0147 - precision_m: 0.9847 - recall_m: 0.9997
Epoch 22/200
640/640 - 1s - 965us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0322
- mae_m: 0.0148 - precision_m: 0.9849 - recall_m: 0.9998
Epoch 23/200
640/640 - 1s - 964us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0321
- mae_m: 0.0147 - precision_m: 0.9848 - recall_m: 0.9998
Epoch 24/200
640/640 - 1s - 940us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0320
- mae_m: 0.0147 - precision_m: 0.9849 - recall_m: 0.9998
Epoch 25/200
640/640 - 1s - 964us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0319
- mae_m: 0.0147 - precision_m: 0.9848 - recall_m: 0.9998
Epoch 26/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0320
- mae_m: 0.0146 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 27/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9922 - loss: 0.0318
- mae_m: 0.0147 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 28/200
640/640 - 1s - 915us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0317
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 29/200
640/640 - 1s - 940us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0318
- mae_m: 0.0147 - precision_m: 0.9848 - recall_m: 0.9998
Epoch 30/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9922 - loss: 0.0317
- mae_m: 0.0146 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 31/200
640/640 - 1s - 915us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0319
- mae_m: 0.0146 - precision_m: 0.9848 - recall_m: 0.9998
Epoch 32/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0316
- mae_m: 0.0146 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 33/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0316
- mae_m: 0.0146 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 34/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0316
- mae_m: 0.0146 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 35/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0316
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 36/200
640/640 - 1s - 916us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0315
- mae_m: 0.0145 - precision_m: 0.9848 - recall_m: 0.9998
Epoch 37/200
640/640 - 1s - 915us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0315
- mae_m: 0.0146 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 38/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0315
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 39/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0314
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 40/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0314
- mae_m: 0.0146 - precision_m: 0.9848 - recall_m: 0.9999
```

```
Epoch 41/200
640/640 - 1s - 915us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0314
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 42/200
640/640 - 1s - 915us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0315
- mae_m: 0.0146 - precision_m: 0.9847 - recall_m: 0.9998
Epoch 43/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0312
- mae_m: 0.0145 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 44/200
640/640 - 1s - 906us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0313
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 45/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0312
- mae_m: 0.0145 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 46/200
640/640 - 1s - 915us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0312
- mae_m: 0.0145 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 47/200
640/640 - 1s - 931us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0311
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 48/200
640/640 - 1s - 900us/step - accuracy: 0.9923 - f1_m: 0.9922 - loss: 0.0312
- mae_m: 0.0145 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 49/200
640/640 - 1s - 891us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0312
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 50/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0311
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 51/200
640/640 - 1s - 915us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0312
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 52/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0311
- mae_m: 0.0144 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 53/200
640/640 - 1s - 905us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0310
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 54/200
640/640 - 1s - 905us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0310
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 55/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0311
- mae_m: 0.0145 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 56/200
640/640 - 1s - 928us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0310
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 57/200
640/640 - 1s - 927us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0309
- mae_m: 0.0144 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 58/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0308
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 59/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0309
- mae_m: 0.0145 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 60/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0310
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
```

```
Epoch 61/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0309
- mae_m: 0.0145 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 62/200
640/640 - 1s - 915us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0307
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 63/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0307
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 64/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0308
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 65/200
640/640 - 1s - 915us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0308
- mae_m: 0.0145 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 66/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0306
- mae_m: 0.0144 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 67/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0306
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 68/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9922 - loss: 0.0306
- mae_m: 0.0144 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 69/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0306
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 70/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0306
- mae_m: 0.0144 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 71/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0306
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 72/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0306
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 73/200
640/640 - 1s - 916us/step - accuracy: 0.9923 - f1_m: 0.9922 - loss: 0.0305
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 74/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0305
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 75/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0305
- mae_m: 0.0144 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 76/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0306 -
mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 77/200
640/640 - 1s - 974us/step - accuracy: 0.9923 - f1_m: 0.9922 - loss: 0.0305
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 78/200
640/640 - 1s - 945us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0305
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 79/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0305
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 80/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0304
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
```

```
Epoch 81/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0306
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 82/200
640/640 - 1s - 950us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 83/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0304
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 84/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 85/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 86/200
640/640 - 1s - 999us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0304
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 87/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 88/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 89/200
640/640 - 1s - 988us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 90/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 91/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0144 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 92/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 93/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 94/200
640/640 - 1s - 988us/step - accuracy: 0.9922 - f1_m: 0.9922 - loss: 0.0309
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9998
Epoch 95/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 96/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 97/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0143 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 98/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 99/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 100/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0301
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
```

```
Epoch 101/200
640/640 - 1s - 994us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 102/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0304
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 103/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302 -
mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 104/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 105/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0301
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 106/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 107/200
640/640 - 1s - 963us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0142 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 108/200
640/640 - 1s - 999us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0301
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 109/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0303
- mae_m: 0.0142 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 110/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0301
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 111/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0301
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 112/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0142 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 113/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 114/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 115/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 116/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 117/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9922 - loss: 0.0301
- mae_m: 0.0142 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 118/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0301
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 119/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0142 - precision_m: 0.9848 - recall_m: 1.0000
Epoch 120/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299
- mae_m: 0.0142 - precision_m: 0.9848 - recall_m: 0.9999
```

```
Epoch 121/200
640/640 - 1s - 950us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 122/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 123/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299 -
mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 124/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 125/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 126/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0142 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 127/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0302
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 128/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0300
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 129/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 130/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299
- mae_m: 0.0143 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 131/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 132/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 133/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0142 - precision_m: 0.9848 - recall_m: 1.0000
Epoch 134/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298 -
mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 135/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297 -
mae_m: 0.0142 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 136/200
640/640 - 1s - 970us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 137/200
640/640 - 1s - 983us/step - accuracy: 0.9923 - f1_m: 0.9924 - loss: 0.0298
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 138/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0142 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 139/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 140/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298 -
mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
```

```
Epoch 141/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0142 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 142/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 143/200
640/640 - 1s - 999us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0298
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 144/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 145/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9924 - loss: 0.0297
- mae_m: 0.0141 - precision_m: 0.9851 - recall_m: 0.9999
Epoch 146/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9924 - loss: 0.0298
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 147/200
640/640 - 1s - 970us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0299
- mae_m: 0.0142 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 148/200
640/640 - 1s - 959us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 149/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9924 - loss: 0.0297
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 150/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 151/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 152/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 153/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 154/200
640/640 - 1s - 974us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297
- mae_m: 0.0142 - precision_m: 0.9848 - recall_m: 0.9999
Epoch 155/200
640/640 - 1s - 954us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 156/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 157/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296 -
mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 158/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 159/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0297
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 160/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9924 - loss: 0.0296
- mae_m: 0.0142 - precision_m: 0.9850 - recall_m: 0.9999
```

```
Epoch 161/200
640/640 - 1s - 974us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 162/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 163/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 164/200
640/640 - 1s - 999us/step - accuracy: 0.9924 - f1_m: 0.9924 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 165/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 166/200
640/640 - 1s - 963us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 167/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 168/200
640/640 - 1s - 999us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0142 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 169/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 170/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 171/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 172/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 173/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 174/200
640/640 - 1s - 993us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 175/200
640/640 - 1s - 928us/step - accuracy: 0.9924 - f1_m: 0.9924 - loss: 0.0294
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 176/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 177/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 178/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 179/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 180/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294 -
mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
```

```
Epoch 181/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294 -
mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 182/200
640/640 - 1s - 984us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294
- mae_m: 0.0140 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 183/200
640/640 - 1s - 979us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 184/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 185/200
640/640 - 1s - 1ms/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294 -
mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 186/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0293
- mae_m: 0.0140 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 187/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 188/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0296
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9998
Epoch 189/200
640/640 - 1s - 965us/step - accuracy: 0.9924 - f1_m: 0.9924 - loss: 0.0293
- mae_m: 0.0140 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 190/200
640/640 - 1s - 946us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 191/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0293
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 192/200
640/640 - 1s - 965us/step - accuracy: 0.9923 - f1_m: 0.9924 - loss: 0.0293
- mae_m: 0.0140 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 193/200
640/640 - 1s - 989us/step - accuracy: 0.9924 - f1_m: 0.9924 - loss: 0.0293
- mae_m: 0.0140 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 194/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9924 - loss: 0.0296
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 195/200
640/640 - 1s - 989us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0295
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9998
Epoch 196/200
640/640 - 1s - 1ms/step - accuracy: 0.9924 - f1_m: 0.9923 - loss: 0.0293 -
mae_m: 0.0140 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 197/200
640/640 - 1s - 964us/step - accuracy: 0.9924 - f1_m: 0.9924 - loss: 0.0292
- mae_m: 0.0140 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 198/200
640/640 - 1s - 964us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0293
- mae_m: 0.0140 - precision_m: 0.9849 - recall_m: 0.9999
Epoch 199/200
640/640 - 1s - 975us/step - accuracy: 0.9923 - f1_m: 0.9924 - loss: 0.0293
- mae_m: 0.0141 - precision_m: 0.9850 - recall_m: 0.9999
Epoch 200/200
640/640 - 1s - 940us/step - accuracy: 0.9923 - f1_m: 0.9923 - loss: 0.0294
- mae_m: 0.0141 - precision_m: 0.9849 - recall_m: 0.9999
```

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,
```

```
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 1s 926us/step - accuracy: 0.9930 - f1_m: 0.
9928 - loss: 0.0303 - mae_m: 0.0122 - precision_m: 0.9865 - recall_m: 0.99
98
```

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, optimal_feat
    model.add(GRU(20, return_sequences=False))  # Set return_sequences=Fa
    model.add(Dense(1, activation='sigmoid'))  # For binary classificatio
    # 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 fea

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

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

# Instantiate the model
model = build_model()

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

```
Epoch 1/200
640/640 - 3s - 5ms/step - accuracy: 0.9701 - loss: 0.1293
Epoch 2/200
640/640 - 1s - 2ms/step - accuracy: 0.9910 - loss: 0.0412
Epoch 3/200
640/640 - 1s - 2ms/step - accuracy: 0.9913 - loss: 0.0377
Epoch 4/200
640/640 - 1s - 2ms/step - accuracy: 0.9914 - loss: 0.0364
Epoch 5/200
640/640 - 1s - 2ms/step - accuracy: 0.9914 - loss: 0.0356
Epoch 6/200
640/640 - 1s - 2ms/step - accuracy: 0.9914 - loss: 0.0350
Epoch 7/200
640/640 - 1s - 2ms/step - accuracy: 0.9915 - loss: 0.0347
Epoch 8/200
640/640 - 1s - 2ms/step - accuracy: 0.9915 - loss: 0.0344
Epoch 9/200
640/640 - 1s - 2ms/step - accuracy: 0.9916 - loss: 0.0342
Epoch 10/200
640/640 - 1s - 2ms/step - accuracy: 0.9919 - loss: 0.0340
Epoch 11/200
640/640 - 1s - 2ms/step - accuracy: 0.9920 - loss: 0.0338
Epoch 12/200
640/640 - 1s - 2ms/step - accuracy: 0.9921 - loss: 0.0337
Epoch 13/200
640/640 - 1s - 2ms/step - accuracy: 0.9921 - loss: 0.0334
Epoch 14/200
640/640 - 1s - 2ms/step - accuracy: 0.9921 - loss: 0.0333
Epoch 15/200
640/640 - 1s - 2ms/step - accuracy: 0.9921 - loss: 0.0332
Epoch 16/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0331
Epoch 17/200
640/640 - 1s - 2ms/step - accuracy: 0.9921 - loss: 0.0331
Epoch 18/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0329
Epoch 19/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0328
Epoch 20/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0327
Epoch 21/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0325
Epoch 22/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0324
Epoch 23/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0323
Epoch 24/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0322
Epoch 25/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0321
Epoch 26/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0320
Epoch 27/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0319
Epoch 28/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0319
Epoch 29/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0318
Epoch 30/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0318
```

```
Epoch 31/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0317
Epoch 32/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0317
Epoch 33/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0316
Epoch 34/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0316
Epoch 35/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0316
Epoch 36/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0315
Epoch 37/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0314
Epoch 38/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0314
Epoch 39/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0314
Epoch 40/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0313
Epoch 41/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0313
Epoch 42/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0312
Epoch 43/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0312
Epoch 44/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0312
Epoch 45/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0311
Epoch 46/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0311
Epoch 47/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0312
Epoch 48/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 49/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 50/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 51/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 52/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0309
Epoch 53/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0309
Epoch 54/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 55/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 56/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 57/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 58/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 59/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0306
Epoch 60/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0307
```

```
Epoch 61/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0307
Epoch 62/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0305
Epoch 63/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0306
Epoch 64/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0305
Epoch 65/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0305
Epoch 66/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0305
Epoch 67/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0305
Epoch 68/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 69/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 70/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 71/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 72/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0303
Epoch 73/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0303
Epoch 74/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0303
Epoch 75/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0301
Epoch 76/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0302
Epoch 77/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 78/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 79/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 80/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0301
Epoch 81/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 82/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 83/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 84/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 85/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 86/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 87/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0299
Epoch 88/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0299
Epoch 89/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0299
Epoch 90/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
```

```
Epoch 91/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 92/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 93/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 94/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 95/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 96/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 97/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0298
Epoch 98/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 99/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 100/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 101/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 102/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 103/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0296
Epoch 104/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 105/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 106/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0295
Epoch 107/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 108/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0295
Epoch 109/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 110/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0295
Epoch 111/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 112/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 113/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0295
Epoch 114/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 115/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 116/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 117/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 118/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0294
Epoch 119/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 120/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0293
```

```
Epoch 121/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 122/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 123/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0294
Epoch 124/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 125/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 126/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 127/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 128/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 129/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0293
Epoch 130/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 131/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0292
Epoch 132/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 133/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 134/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 135/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0292
Epoch 136/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0291
Epoch 137/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0292
Epoch 138/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 139/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 140/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 141/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 142/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0291
Epoch 143/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 144/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0291
Epoch 145/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0291
Epoch 146/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 147/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0291
Epoch 148/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0290
Epoch 149/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 150/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
```

```
Epoch 151/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0290
Epoch 152/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0290
Epoch 153/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 154/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0291
Epoch 155/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 156/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 157/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0289
Epoch 158/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 159/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 160/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 161/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 162/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 163/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 164/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 165/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 166/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 167/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0289
Epoch 168/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 169/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 170/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 171/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 172/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 173/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 174/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 175/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 176/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 177/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 178/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 179/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 180/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
```

```
Epoch 181/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 182/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 183/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0286
Epoch 184/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0287
Epoch 185/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 186/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0286
Epoch 187/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0286
Epoch 188/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0286
Epoch 189/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0286
Epoch 190/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0285
Epoch 191/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0285
Epoch 192/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0286
Epoch 193/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0286
Epoch 194/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0285
Epoch 195/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0285
Epoch 196/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0286
Epoch 197/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0285
Epoch 198/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0286
Epoch 199/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0286
Epoch 200/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0285
```

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
```

```
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 1s 866us/step - accuracy: 0.9930 - loss: 0.
0290
```

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

Out[ ]:
```
(128000, 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, optimal_fea
    model.add(LSTM(20, return_sequences=False))  # Set return_sequences=F
    model.add(Dense(1, activation='sigmoid'))  # Use 1 neuron and sigmoia
    # 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 fea

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

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

# Instantiate the model
model = build_model()

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

```
Epoch 1/200
640/640 - 3s - 5ms/step - accuracy: 0.9699 - loss: 0.1663
Epoch 2/200
640/640 - 1s - 2ms/step - accuracy: 0.9909 - loss: 0.0447
Epoch 3/200
640/640 - 1s - 2ms/step - accuracy: 0.9912 - loss: 0.0401
Epoch 4/200
640/640 - 1s - 2ms/step - accuracy: 0.9912 - loss: 0.0372
Epoch 5/200
640/640 - 1s - 2ms/step - accuracy: 0.9914 - loss: 0.0357
Epoch 6/200
640/640 - 1s - 2ms/step - accuracy: 0.9917 - loss: 0.0349
Epoch 7/200
640/640 - 1s - 2ms/step - accuracy: 0.9920 - loss: 0.0344
Epoch 8/200
640/640 - 1s - 2ms/step - accuracy: 0.9921 - loss: 0.0340
Epoch 9/200
640/640 - 1s - 2ms/step - accuracy: 0.9921 - loss: 0.0336
Epoch 10/200
640/640 - 1s - 2ms/step - accuracy: 0.9921 - loss: 0.0333
Epoch 11/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0330
Epoch 12/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0328
Epoch 13/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0327
Epoch 14/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0325
Epoch 15/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0325
Epoch 16/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0323
Epoch 17/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0322
Epoch 18/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0321
Epoch 19/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0320
Epoch 20/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0319
Epoch 21/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0319
Epoch 22/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0318
Epoch 23/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0317
Epoch 24/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0316
Epoch 25/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0316
Epoch 26/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0316
Epoch 27/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0315
Epoch 28/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0315
Epoch 29/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0315
Epoch 30/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0314
```

```
Epoch 31/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0313
Epoch 32/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0313
Epoch 33/200
640/640 - 1s - 2ms/step - accuracy: 0.9922 - loss: 0.0313
Epoch 34/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0312
Epoch 35/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0312
Epoch 36/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0312
Epoch 37/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 38/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0311
Epoch 39/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 40/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 41/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0311
Epoch 42/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0311
Epoch 43/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 44/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0310
Epoch 45/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0309
Epoch 46/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0309
Epoch 47/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 48/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0309
Epoch 49/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 50/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 51/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0308
Epoch 52/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0307
Epoch 53/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0307
Epoch 54/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0307
Epoch 55/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0306
Epoch 56/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0307
Epoch 57/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0307
Epoch 58/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0306
Epoch 59/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0307
Epoch 60/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0306
```

```
Epoch 61/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0306
Epoch 62/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0306
Epoch 63/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0305
Epoch 64/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 65/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0305
Epoch 66/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 67/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0305
Epoch 68/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0303
Epoch 69/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 70/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 71/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 72/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0303
Epoch 73/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0304
Epoch 74/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0303
Epoch 75/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0303
Epoch 76/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 77/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0303
Epoch 78/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 79/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 80/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0301
Epoch 81/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 82/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 83/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0302
Epoch 84/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0301
Epoch 85/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0301
Epoch 86/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 87/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 88/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 89/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0301
Epoch 90/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
```

```
Epoch 91/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 92/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 93/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 94/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0299
Epoch 95/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 96/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 97/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 98/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0300
Epoch 99/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0299
Epoch 100/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 101/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0299
Epoch 102/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 103/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0299
Epoch 104/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 105/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 106/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 107/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 108/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 109/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 110/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0298
Epoch 111/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 112/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 113/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 114/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 115/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 116/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 117/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0297
Epoch 118/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 119/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 120/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
```

```
Epoch 121/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 122/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 123/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 124/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 125/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0296
Epoch 126/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 127/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 128/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0294
Epoch 129/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 130/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 131/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 132/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0295
Epoch 133/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 134/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 135/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 136/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 137/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 138/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 139/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0294
Epoch 140/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 141/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 142/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 143/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 144/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0293
Epoch 145/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0292
Epoch 146/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0292
Epoch 147/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 148/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 149/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0291
Epoch 150/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
```

```
Epoch 151/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 152/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0292
Epoch 153/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0291
Epoch 154/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 155/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 156/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 157/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 158/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0290
Epoch 159/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0291
Epoch 160/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0290
Epoch 161/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 162/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 163/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 164/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 165/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 166/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 167/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 168/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 169/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0290
Epoch 170/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 171/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0289
Epoch 172/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 173/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0290
Epoch 174/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0289
Epoch 175/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0289
Epoch 176/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0289
Epoch 177/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0289
Epoch 178/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0289
Epoch 179/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 180/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
```

```
Epoch 181/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 182/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 183/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 184/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0289
Epoch 185/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 186/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 187/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 188/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 189/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 190/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0288
Epoch 191/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 192/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0287
Epoch 193/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0288
Epoch 194/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 195/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0287
Epoch 196/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 197/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
Epoch 198/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0287
Epoch 199/200
640/640 - 1s - 2ms/step - accuracy: 0.9923 - loss: 0.0287
Epoch 200/200
640/640 - 1s - 2ms/step - accuracy: 0.9924 - loss: 0.0287
```

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
```

```
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 1s 852us/step - accuracy: 0.9931 - loss: 0.
0292
```

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
                                                                     'Pre
                                                                     'Rec
                                                                     'F1-
                                                                     'MAE
                                                                     'tim
                                                                     'tim
                                                                     'tot
                                                                     })
```

Out[ ]:

	Accuracy	Recall	Precision	F1-Score	Mean Absolute Error	time to train
DecisionTreeClassifier	98.60%	98.58%	98.64%	98.61%	0.014000	1.0
RandomForestClassifier	99.22%	99.92%	98.55%	99.23%	0.007812	4.4
GradientBoostingClassifier	99.25%	99.99%	98.54%	99.26%	0.007500	38.9
LogisticRegression	98.96%	99.67%	98.29%	98.97%	0.010406	0.2
KNeighborsClassifier	99.11%	99.74%	98.51%	99.12%	0.008906	0.0
ExtraTreesClassifier	99.19%	99.83%	98.58%	99.20%	0.008094	2.0
SGDClassifier	98.89%	99.66%	98.16%	98.90%	0.011125	0.4
LinearSVC	98.90%	99.67%	98.17%	98.92%	0.011000	1.5
SVM model (OneClassSVM)	98.92%	0.00%	0.00%	0.00%	0.010796	11.4
Ensemble (without OC)	99.24%	99.24%	99.25%	99.24%	0.007562	48.0
MLP (Keras)	99.23%	99.96%	98.54%	99.22%	0.013026	124.8
GRU (Keras)	99.24%	99.24%	99.24%	99.24%	0.992375	220.9
LSTM (Keras)	99.24%	99.24%	99.24%	99.24%	0.992437	208.8

9 | End of Performance Monitoring

In [ ]:
```
# Convert seconds to minutes and seconds
def convert_seconds(seconds):
    minutes = seconds // 60
    remaining_seconds = seconds % 60
    return f"{minutes} minutes, {remaining_seconds:.2f} seconds"

# Convert bytes to megabytes
def bytes_to_mb(bytes_value):
    return bytes_value / (1024 * 1024)

# Record the end time and final system statistics
notebook_end_time = time.time()
```

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

# Stop the CPU sampling
cpu_thread.join(0)  # Stop the thread by joining it with a timeout of 0

# Calculate the total run time
total_run_time = notebook_end_time - notebook_start_time

# Calculate CPU usage (in seconds, specific to the Jupyter process)
cpu_usage = {
    'user': end_cpu_times.user - start_cpu_times.user,
    'system': end_cpu_times.system - start_cpu_times.system,
}

# 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 (specific to the Jupyter process)
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 (global)
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
}

# Calculate additional CPU metrics
average_cpu_usage = sum(cpu_percentages) / len(cpu_percentages) if cpu_pe
peak_cpu_usage = max(cpu_percentages) if cpu_percentages else 0

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

# Convert metrics to more readable format
readable_metrics = {
    "total_run_time": convert_seconds(total_run_time),
    "cpu_usage": {k: convert_seconds(v) for k, v in cpu_usage.items()},
    "average_cpu_usage_percent": f"{average_cpu_usage:.2f}%",
    "peak_cpu_usage_percent": f"{peak_cpu_usage:.2f}%",
    "memory_usage": {k: bytes_to_mb(v) for k, v in memory_usage.items()},
    "disk_io": {k: v if 'count' in k else bytes_to_mb(v) for k, v in disk
```

```
    "network_io": {k: bytes_to_mb(v) for k, v in net_io.items()}
}

with open('system_metrics_readable.txt', 'w') as file:
    file.write(json.dumps(readable_metrics, indent=4))

# Print the statistics
print("Total run time:", readable_metrics["total_run_time"])
print("CPU usage (minutes:seconds):", readable_metrics["cpu_usage"])
print("Average CPU usage (%):", readable_metrics["average_cpu_usage_perce
print("Peak CPU usage (%):", readable_metrics["peak_cpu_usage_percent"])
print("Memory usage (MB):", readable_metrics["memory_usage"])
print("Disk I/O (counts and MB):", readable_metrics["disk_io"])
print("Network I/O (MB):", readable_metrics["network_io"])
```

```
Total run time: 20.0 minutes, 1.19 seconds
CPU usage (minutes:seconds): {'user': '30.0 minutes, 17.91 seconds', 'syst
em': '1.0 minutes, 21.34 seconds'}
Average CPU usage (%): 36.28%
Peak CPU usage (%): 100.00%
Memory usage (MB): {'rss': 1002.83984375, 'vms': 6308.08203125}
Disk I/O (counts and MB): {'read_count': 189896, 'write_count': 157183, 'r
ead_bytes': 916.7279453277588, 'write_bytes': 1875.301817893982}
Network I/O (MB): {'bytes_sent': 0.8385915756225586, 'bytes_recv': 0.93109
03549194336}
```