

# CHAPTER 1: DEFINITIONS, PROBLEM STATEMENT AND DATASET

## INTRODUCTION TO NETWORK ANOMALY DETECTION SYSTEM

### What is INTRUSION DETECTION SYSTEM?

An intrusion detection system is typically either a software application or a hardware device that monitors incoming and outgoing network traffic for signs of malicious activity or violations of security policies.

Intrusion detection systems and IDS products are often likened to intruder alarms, notifying you of any activity that might compromise your data or network.

IDS products search for suspicious behavior or signs of a potential compromise by analyzing the packets that move across your network and the network traffic patterns to identify any anomalies.

### Anomaly Based Approach

Anomaly-based intrusion detection systems can alert you to suspicious behavior that is unknown. Instead of searching for known threats, an anomaly-based detection system utilizes machine learning to train the detection system to recognize a normalized baseline. The baseline represents how the system normally behaves, and then all network activity is compared to that baseline. Rather than searching for known IOCs, anomaly-based IDS simply identifies any out-of-the-ordinary behavior to trigger alerts.

In this project, We are working on various machine learning algorithms used in Network Intrusion Detection. So our project comes into category of Anomaly Based method. So we can call our project as NETWORK ANOMALY DETECTION SYSTEM.

### Need for Network Anomaly Detection

The need for robust network anomaly detection systems is driven by several key factors:

- **Evolving Security Threats:** Cyber threats are constantly evolving, with attackers finding new ways to bypass traditional security measures. An effective anomaly detection system must adapt to new threats dynamically.
- **Increasing Network Complexity:** Modern networks encompass a wide range of devices and applications, many of which are interconnected across multiple platforms. This complexity makes it difficult to establish a baseline of "normal" behavior and identify deviations using traditional methods.
- **Operational Continuity:** Network anomalies can lead to significant disruptions in business operations and services. Detecting and addressing these anomalies promptly ensures that network services remain reliable and available.

- **Regulatory Compliance:** Many industries face stringent regulatory requirements for data security and privacy. Anomaly detection helps organizations comply with these regulations by providing tools to detect and mitigate potential security breaches.

## INTRODUCTION TO KDD-1999 DATASET AND NSL-KDD 1999 DATASET

### What is KDD-1999 Cup?

The full form of KDD - Knowledge Discovery and Data Mining . KDD Cup is the annual Data Mining and Knowledge Discovery competition organized by ACM Special Interest Group on Knowledge Discovery and Data Mining , the leading professional organization of data miners.

We are using the dataset from KDD-1999 cup which is related to Network Anomaly Detection System.

### How KDD-1999 cup dataset was generated?

The 1998 DARPA Intrusion Detection Evaluation Program , managed by MIT Lincoln Labs , aimed to evaluate intrusion detection systems using simulated data from a military network . They captured nine weeks of raw TCP dump data from a simulated U.S. Air Force LAN , resulting in millions of connection records , each representing a sequence of TCP packets between a source and target IP address , labeled as either normal or an attack . The attacks were categorized into four types , with test data including new attack types not present in the training data, making the evaluation more realistic.

#### Attack categories:

Attack Category	Definition	Attack types under this category
DOS (Denial-of-Service)	Disrupting network services, e.g., syn flood attacks	back , land , neptune , pod , smurf , teardrop
R2L (Remote-to-Local)	Unauthorized access from a remote machine, e.g., password guessing	ftp_write , guess_passwd , imap , multihop , phf , spy , warezclient , warezmaster
U2R (User-to-Root)	Gaining unauthorized root access, e.g., buffer overflow attacks	buffer_overflow , loadmodule , perl , rootkit
Probe	Scanning and surveillance of the network, e.g., port scanning	ipsweep , nmap , portsweep , satan

```
In [ ]: attack_categories_dict = {
    "Normal": ['normal'],
    "DOS": ['back', 'land', 'neptune', 'pod', 'smurf', 'teardrop'],
    "R2L": ['ftp_write', 'guess_passwd', 'imap', 'multihop', 'phf', 'spy', 'warezclient', 'warezmaster'],
    "U2R": ['buffer_overflow', 'loadmodule', 'perl', 'rootkit'],
    "Probe": ['ipsweep', 'nmap', 'portsweep', 'satan']
}
```

#### Attack types:

Attack types	Definition
back	A malicious HTTP request designed to overload a web server.
land	Spoofed SYN packets are sent with the same source and destination IP, causing the system to crash.
neptune	SYN flood is used to overwhelm and crash a targeted machine or service.
pod	Oversized ping packets are sent to crash the target system.
smurf	ICMP requests are sent with the target's IP address, flooding the target with reply traffic.
teardrop	Exploits fragmentation bugs in the TCP/IP protocol to crash the target system.
ftp_write	An unauthorized user writes files to an FTP server, potentially compromising the system.
guess_passwd	Repeated guessing of passwords to gain unauthorized access to a system.
imap	Exploiting vulnerabilities in the IMAP protocol to gain unauthorized access.
multihop	Involves a user hopping through multiple hosts to obscure their identity and gain unauthorized access.
phf	Exploiting a vulnerability in the "phf" web application to execute commands on the server.
spy	Involves unauthorized monitoring of data or communications, such as email snooping.
warezclient	Unauthorized software is downloaded from a compromised FTP server.
warezmaster	A user uploads pirated software to an FTP server for distribution.
buffer_overflow	Excess data overflows into adjacent memory, allowing the attacker to execute arbitrary code.
loadmodule	A malicious user loads a module into the system kernel, allowing privilege escalation.
perl	Vulnerabilities in Perl scripts are exploited to gain root-level access.
rootkit	A set of malicious tools is installed to maintain root access on a system while hiding the intrusion.
ipsweep	Used to discover active IP addresses on a network by systematically sending queries to multiple addresses.
nmap	Used for network mapping by scanning ports to identify available services on a target.
portsweep	Involves scanning multiple ports on a host to detect open or vulnerable services.
satan	Using the SATAN tool to scan networks for known vulnerabilities in network services.

## What is NSL-KDD Dataset?

The full form of NSL-KDD is Network Security Layer - Knowledge Discovery and Data mining . It is a derived and modified dataset of original KDD-1999 dataset.

NSL-KDD is a data set suggested to solve some of the inherent problems of the KDD-1999 data set .

### Improvements to the KDD'99 data set:

It does not include redundant records in the train set , so the classifiers will not be biased towards more frequent records.

There is no duplicate records in the proposed test sets; therefore, the performance of the learners are not biased by the methods which have better detection rates on the frequent records.

The number of selected records from each difficulty level group is inversely proportional to the percentage of records in the original KDD data set. As a result, the classification rates of distinct machine learning methods vary in a wider range, which makes it more efficient to have an accurate evaluation of different learning techniques.

The number of records in the train and test sets are reasonable, which makes it affordable to run the experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research works will be consistent and comparable.

## How 'LastFlag' or 'DifficultyLevel' feature was generated?

The "Difficulty Level" feature, named as LastFlag in the dataset, is generated by evaluating how many of the 21 trained learners were able to correctly predict the label of each record. To calculate this:

1. The dataset is split into smaller subsets, and 21 models (7 learners, each trained 3 times) are used to predict the labels of the entire KDD train and test sets.
2. For each record, a #successfulPrediction counter is initialized at zero.
3. The predicted label from each learner is compared with the actual label of the record. If the prediction matches the true label, the #successfulPrediction counter is incremented by one.
4. The LastFlag (Difficulty Level) is based on this counter, where the value ranges from 0 to 21. A higher value indicates that more learners correctly predicted the label, meaning the record was easier to classify, and a lower value suggests greater difficulty in classification.

## PROBLEM STATEMENT

In the realm of cybersecurity, network anomaly detection is a critical task that involves identifying unusual patterns or behaviors that deviate from the norm within network traffic. These anomalies could signify a range of security threats, from compromised devices and malware infections to large-scale cyber-attacks like DDoS (Distributed Denial of Service).

So assume you are working as a data scientist with cyber security department. You are provided with the Networking data (NSL-KDD dataset).

Your task is to visualise and analyse the given data. Apply Supervised Learning algorithms to find the best model which can classify the data using attack column. Apply Unsupervised Learning algorithms to find the best model to detect the anomalies in the dataset without attack column. Finally deploy the machine learning models via Flask API.

This project divided into four blocks -

1. Tableau Visualisations

- 2. EDA and Hypothesis Testing
- 3. ML Modeling
- 4. Deployment

## IMPORT LIBRARIES AND DATASET

```
In [ ]: !pip install mlflow
```

```
Collecting mlflow
  Using cached mlflow-2.22.0-py3-none-any.whl.metadata (30 kB)
Collecting mlflow-skinny==2.22.0 (from mlflow)
  Using cached mlflow_skinny-2.22.0-py3-none-any.whl.metadata (31 kB)
Collecting Flask<4 (from mlflow)
  Using cached flask-3.1.1-py3-none-any.whl.metadata (3.0 kB)
Collecting Jinja2<4,>=3.0 (from mlflow)
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting alembic!=1.10.0,<2 (from mlflow)
  Using cached alembic-1.15.2-py3-none-any.whl.metadata (7.3 kB)
Collecting docker<8,>=4.0.0 (from mlflow)
  Using cached docker-7.1.0-py3-none-any.whl.metadata (3.8 kB)
Collecting graphene<4 (from mlflow)
  Using cached graphene-3.4.3-py2.py3-none-any.whl.metadata (6.9 kB)
Collecting markdown<4,>=3.3 (from mlflow)
  Using cached markdown-3.8-py3-none-any.whl.metadata (5.1 kB)
Collecting matplotlib<4 (from mlflow)
  Downloading matplotlib-3.10.3-cp313-cp313-win_amd64.whl.metadata (11 kB)
Collecting numpy<3 (from mlflow)
  Downloading numpy-2.2.5-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting pandas<3 (from mlflow)
  Downloading pandas-2.2.3-cp313-cp313-win_amd64.whl.metadata (19 kB)
Collecting pyarrow<20,>=4.0.0 (from mlflow)
  Downloading pyarrow-19.0.1-cp313-cp313-win_amd64.whl.metadata (3.4 kB)
Collecting scikit-learn<2 (from mlflow)
  Downloading scikit_learn-1.6.1-cp313-cp313-win_amd64.whl.metadata (15 kB)
Collecting scipy<2 (from mlflow)
  Downloading scipy-1.15.3-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting sqlalchemy<3,>=1.4.0 (from mlflow)
  Downloading sqlalchemy-2.0.41-cp313-cp313-win_amd64.whl.metadata (9.8 kB)
Collecting waitress<4 (from mlflow)
  Using cached waitress-3.0.2-py3-none-any.whl.metadata (5.8 kB)
Collecting cachetools<6,>=5.0.0 (from mlflow-skinny==2.22.0->mlflow)
  Using cached cachetools-5.5.2-py3-none-any.whl.metadata (5.4 kB)
Collecting click<9,>=7.0 (from mlflow-skinny==2.22.0->mlflow)
  Downloading click-8.2.0-py3-none-any.whl.metadata (2.5 kB)
Collecting云pickle<4 (from mlflow-skinny==2.22.0->mlflow)
  Using cached云pickle-3.1.1-py3-none-any.whl.metadata (7.1 kB)
Collecting databricks-sdk<1,>=0.20.0 (from mlflow-skinny==2.22.0->mlflow)
  Using cached databricks_sdk-0.53.0-py3-none-any.whl.metadata (39 kB)
Collecting fastapi<1 (from mlflow-skinny==2.22.0->mlflow)
  Using cached fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting gitpython<4,>=3.1.9 (from mlflow-skinny==2.22.0->mlflow)
  Using cached GitPython-3.1.44-py3-none-any.whl.metadata (13 kB)
Collecting importlib_metadata!=4.7.0,<9,>=3.7.0 (from mlflow-skinny==2.22.0->mlflow)
  Downloading importlib_metadata-8.7.0-py3-none-any.whl.metadata (4.8 kB)
Collecting opentelemetry-api<3,>=1.9.0 (from mlflow-skinny==2.22.0->mlflow)
  Using cached opentelemetry_api-1.33.0-py3-none-any.whl.metadata (1.6 kB)
Collecting opentelemetry-sdk<3,>=1.9.0 (from mlflow-skinny==2.22.0->mlflow)
  Using cached opentelemetry_sdk-1.33.0-py3-none-any.whl.metadata (1.6 kB)
Collecting packaging<25 (from mlflow-skinny==2.22.0->mlflow)
  Downloading packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Collecting protobuf<7,>=3.12.0 (from mlflow-skinny==2.22.0->mlflow)
  Using cached protobuf-6.31.0-cp310-abi3-win_amd64.whl.metadata (593 bytes)
Collecting pydantic<3,>=1.10.8 (from mlflow-skinny==2.22.0->mlflow)
  Using cached pydantic-2.11.4-py3-none-any.whl.metadata (66 kB)
Collecting pyyaml<7,>=5.1 (from mlflow-skinny==2.22.0->mlflow)
  Downloading PyYAML-6.0.2-cp313-cp313-win_amd64.whl.metadata (2.1 kB)
Collecting requests<3,>=2.17.3 (from mlflow-skinny==2.22.0->mlflow)
  Downloading requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting sqlparse<1,>=0.4.0 (from mlflow-skinny==2.22.0->mlflow)
  Using cached sqlparse-0.5.3-py3-none-any.whl.metadata (3.9 kB)
Collecting typing_extensions<5,>=4.0.0 (from mlflow-skinny==2.22.0->mlflow)
  Using cached typing_extensions-4.13.2-py3-none-any.whl.metadata (3.0 kB)
Collecting uvicorn<1 (from mlflow-skinny==2.22.0->mlflow)
  Using cached uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Collecting Mako (from alembic!=1.10.0,<2->mlflow)
  Using cached mako-1.3.10-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: pywin32>=304 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from docker<8,>=4.0.0->mlflow) (310)
Collecting urllib3>=1.26.0 (from docker<8,>=4.0.0->mlflow)
  Downloading urllib3-2.4.0-py3-none-any.whl.metadata (6.5 kB)
Collecting blinker>=1.9.0 (from Flask<4->mlflow)
  Using cached blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
```

```
Collecting itsdangerous>=2.2.0 (from Flask<4->mlflow)
  Using cached itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting markupsafe>=2.1.1 (from Flask<4->mlflow)
  Downloading MarkupSafe-3.0.2-cp313-cp313-win_amd64.whl.metadata (4.1 kB)
Collecting werkzeug>=3.1.0 (from Flask<4->mlflow)
  Using cached werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting graphql-core<3.3,>=3.1 (from graphene<4->mlflow)
  Using cached graphql_core-3.2.6-py3-none-any.whl.metadata (11 kB)
Collecting graphql-relay<3.3,>=3.1 (from graphene<4->mlflow)
  Using cached graphql_relay-3.2.0-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: python-dateutil<3,>=2.7.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from graphene<4->mlflow) (2.9.0.post0)
Collecting contourpy>=1.0.1 (from matplotlib<4->mlflow)
  Downloading contourpy-1.3.2-cp313-cp313-win_amd64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib<4->mlflow)
  Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib<4->mlflow)
  Downloading fonttools-4.58.0-cp313-cp313-win_amd64.whl.metadata (106 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib<4->mlflow)
  Downloading kiwisolver-1.4.8-cp313-cp313-win_amd64.whl.metadata (6.3 kB)
Collecting pillow>=8 (from matplotlib<4->mlflow)
  Downloading pillow-11.2.1-cp313-cp313-win_amd64.whl.metadata (9.1 kB)
Collecting pyparsing>=2.3.1 (from matplotlib<4->mlflow)
  Using cached pyparsing-3.2.3-py3-none-any.whl.metadata (5.0 kB)
Collecting pytz>=2020.1 (from pandas<3->mlflow)
  Downloading pytz-2025.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas<3->mlflow)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting joblib>=1.2.0 (from scikit-learn<2->mlflow)
  Downloading joblib-1.5.0-py3-none-any.whl.metadata (5.6 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn<2->mlflow)
  Using cached threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Collecting greenlet>=1 (from sqlalchemy<3,>=1.4.0->mlflow)
  Downloading greenlet-3.2.2-cp313-cp313-win_amd64.whl.metadata (4.2 kB)
Requirement already satisfied: colorama in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from click<9,>=7.0->mlflow-skinny==2.22.0->mlflow) (0.4.6)
Collecting google-auth~=2.0 (from databricks-sdk<1,>=0.20.0->mlflow-skinny==2.22.0->mlflow)
  Using cached google_auth-2.40.1-py2.py3-none-any.whl.metadata (6.2 kB)
Collecting starlette<0.47.0,>=0.40.0 (from fastapi<1->mlflow-skinny==2.22.0->mlflow)
  Using cached starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting gitdb<5,>=4.0.1 (from gitpython<4,>=3.1.9->mlflow-skinny==2.22.0->mlflow)
  Using cached gitdb-4.0.12-py3-none-any.whl.metadata (1.2 kB)
Collecting zipp>=3.20 (from importlib_metadata!=4.7.0,<9,>=3.7.0->mlflow-skinny==2.22.0->mlflow)
  Using cached zipp-3.21.0-py3-none-any.whl.metadata (3.7 kB)
Collecting deprecated>=1.2.6 (from opentelemetry-api<3,>=1.9.0->mlflow-skinny==2.22.0->mlflow)
  Using cached Deprecated-1.2.18-py2.py3-none-any.whl.metadata (5.7 kB)
Collecting importlib_metadata!=4.7.0,<9,>=3.7.0 (from mlflow-skinny==2.22.0->mlflow)
  Using cached importlib_metadata-8.6.1-py3-none-any.whl.metadata (4.7 kB)
Collecting opentelemetry-semantic-conventions==0.54b0 (from opentelemetry-sdk<3,>=1.9.0->mlflow-skinny==2.22.0->mlflow)
  Using cached opentelemetry_semantic_conventions-0.54b0-py3-none-any.whl.metadata (2.5 kB)
Collecting annotated-types>=0.6.0 (from pydantic<3,>=1.10.8->mlflow-skinny==2.22.0->mlflow)
  Using cached annotated_types-0.7.0-py3-none-any.whl.metadata (15 kB)
Collecting pydantic-core==2.33.2 (from pydantic<3,>=1.10.8->mlflow-skinny==2.22.0->mlflow)
  Downloading pydantic_core-2.33.2-cp313-cp313-win_amd64.whl.metadata (6.9 kB)
Collecting typing-inspection>=0.4.0 (from pydantic<3,>=1.10.8->mlflow-skinny==2.22.0->mlflow)
  Using cached typing_inspection-0.4.0-py3-none-any.whl.metadata (2.6 kB)
Requirement already satisfied: six=>1.5 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil<3,>=2.7.0->graphene<4->mlflow) (1.17.0)
Collecting charset-normalizer<4,>=2 (from requests<3,>=2.17.3->mlflow-skinny==2.22.0->mlflow)
  Downloading charset_normalizer-3.4.2-cp313-cp313-win_amd64.whl.metadata (36 kB)
Collecting idna<4,>=2.5 (from requests<3,>=2.17.3->mlflow-skinny==2.22.0->mlflow)
  Downloading idna-3.10-py3-none-any.whl.metadata (10 kB)
```

```
Collecting certifi>=2017.4.17 (from requests<3,>=2.17.3->mlflow-skinny==2.22.0->mlflow)
  Downloading certifi-2025.4.26-py3-none-any.whl.metadata (2.5 kB)
Collecting h11>=0.8 (from uvicorn<1->mlflow-skinny==2.22.0->mlflow)
  Using cached h11-0.16.0-py3-none-any.whl.metadata (8.3 kB)
Collecting wrapt<2,>=1.10 (from deprecated>=1.2.6->opentelemetry-api<3,>=1.9.0->mlflow-skinny==2.22.0->mlflow)
  Downloading wrapt-1.17.2-cp313-cp313-win_amd64.whl.metadata (6.5 kB)
Collecting smmap<6,>=3.0.1 (from gitdb<5,>=4.0.1->gitpython<4,>=3.1.9->mlflow-skinny==2.22.0->mlflow)
  Using cached smmap-5.0.2-py3-none-any.whl.metadata (4.3 kB)
Collecting pyasn1-modules>=0.2.1 (from google-auth~=2.0->databricks-sdk<1,>=0.20.0->mlflow-skinny==2.22.0->mlflow)
  Using cached pyasn1_modules-0.4.2-py3-none-any.whl.metadata (3.5 kB)
Collecting rsa<5,>=3.1.4 (from google-auth~=2.0->databricks-sdk<1,>=0.20.0->mlflow-skinny==2.22.0->mlflow)
  Using cached rsa-4.9.1-py3-none-any.whl.metadata (5.6 kB)
Collecting aiohttp<5,>=3.6.2 (from starlette<0.47.0,>=0.40.0->fastapi<1->mlflow-skinny==2.22.0->mlflow)
  Using cached aiohttp-4.9.0-py3-none-any.whl.metadata (4.7 kB)
Collecting sniffio>=1.1 (from aiohttp<5,>=3.6.2->starlette<0.47.0,>=0.40.0->fastapi<1->mlflow-skinny==2.22.0->mlflow)
  Using cached sniffio-1.3.1-py3-none-any.whl.metadata (3.9 kB)
Collecting pyasn1<0.7.0,>=0.6.1 (from pyasn1-modules>=0.2.1->google-auth~=2.0->databricks-sdk<1,>=0.20.0->mlflow-skinny==2.22.0->mlflow)
  Using cached pyasn1-0.6.1-py3-none-any.whl.metadata (8.4 kB)
Using cached mlflow-2.22.0-py3-none-any.whl (29.0 MB)
Using cached mlflow_skinny-2.22.0-py3-none-any.whl (6.3 MB)
Using cached alembic-1.15.2-py3-none-any.whl (231 kB)
Using cached docker-7.1.0-py3-none-any.whl (147 kB)
Using cached flask-3.1.1-py3-none-any.whl (103 kB)
Using cached graphene-3.4.3-py2.py3-none-any.whl (114 kB)
Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
Using cached markdown-3.8-py3-none-any.whl (106 kB)
Downloading matplotlib-3.10.3-cp313-cp313-win_amd64.whl (8.1 MB)
----- 0.0/8.1 MB ? eta -:---:-
----- 1.8/8.1 MB 9.5 MB/s eta 0:00:01
----- 4.2/8.1 MB 10.4 MB/s eta 0:00:01
----- 6.3/8.1 MB 10.8 MB/s eta 0:00:01
----- 8.1/8.1 MB 10.5 MB/s eta 0:00:00
Downloading numpy-2.2.5-cp313-cp313-win_amd64.whl (12.6 MB)
----- 0.0/12.6 MB ? eta -:---:-
----- 2.4/12.6 MB 11.8 MB/s eta 0:00:01
----- 5.0/12.6 MB 11.8 MB/s eta 0:00:01
----- 7.3/12.6 MB 11.6 MB/s eta 0:00:01
----- 10.0/12.6 MB 11.8 MB/s eta 0:00:01
----- 12.3/12.6 MB 11.8 MB/s eta 0:00:01
----- 12.6/12.6 MB 11.2 MB/s eta 0:00:00
Downloading pandas-2.2.3-cp313-cp313-win_amd64.whl (11.5 MB)
----- 0.0/11.5 MB ? eta -:---:-
----- 2.4/11.5 MB 11.7 MB/s eta 0:00:01
----- 4.7/11.5 MB 11.8 MB/s eta 0:00:01
----- 7.3/11.5 MB 11.8 MB/s eta 0:00:01
----- 9.7/11.5 MB 11.8 MB/s eta 0:00:01
----- 11.5/11.5 MB 11.4 MB/s eta 0:00:00
Downloading pyarrow-19.0.1-cp313-cp313-win_amd64.whl (25.2 MB)
----- 0.0/25.2 MB ? eta -:---:-
----- 1.8/25.2 MB 11.2 MB/s eta 0:00:03
----- 3.9/25.2 MB 9.5 MB/s eta 0:00:03
----- 6.6/25.2 MB 10.3 MB/s eta 0:00:02
----- 8.9/25.2 MB 10.7 MB/s eta 0:00:02
----- 11.5/25.2 MB 10.9 MB/s eta 0:00:02
----- 13.9/25.2 MB 11.0 MB/s eta 0:00:02
----- 16.3/25.2 MB 11.1 MB/s eta 0:00:01
----- 18.6/25.2 MB 11.2 MB/s eta 0:00:01
----- 21.0/25.2 MB 11.2 MB/s eta 0:00:01
----- 23.6/25.2 MB 11.3 MB/s eta 0:00:01
----- 25.2/25.2 MB 10.9 MB/s eta 0:00:00
Downloading scikit_learn-1.6.1-cp313-cp313-win_amd64.whl (11.1 MB)
----- 0.0/11.1 MB ? eta -:---:-
----- 2.4/11.1 MB 11.8 MB/s eta 0:00:01
----- 4.7/11.1 MB 11.8 MB/s eta 0:00:01
----- 7.1/11.1 MB 11.7 MB/s eta 0:00:01
----- 9.4/11.1 MB 11.8 MB/s eta 0:00:01
```

```
----- 11.1/11.1 MB 11.2 MB/s eta 0:00:00
Downloading scipy-1.15.3-cp313-cp313-win_amd64.whl (41.0 MB)
----- 0.0/41.0 MB ? eta -:-:-
----- 2.4/41.0 MB 11.6 MB/s eta 0:00:04
----- 4.5/41.0 MB 11.8 MB/s eta 0:00:04
----- 7.1/41.0 MB 11.7 MB/s eta 0:00:03
----- 9.7/41.0 MB 11.8 MB/s eta 0:00:03
----- 12.1/41.0 MB 11.8 MB/s eta 0:00:03
----- 14.4/41.0 MB 11.6 MB/s eta 0:00:03
----- 16.5/41.0 MB 11.6 MB/s eta 0:00:03
----- 18.4/41.0 MB 11.4 MB/s eta 0:00:02
----- 20.4/41.0 MB 11.0 MB/s eta 0:00:02
----- 22.0/41.0 MB 10.7 MB/s eta 0:00:02
----- 24.4/41.0 MB 10.8 MB/s eta 0:00:02
----- 27.0/41.0 MB 10.8 MB/s eta 0:00:02
----- 29.4/41.0 MB 10.9 MB/s eta 0:00:02
----- 31.7/41.0 MB 11.0 MB/s eta 0:00:01
----- 34.1/41.0 MB 11.0 MB/s eta 0:00:01
----- 36.7/41.0 MB 11.1 MB/s eta 0:00:01
----- 39.3/41.0 MB 11.1 MB/s eta 0:00:01
----- 40.9/41.0 MB 11.2 MB/s eta 0:00:01
----- 41.0/41.0 MB 10.8 MB/s eta 0:00:00
Downloading sqlalchemy-2.0.41-cp313-cp313-win_amd64.whl (2.1 MB)
----- 0.0/2.1 MB ? eta -:-:-
----- 2.1/2.1 MB 10.9 MB/s eta 0:00:00
Using cached waitress-3.0.2-py3-none-any.whl (56 kB)
Using cached blinker-1.9.0-py3-none-any.whl (8.5 kB)
Using cached cachetools-5.5.2-py3-none-any.whl (10 kB)
Downloading click-8.2.0-py3-none-any.whl (102 kB)
Using cached cloudpickle-3.1.1-py3-none-any.whl (20 kB)
Downloading contourpy-1.3.2-cp313-cp313-win_amd64.whl (223 kB)
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Using cached databricks_sdk-0.53.0-py3-none-any.whl (700 kB)
Using cached fastapi-0.115.12-py3-none-any.whl (95 kB)
Downloading fonttools-4.58.0-cp313-cp313-win_amd64.whl (2.2 MB)
----- 0.0/2.2 MB ? eta -:-:-
----- 2.2/2.2 MB 11.0 MB/s eta 0:00:00
Using cached GitPython-3.1.44-py3-none-any.whl (207 kB)
Using cached graphql_core-3.2.6-py3-none-any.whl (203 kB)
Using cached graphql_relay-3.2.0-py3-none-any.whl (16 kB)
Downloading greenlet-3.2.2-cp313-cp313-win_amd64.whl (296 kB)
Using cached itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading joblib-1.5.0-py3-none-any.whl (307 kB)
Downloading kiwisolver-1.4.8-cp313-cp313-win_amd64.whl (71 kB)
Downloading MarkupSafe-3.0.2-cp313-cp313-win_amd64.whl (15 kB)
Using cached opentelemetry_api-1.33.0-py3-none-any.whl (65 kB)
Using cached importlib_metadata-8.6.1-py3-none-any.whl (26 kB)
Using cached opentelemetry_sdk-1.33.0-py3-none-any.whl (118 kB)
Using cached opentelemetry_semantic_conventions-0.54b0-py3-none-any.whl (194 kB)
Downloading packaging-24.2-py3-none-any.whl (65 kB)
Downloading pillow-11.2.1-cp313-cp313-win_amd64.whl (2.7 MB)
----- 0.0/2.7 MB ? eta -:-:-
----- 2.4/2.7 MB 11.8 MB/s eta 0:00:01
----- 2.7/2.7 MB 10.9 MB/s eta 0:00:00
Using cached protobuf-6.31.0-cp310-abi3-win_amd64.whl (435 kB)
Using cached pydantic-2.11.4-py3-none-any.whl (443 kB)
Downloading pydantic_core-2.33.2-cp313-cp313-win_amd64.whl (2.0 MB)
----- 0.0/2.0 MB ? eta -:-:-
----- 2.0/2.0 MB 11.0 MB/s eta 0:00:00
Using cached pyparsing-3.2.3-py3-none-any.whl (111 kB)
Downloading pytz-2025.2-py2.py3-none-any.whl (509 kB)
Downloading PyYAML-6.0.2-cp313-cp313-win_amd64.whl (156 kB)
Downloading requests-2.32.3-py3-none-any.whl (64 kB)
Using cached sqlparse-0.5.3-py3-none-any.whl (44 kB)
Using cached threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Using cached typing_extensions-4.13.2-py3-none-any.whl (45 kB)
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Downloading urllib3-2.4.0-py3-none-any.whl (128 kB)
Using cached uvicorn-0.34.2-py3-none-any.whl (62 kB)
Using cached werkzeug-3.1.3-py3-none-any.whl (224 kB)
Using cached mako-1.3.10-py3-none-any.whl (78 kB)
Using cached annotated_types-0.7.0-py3-none-any.whl (13 kB)
Downloading certifi-2025.4.26-py3-none-any.whl (159 kB)
Downloading charset_normalizer-3.4.2-cp313-cp313-win_amd64.whl (105 kB)
```

```
Using cached Deprecated-1.2.18-py2.py3-none-any.whl (10.0 kB)
Using cached gitdb-4.0.12-py3-none-any.whl (62 kB)
Using cached google_auth-2.40.1-py2.py3-none-any.whl (216 kB)
Using cached h11-0.16.0-py3-none-any.whl (37 kB)
Downloading idna-3.10-py3-none-any.whl (70 kB)
Using cached starlette-0.46.2-py3-none-any.whl (72 kB)
Using cached typing_inspection-0.4.0-py3-none-any.whl (14 kB)
Using cached zipp-3.21.0-py3-none-any.whl (9.6 kB)
Using cached anyio-4.9.0-py3-none-any.whl (100 kB)
Using cached pyasn1_modules-0.4.2-py3-none-any.whl (181 kB)
Using cached rsa-4.9.1-py3-none-any.whl (34 kB)
Using cached smmap-5.0.2-py3-none-any.whl (24 kB)
Downloading wrapt-1.17.2-cp313-cp313-win_amd64.whl (38 kB)
Using cached pyasn1-0.6.1-py3-none-any.whl (83 kB)
Using cached sniffio-1.3.1-py3-none-any.whl (10 kB)
Installing collected packages: pytz, zipp, wrapt, waitress, urllib3, tzdata, typing_extensions, threadpoolctl, sqlparse, sniffio, smmap, pyyaml, pyparsing, pyasn1, pyarrow, protobuf, pillow, packaging, numpy, markupsafe, markdown, kiwisolver, joblib, itsdangerous, idna, h11, greenlet, graphql-core, fonttools, cycler, cloudpickle, click, charset-normalizer, certifi, cachetools, blinker, annotated-types, werkzeug, uvicorn, typing-inspection, sqlalchemy, scipy, rsa, requests, pydantic-core, pyasn1-modules, pandas, Mako, Jinja2, importlib_metadata, graphql-relay, gitdb, deprecated, contourpy, anyio, starlette, scikit-learn, pydantic, opentelemetry-api, matplotlib, graphene, google-auth, gitpython, Flask, docker, alembic, opentelemetry-semantic-conventions, fastapi, databricks-sdk, opentelemetry-sdk, mlflow-skinny, mlflow
Attempting uninstall: packaging
  Found existing installation: packaging 25.0
  Uninstalling packaging-25.0:
    Successfully uninstalled packaging-25.0
Successfully installed Flask-3.1.1 Jinja2-3.1.6 Mako-1.3.10 alembic-1.15.2 annotated-types-0.7.0 anyio-4.9.0 blinker-1.9.0 cachetools-5.5.2 certifi-2025.4.26 charset-normalizer-3.4.2 click-8.2.0 cloudpickle-3.1.1 contourpy-1.3.2 cycler-0.12.1 databricks-sdk-0.53.0 deprecated-1.2.18 docker-7.1.0 fastapi-0.115.12 fonttools-4.58.0 gitdb-4.0.12 gitpython-3.1.44 google-auth-2.40.1 graphene-3.4.3 graphql-core-3.2.6 graphql-relay-3.2.0 greenlet-3.2.2 h11-0.16.0 idna-3.10 importlib_metadata-8.6.1 itsdangerous-2.2.0 joblib-1.5.0 kiwisolver-1.4.8 markdown-3.8 markupsafe-3.0.2 matplotlib-3.10.3 mlflow-2.22.0 mlflow-skinny-2.22.0 numpy-2.2.5 opentelemetry-api-1.33.0 opentelemetry-sdk-1.33.0 opentelemetry-semantic-conventions-0.54b0 packaging-24.2 pandas-2.2.3 pillow-11.2.1 protobuf-6.31.0 pyarrow-19.0.1 pyasn1-0.6.1 pyasn1-modules-0.4.2 pydantic-2.1.4 pydantic-core-2.33.2 pyparsing-3.2.3 pytz-2025.2 pyyaml-6.0.2 requests-2.32.3 rsa-4.9.1 scikit-learn-1.6.1 scipy-1.15.3 smmap-5.0.2 sniffio-1.3.1 sqlalchemy-2.0.41 sqlparse-0.5.3 starlette-0.46.2 threadpoolctl-3.6.0 typing-extensions-4.13.2 typing-inspection-0.4.0 tzdata-2025.2 urllib3-2.4.0 uvicorn-0.34.2 waitress-3.0.2 werkzeug-3.1.3 wrapt-1.17.2 zipp-3.21.0
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
In [ ]: !pip install matplotlib
!pip install seaborn
!pip install statsmodels

Requirement already satisfied: matplotlib in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (3.10.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (2.2.5)
Requirement already satisfied: packaging>=20.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Collecting seaborn
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from seaborn) (2.2.5)
Requirement already satisfied: pandas>=1.2 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from seaborn) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from seaborn) (3.10.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.2)
Requirement already satisfied: cycler>=0.10 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.5.8.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Installing collected packages: seaborn
Successfully installed seaborn-0.13.2
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Collecting statsmodels
  Downloading statsmodels-0.14.4-cp313-cp313-win_amd64.whl.metadata (9.5 kB)
Requirement already satisfied: numpy<3,>=1.22.3 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from statsmodels) (2.2.5)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from statsmodels) (1.15.3)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from statsmodels) (2.2.3)
Collecting patsy>=0.5.6 (from statsmodels)
  Using cached patsy-1.0.1-py2.py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: packaging>=21.3 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from statsmodels) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.17.0)
Downloading statsmodels-0.14.4-cp313-cp313-win_amd64.whl (9.8 MB)
----- 0.0/9.8 MB ? eta ---:--
----- 0.5/9.8 MB 3.1 MB/s eta 0:00:03
----- 1.0/9.8 MB 3.3 MB/s eta 0:00:03
----- 2.1/9.8 MB 3.9 MB/s eta 0:00:02
----- 3.1/9.8 MB 4.2 MB/s eta 0:00:02
----- 4.7/9.8 MB 4.9 MB/s eta 0:00:02
----- 6.0/9.8 MB 5.2 MB/s eta 0:00:01
----- 7.6/9.8 MB 5.4 MB/s eta 0:00:01
----- 9.2/9.8 MB 5.7 MB/s eta 0:00:01
----- 9.8/9.8 MB 5.5 MB/s eta 0:00:00
Using cached patsy-1.0.1-py2.py3-none-any.whl (232 kB)
Installing collected packages: patsy, statsmodels
Successfully installed patsy-1.0.1 statsmodels-0.14.4
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
In [ ]: !pip install sklearn
!pip install six
!pip install xgboost
!pip install imblearn

!pip install lightgbm
```

```
Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'error'
```

```
error: subprocess-exited-with-error

  × Getting requirements to build wheel did not run successfully.
    | exit code: 1
    |> [15 lines of output]
      The 'sklearn' PyPI package is deprecated, use 'scikit-learn'
      rather than 'sklearn' for pip commands.

      Here is how to fix this error in the main use cases:
      - use 'pip install scikit-learn' rather than 'pip install sklearn'
      - replace 'sklearn' by 'scikit-learn' in your pip requirements files
        (requirements.txt, setup.py, setup.cfg, Pipfile, etc ...)
      - if the 'sklearn' package is used by one of your dependencies,
        it would be great if you take some time to track which package uses
        'sklearn' instead of 'scikit-learn' and report it to their issue tracker
      - as a last resort, set the environment variable
        SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True to avoid this error
```

More information is available at  
<https://github.com/scikit-learn/scikit-learn-pypi-package>  
[end of output]

note: This error originates from a subprocess, and is likely not a problem with pip.

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
error: subprocess-exited-with-error
```

```
  × Getting requirements to build wheel did not run successfully.
    | exit code: 1
    |> See above for output.
```

note: This error originates from a subprocess, and is likely not a problem with pip.

```
Collecting umap
  Using cached umap-0.1.1.tar.gz (3.2 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Building wheels for collected packages: umap
  Building wheel for umap (pyproject.toml): started
  Building wheel for umap (pyproject.toml): finished with status 'done'
  Created wheel for umap: filename=umap-0.1.1-py3-none-any.whl size=3585 sha256=0b3fa
64511351264d4b843883da12aaad50261a5eab04701821d97fe7642cd33
  Stored in directory: c:\users\amaan\appdata\local\pip\cache\wheels\18\d5\8e\40d69c4
1defe88bec50a72f118c4e604eea8e9f2e7830170c3
Successfully built umap
Installing collected packages: umap
Successfully installed umap-0.1.1
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: six in c:\users\amaan\appdata\local\programs\python\py
thon313\lib\site-packages (1.17.0)
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Collecting xgboost
  Using cached xgboost-3.0.1-py3-none-win_amd64.whl.metadata (2.1 kB)
  Requirement already satisfied: numpy in c:\users\amaan\appdata\local\programs\python
\python313\lib\site-packages (from xgboost) (2.2.5)
  Requirement already satisfied: scipy in c:\users\amaan\appdata\local\programs\python
\python313\lib\site-packages (from xgboost) (1.15.3)
  Using cached xgboost-3.0.1-py3-none-win_amd64.whl (150.0 MB)
  Installing collected packages: xgboost
  Successfully installed xgboost-3.0.1
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Collecting imblearn
  Using cached imblearn-0.0-py2.py3-none-any.whl.metadata (355 bytes)
Collecting imbalanced-learn (from imblearn)
  Using cached imbalanced_learn-0.13.0-py3-none-any.whl.metadata (8.8 kB)
Requirement already satisfied: numpy<3,>=1.24.3 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from imbalanced-learn->imblearn) (2.2.5)
Requirement already satisfied: scipy<2,>=1.10.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from imbalanced-learn->imblearn) (1.15.3)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from imbalanced-learn->imblearn) (1.6.1)
Collecting sklearn-compat<1,>=0.1 (from imbalanced-learn->imblearn)
  Using cached sklearn_compatible-0.1.3-py3-none-any.whl.metadata (18 kB)
Requirement already satisfied: joblib<2,>=1.1.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from imbalanced-learn->imblearn) (1.5.0)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from imbalanced-learn->imblearn) (3.6.0)
Using cached imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Using cached imbalanced_learn-0.13.0-py3-none-any.whl (238 kB)
Using cached sklearn_compatible-0.1.3-py3-none-any.whl (18 kB)
Installing collected packages: sklearn-compat, imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.13.0 imblearn-0.0 sklearn-compat-0.1.3
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Collecting lightgbm
```

```
  Using cached lightgbm-4.6.0-py3-none-win_amd64.whl.metadata (17 kB)
Requirement already satisfied: numpy>=1.17.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from lightgbm) (2.2.5)
Requirement already satisfied: scipy in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from lightgbm) (1.15.3)
Using cached lightgbm-4.6.0-py3-none-win_amd64.whl (1.5 MB)
Installing collected packages: lightgbm
Successfully installed lightgbm-4.6.0
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
In [ ]: !pip install numpy
!pip install pandas
```

```
Requirement already satisfied: numpy in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (2.2.5)
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Requirement already satisfied: pandas in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (2.2.3)
Requirement already satisfied: numpy>=1.26.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2.2.5)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
In [ ]: !pip install umap-learn
```

```
Requirement already satisfied: umap-learn in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (0.5.7)
Requirement already satisfied: numpy>=1.17 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from umap-learn) (2.2.5)
Requirement already satisfied: scipy>=1.3.1 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from umap-learn) (1.15.3)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from umap-learn) (1.6.1)
Requirement already satisfied: numba>=0.51.2 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from umap-learn) (0.61.2)
Requirement already satisfied: pynndescent>=0.5 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from umap-learn) (0.5.13)
Requirement already satisfied: tqdm in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from umap-learn) (4.67.1)
Requirement already satisfied: llvmlite<0.45,>=0.44.0dev0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from numba>=0.51.2->umap-learn) (0.44.0)
Requirement already satisfied: joblib>=0.11 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from pynndescent>=0.5->umap-learn) (1.5.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from scikit-learn>=0.22->umap-learn) (3.6.0)
Requirement already satisfied: colorama in c:\users\amaan\appdata\local\programs\python\python313\lib\site-packages (from tqdm->umap-learn) (0.4.6)

[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

## Importing all the required libraries

```
In [ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math, warnings, pickle, mlflow, datetime, time, os, mlflow.sklearn, gzip
from scipy import stats
from scipy.stats import ttest_ind, levene, anderson, shapiro, mannwhitneyu, f_oneway, kruskal
from statsmodels.stats.contingency_tables import StratifiedTable
from itertools import combinations, permutations, product
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder, PowerTransformer
from sklearn.model_selection import train_test_split, learning_curve, GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor, NearestNeighbors
from sklearn.covariance import EllipticEnvelope
from sklearn.svm import OneClassSVM
from sklearn.mixture import GaussianMixture
from sklearn.cluster import DBSCAN, KMeans, AgglomerativeClustering
from umap import UMAP
from matplotlib.colors import ListedColormap
from sklearn.metrics import silhouette_score, davies_bouldin_score, fowlkes_mallows_score, adjusted_rand_score, normalized_mutual_info_score, homogeneity_score, completeness_score, accuracy_score
from sklearn.metrics.cluster import pair_confusion_matrix, contingency_matrix
```

c:\Users\amaan\AppData\Local\Programs\Python\Python313\Lib\site-packages\tqdm\auto.py:21: TqdmWarning: IPProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user\_install.html  
from .autonotebook import tqdm as notebook\_tqdm

```
In [ ]: from pandas.api.types import is_datetime64_any_dtype
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree, export_graphviz
from six import StringIO
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, BaggingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.impute import KNNImputer
from sklearn.metrics import accuracy_score, precision_score, recall_score, precision_recall_fscore_support
from sklearn.neural_network import MLPClassifier
```

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier,DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier,BaggingClassifier,VotingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.base import BaseEstimator, ClassifierMixin, clone
pd.set_option('display.max_columns', None)

```

## Common functions used in this python workbook

### Function to display all rows and columns from pandas dataframe

```
In [ ]: def display_all(df):      # For any Dataframe df
         with pd.option_context('display.max_rows',100): # Change number of rows according
             with pd.option_context('display.max_columns',100): # Change number of columns
                 display(df)
```

## Importing the dataset

```
In [ ]: nadp = pd.read_csv("./Network_anomaly_data.csv")
```

```
In [ ]: display_all(nadp.head(5))
```

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urgent	hot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0



```
In [ ]: nadp.columns
```

```
Out[ ]: Index(['duration', 'protocoltype', 'service', 'flag', 'srcbytes', 'dstbytes',
       'land', 'wrongfragment', 'urgent', 'hot', 'numfailedlogins', 'loggedin',
       'numcompromised', 'rootshell', 'suattempted', 'numroot',
       'numfilecreations', 'numshells', 'numaccessfiles', 'numoutboundcmds',
       'ishostlogin', 'isguestlogin', 'count', 'srvcount', 'serrorrate',
       'srvserrorrate', 'rerrorrate', 'srverrorrate', 'samesrvrate',
       'diffsrvrate', 'srvidffhostrate', 'dsthostcount', 'dsthostsrvcount',
       'dsthostsamesrvrate', 'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
       'dsthostsrvdiffhostrate', 'dsthostsserrorrate', 'dsthostsrvserrorrate',
       'dsthoststrerrorrate', 'dsthostsrvrerrorrate', 'attack', 'lastflag'],
      dtype='object')
```

```
In [ ]: len(nadp.columns)
```

```
Out[ ]: 43
```

## Dataset Feature Description

There are 43 features in the given dataset. These 43 features are grouped into 5 different groups. Those 5 groups are

1. Basic Connection Features
2. Content Related Features
3. Time-Related Traffic Features
4. Host-Based Traffic Features
5. Attack-type and LastFlag (Difficulty Level) Features

## Basic Connection Features

S.No	Feature name	Description	Continuous or Discrete
1	duration	Length of time duration of the connection. (seconds)	Continuous
2	protocoltype	Protocol used in the connection.	Discrete
3	service	Destination network service used.	Discrete
4	flag	Status of the connection (Normal or Error).	Discrete
5	srcbytes	Number of data bytes transferred from source to destination in a single connection.	Continuous
6	dstbytes	Number of data bytes transferred from destination to source in a single connection.	Continuous
7	land	Indicator if source and destination IP addresses and port numbers are equal (1 if equal, 0 otherwise).	Discrete
8	wrongfragment	Total number of wrong fragments in this connection.	Continuous
9	urgent	Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated.	Continuous

## Content-Related Features

S.No	Feature name	Description	Continuous or Discrete
10	hot	Number of 'hot' indicators in the content, such as entering a system directory, creating programs, and executing programs.	Continuous
11	numfailedlogins	Count of failed login attempts.	Continuous
12	loggedin	Login status (1 if successfully logged in, 0 otherwise).	Discrete
13	numcompromised	Number of 'compromised' conditions.	Continuous
14	rootshell	Indicator if root shell is obtained (1 if yes, 0 otherwise).	Discrete
15	suattempted	Indicator if 'su root' command is attempted or used (1 if yes, 0 otherwise).	Discrete
16	numroot	Number of 'root' accesses or operations performed as root in the connection.	Continuous
17	numfilecreations	Number of file creation operations in the connection.	Continuous
18	numshells	Number of shell prompts.	Continuous
19	numaccessfiles	Number of operations on access control files.	Continuous
20	numoutboundcmds	Number of outbound commands in an FTP session.	Continuous
21	ishotlogin	Indicator if the login belongs to the 'hot' list, i.e., root or admin (1 if yes, 0 otherwise).	Discrete
22	isguestlogin	Indicator if the login is a 'guest' login (1 if yes, 0 otherwise).	Discrete

## Time-Related Traffic Features

S.No	Feature name	Description	Continuous or Discrete
23	count	Number of connections to the same destination host as the current connection in the past two seconds.	Continuous
24	srvcount	Number of connections to the same service as the current connection in the past two seconds.	Continuous
25	serrorrate	Percentage of connections that have activated the flag s0, s1, s2, or s3, among the connections aggregated in count.	Continuous
26	srvserrorrate	Percentage of connections that have activated the flag s0, s1, s2, or s3, among the connections aggregated in srv_count.	Continuous
27	rerrorrate	Percentage of connections that have activated the flag REJ, among the connections aggregated in count.	Continuous
28	srv_rerrorrate	Percentage of connections that have activated the flag REJ, among the connections aggregated in srv_count.	Continuous
29	samesrvrate	Percentage of connections that were to the same service, among the connections aggregated in count.	Continuous
30	diffsrvrate	Percentage of connections that were to different services, among the connections aggregated in count.	Continuous
31	srvdifffhostrate	Percentage of connections that were to different destination machines, among the connections aggregated in srv_count.	Continuous

### Host-Based Traffic Features

S.No	Feature name	Description	Continuous or Discrete
32	dsthostcount	Number of connections having the same destination host IP address.	Continuous
33	dsthostsrvcount	Number of connections having the same port number.	Continuous
34	dsthostsamesrvrate	Percentage of connections that were to the same service, among the connections aggregated in dst_host_count.	Continuous
35	dsthostdiffsrvrate	Percentage of connections that were to different services, among the connections aggregated in dst_host_count.	Continuous
36	dsthostsamesrcportrate	Percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count.	Continuous
37	dsthostsrvdifffhostrate	Percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count.	Continuous
38	dsthosterrorrate	Percentage of connections that have activated the flag s0, s1, s2, or s3, among the connections aggregated in dst_host_count.	Continuous
39	dsthostsrvserrorrate	Percentage of connections that have activated the flag s0, s1, s2, or s3, among the connections aggregated in dst_host_srv_count.	Continuous

S.No	Feature name	Description	Continuous or Discrete
40	dsthosterrorrate	Percentage of connections that have activated the flag REJ, among the connections aggregated in dst_host_count.	Continuous
41	dsthostsrv_errorrate	Percentage of connections that have activated the flag REJ, among the connections aggregated in dst_host_srv_count.	Continuous

### Attack-types and LastFlag(Difficulty Level) Features

S.No	Feature name	Description	Continuous or Discrete
42	attack	Indicates the type of malicious attack (There are 22 attack types) . Normal records are indicated as normal	Discrete
43	lastflag	The "Difficulty Level" feature, named as LastFlag in the dataset, is generated by evaluating how many of the 21 trained learners were able to correctly predict the label of each record.	Discrete

## CHAPTER 2: EDA

### EXPLORATORY DATA ANALYSIS

#### Shape of the data

```
In [ ]: print(f"Number of rows in the dataset = {nadp.shape[0]}")
print(f"Number of columns in the dataset = {nadp.shape[1]}")
```

Number of rows in the dataset = 125973  
Number of columns in the dataset = 43

#### Datatypes and info of all the attributes

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125973 entries, 0 to 125972
Data columns (total 43 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   duration         125973 non-null   int64  
 1   protocoltype    125973 non-null   object  
 2   service          125973 non-null   object  
 3   flag             125973 non-null   object  
 4   srcbytes         125973 non-null   int64  
 5   dstbytes         125973 non-null   int64  
 6   land             125973 non-null   int64  
 7   wrongfragment   125973 non-null   int64  
 8   urgent            125973 non-null   int64  
 9   hot               125973 non-null   int64  
 10  numfailedlogins 125973 non-null   int64  
 11 loggedin          125973 non-null   int64  
 12  numcompromised   125973 non-null   int64  
 13  rootshell         125973 non-null   int64  
 14  suattempted      125973 non-null   int64  
 15  numroot           125973 non-null   int64  
 16  numfilecreations 125973 non-null   int64  
 17  numshells          125973 non-null   int64  
 18  numaccessfiles   125973 non-null   int64  
 19  numoutboundcmds   125973 non-null   int64  
 20  ishostlogin       125973 non-null   int64  
 21  isguestlogin      125973 non-null   int64  
 22  count              125973 non-null   int64  
 23  srvcount          125973 non-null   int64  
 24  serrorrate        125973 non-null   float64 
 25  srvserrorrate     125973 non-null   float64 
 26  rerrorrate        125973 non-null   float64 
 27  svrerrorrate      125973 non-null   float64 
 28  samesrvrate       125973 non-null   float64 
 29  diffsrvrate       125973 non-null   float64 
 30  srvdifffhostrate 125973 non-null   float64 
 31  dsthostcount      125973 non-null   int64  
 32  dsthostsrvcount   125973 non-null   int64  
 33  dsthostsamesrvrate 125973 non-null   float64 
 34  dsthostdiffsrvrate 125973 non-null   float64 
 35  dsthostsamesrcportrate 125973 non-null   float64 
 36  dsthostsrvdifffhostrate 125973 non-null   float64 
 37  dsthosterrorrate   125973 non-null   float64 
 38  dsthostsrverrorrate 125973 non-null   float64 
 39  dsthosterrorrate   125973 non-null   float64 
 40  dsthostsrverrorrate 125973 non-null   float64 
 41  attack             125973 non-null   object  
 42  lastflag           125973 non-null   int64  
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
```

## Missing value or Null Value Detection

```
In [ ]: # Null value count
nadb.isnull().sum()
```

```
Out[ ]: duration          0
        protocoltype      0
        service           0
        flag              0
        srcbytes          0
        dstbytes          0
        land              0
        wrongfragment     0
        urgent             0
        hot                0
        numfailedlogins   0
       loggedin           0
        numcompromised     0
        rootshell          0
        suattempted        0
        numroot            0
        numfilecreations   0
        numshells          0
        numaccessfiles     0
        numoutboundcmds    0
        ishostlogin         0
        isguestlogin        0
        count              0
        srvcount            0
        serrorrate          0
        svrerrorrate        0
        rerrorrate          0
        svrerrorrate        0
        samesrvrate         0
        diffsrvrate         0
        srvdifffhostrate   0
        dsthostcount        0
        dsthostsrvcount     0
        dsthostsamesrvrate 0
        dsthostdiffsrvrate 0
        dsthostsamesrcportrate 0
        dsthostsrvdiffhostrate 0
        dsthosterrorrate    0
        dsthostsrverrorrate 0
        dsthoststrerrorrate 0
        dsthostsrvrerrorrate 0
        attack              0
        lastflag            0
        dtype: int64
```

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

```
Out[ ]: 0
```

### Observations

There are no null values in the given dataset.

## Descriptive Statistics regarding each column of dataset

```
In [ ]: display_all(nadp.describe())
```

	duration	srcbytes	dstbytes	land	wrongfragment	urge
<b>count</b>	125973.00000	1.259730e+05	1.259730e+05	125973.00000	125973.000000	125973.0000
<b>mean</b>	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.0001
<b>std</b>	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.0143
<b>min</b>	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.0000
<b>25%</b>	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.0000
<b>50%</b>	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.0000
<b>75%</b>	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.0000
<b>max</b>	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.0000

◀ ▶

In [ ]: `nadp.describe(include="object")`

	protcoltype	service	flag	attack
<b>count</b>	125973	125973	125973	125973
<b>unique</b>	3	70	11	23
<b>top</b>	tcp	http	SF	normal
<b>freq</b>	102689	40338	74945	67343

## Observations

There is no primary key in the given dataset. Combination of all the features creates primary key.

Features like duration, wrongfragment, urgent, hot, numfailedlogins, numcompromised, numroot, numfilecreations, numshells and numaccessfiles have mostly filled with zeros. Because of that, These features have zeros in 25th, 50th and 75th percentiles. These are sparse features.

Features like land,loggedin,rootshell,suattempted,ishostlogin, isguestlogin should binary discrete features according to the feature description. But suattempted has max = 2, So we have to convert that feature to binary by replacing all the 2's with 1's in that feature.

Features like protcoltype, service, flag and attack are nominal categorical features.

lastflag features is also ordinal categorical feature. lastflag feature should not be considered for ml modeling as it by\_product of mulitple ml model predictions and it may create bias. It can be used for evaluating the difficulty level of our trained model.

All the rate related feature are having the range 0 to 1.

Crucial Continuous features in the dataset are duration, srcbytes, dstbytes, count, srvcount, dsthostcount, dsthostsrvcount

## Number of unique values in each column of given dataset

In [ ]: `for i in nadp.columns:  
 print(i,":",nadp[i].nunique())`

```
duration : 2981
protocoltype : 3
service : 70
flag : 11
srcbytes : 3341
dstbytes : 9326
land : 2
wrongfragment : 3
urgent : 4
hot : 28
numfailedlogins : 6
loggedin : 2
numcompromised : 88
rootshell : 2
suattempted : 3
numroot : 82
numfilecreations : 35
numshells : 3
numaccessfiles : 10
numoutboundcmds : 1
ishostlogin : 2
isguestlogin : 2
count : 512
srvcount : 509
serrorrate : 89
srvserrorrate : 86
rerrorrate : 82
srvrerrorrate : 62
samesrvrate : 101
diffsrvrate : 95
srvidffhostrate : 60
dsthostcount : 256
dsthostsrvcount : 256
dsthostsamesrvrate : 101
dsthostdiffsrvrate : 101
dsthostsamesrcportrate : 101
dsthostsrvdiffhostrate : 75
dsthosterrorrate : 101
dsthostsrvserrorrate : 100
dsthostrrorrate : 101
dsthostsrvrrorrate : 101
attack : 23
lastflag : 22
```

## Unique values of columns whose nunique <= 70

```
In [ ]: for i in nadp.columns:
    if nadp[i].nunique() <= 70:
        print(i,(nadp[i].unique()),"",sep = "\n")
```

```
protocoltype
['tcp' 'udp' 'icmp']

service
['ftp_data' 'other' 'private' 'http' 'remote_job' 'name' 'netbios_ns'
 'eco_i' 'mtp' 'telnet' 'finger' 'domain_u' 'supdup' 'uucp_path' 'Z39_50'
 'smtp' 'csnet_ns' 'uucp' 'netbios_dgm' 'urp_i' 'auth' 'domain' 'ftp'
 'bgp' 'ldap' 'ecr_i' 'gopher' 'vmnet' 'systat' 'http_443' 'efs' 'whois'
 'imap4' 'iso_tsap' 'echo' 'klogin' 'link' 'sunrpc' 'login' 'kshell'
 'sql_net' 'time' 'hostnames' 'exec' 'ntp_u' 'discard' 'nntp' 'courier'
 'ctf' 'ssh' 'daytime' 'shell' 'netstat' 'pop_3' 'nnsp' 'IRC' 'pop_2'
 'printer' 'tim_i' 'pm_dump' 'red_i' 'netbios_ssn' 'rje' 'X11' 'urh_i'
 'http_8001' 'aol' 'http_2784' 'tftp_u' 'harvest']

flag
['SF' 'S0' 'REJ' 'RSTR' 'SH' 'RSTO' 'S1' 'RSTOS0' 'S3' 'S2' 'OTH']

land
[0 1]

wrongfragment
[0 3 1]

urgent
[0 1 3 2]

hot
[ 0 5 6 4 2 1 28 30 22 24 14 3 15 25 19 18 77 17 11 7 20 12 9 10
 8 21 33 44]

numfailedlogins
[0 2 1 3 4 5]

loggedin
[0 1]

rootshell
[0 1]

suattempted
[0 1 2]

numfilecreations
[ 0 1 8 4 2 15 13 29 19 18 6 14 5 21 17 40 3 20 11 38 23 10 27 25
 12 16 28 26 7 9 33 22 43 36 34]

numshells
[0 1 2]

numaccessfiles
[0 1 2 3 5 4 8 6 7 9]

numoutboundcmds
[0]

ishostlogin
[0 1]

isguestlogin
[0 1]

srvrerrorrate
[0. 1. 0.03 0.1 0.2 0.25 0.08 0.11 0.33 0.43 0.5 0.07 0.14 0.62
 0.09 0.81 0.06 0.17 0.05 0.75 0.8 0.29 0.12 0.04 0.02 0.71 0.79 0.83
 0.56 0.67 0.84 0.4 0.64 0.74 0.38 0.85 0.15 0.69 0.6 0.73 0.76 0.57
 0.86 0.92 0.82 0.18 0.22 0.78 0.77 0.88 0.96 0.7 0.9 0.72 0.01 0.89
 0.55 0.87 0.95 0.13 0.58 0.3 ]

srvdifffhostrate
[0. 0.09 0.43 0.22 0.2 0.18 1. 0.01 0.14 0.05 0.03 0.21 0.11 0.1
 0.33 0.4 0.67 0.29 0.02 0.5 0.17 0.57 0.06 0.13 0.75 0.07 0.27 0.08
 0.56 0.16 0.25 0.88 0.15 0.12 0.23 0.55 0.04 0.3 0.6 0.31 0.19 0.36
 0.38 0.83 0.24 0.8 0.45 0.44 0.28 0.32 0.42 0.26 0.62 0.35 0.71 0.47
 0.37 0.54 0.46 0.41]
```

```

attack
['normal' 'neptune' 'warezclient' 'ipsweep' 'portsweep' 'teardrop' 'nmap'
'satan' 'smurf' 'pod' 'back' 'guess_passwd' 'ftp_write' 'multihop'
'rootkit' 'buffer_overflow' 'imap' 'warezmaster' 'phf' 'land'
'loadmodule' 'spy' 'perl']

lastflag
[20 15 19 21 18 17 16 12 14 11 2 13 10 9 8 7 3 5 1 6 0 4]

```

## Observations

`numoutboundcmds` feature has only one unique value. So that feature has no significance. We can drop that feature.

To get clear understanding, we can categorise the `service`, `flag` and `attack` features.

`protocoltype` has 3 unique features which can be abbreviated as `tcp` - `transmission control protocol`, `udp` - `user datagram protocol`, `icmp` - `internet control message protocol`

## Brief Explanation of Network Protocols

Network protocols are established rules and conventions that govern how data is transmitted and received over a network. They define the format, order of messages, and actions taken in response to various events, ensuring reliable communication between devices. Here are some key points regarding network protocols:

### 1. Purpose of Network Protocols

- **Communication Standards:** Protocols provide a standard way for devices to communicate, ensuring compatibility and interoperability.
- **Data Integrity:** They help ensure that data is transmitted accurately and without corruption.
- **Flow Control:** Protocols manage the rate of data transmission to prevent network congestion.
- **Error Handling:** They define how errors in data transmission are detected and corrected.

### 2. Types of Network Protocols

- **Transmission Control Protocol (TCP):** A connection-oriented protocol that ensures reliable data transfer by establishing a connection between sender and receiver, sequencing packets, and handling retransmissions.
- **User Datagram Protocol (UDP):** A connectionless protocol that allows for faster data transfer without ensuring reliability or order. It's often used in applications where speed is critical, such as video streaming or online gaming.
- **Internet Protocol (IP):** Responsible for addressing and routing packets across networks. It operates at the network layer and is essential for communication over the internet.
- **Hypertext Transfer Protocol (HTTP):** The foundation of data communication on the World Wide Web. It defines how messages are formatted and transmitted, as well as how web servers and browsers should respond to various commands.
- **File Transfer Protocol (FTP):** Used for transferring files between a client and a server on a network. It provides mechanisms for uploading, downloading, and managing files.
- **Simple Mail Transfer Protocol (SMTP):** The standard protocol for sending emails across the internet. It works alongside other protocols, like POP3 or IMAP, for receiving emails.
- **Post Office Protocol (POP) and Internet Message Access Protocol (IMAP):** Used for retrieving emails from a mail server. POP downloads emails and removes them from the

server, while IMAP allows for email management on the server.

### 3. Layers of Network Protocols

Network protocols are often organized into layers, with each layer serving a specific function:

- **Application Layer:** This is where user applications interact with the network, utilizing protocols like HTTP, FTP, and SMTP.
- **Transport Layer:** Protocols like TCP and UDP operate here, managing data transmission and ensuring delivery.
- **Network Layer:** This layer is where IP operates, handling packet forwarding and routing.
- **Data Link Layer:** Responsible for node-to-node data transfer and error detection, it includes protocols like Ethernet.
- **Physical Layer:** This layer deals with the physical transmission of data over the network medium (e.g., cables, radio waves).

### 4. Importance of Network Protocols

- **Interoperability:** Protocols enable devices from different manufacturers to communicate seamlessly.
- **Scalability:** They allow networks to grow and adapt to new technologies and devices.
- **Security:** Many protocols include mechanisms for encrypting data and authenticating users, enhancing network security.

## Service Categories

Using ChatGPT and Networking knowledge, Services are classified into following 8 categories.  
(This categorisation is subjective in nature)

```
In [ ]: service_categories_dict = {
    "Remote Access and Control Services": [
        "telnet", "ssh", "login", "kshell", "klogin", "remote_job", "rje", "shell",
    ],
    "File Transfer and Storage Services": [
        "ftp", "ftp_data", "tftp_u", "uucp", "uucp_path", "pm_dump", "printer"
    ],
    "Web and Internet Services": [
        "http", "http_443", "http_2784", "http_8001", "gopher", "whois", "z39_50", "
    ],
    "Email and Messaging Services": [
        "smtp", "imap4", "pop_2", "pop_3", "IRC", "nntp", "nnsp"
    ],
    "Networking Protocols and Name Services": [
        "domain", "domain_u", "netbios_dgm", "netbios_ns", "netbios_ssn", "ntp_u", "
    ],
    "Database and Directory Services": [
        "ldap", "sql_net"
    ],
    "Error and Diagnostic Services": [
        "echo", "discard", "netstat", "systat"
    ],
    "Miscellaneous and Legacy Services": [
        "aol", "auth", "bgp", "csnet_ns", "daytime", "exec", "finger", "time",
        "tim_i", "urh_i", "urp_i", "vmnet", "sunrpc", "iso_tsap", "ctf",
        "mtp", "link", "harvest", "courier", "X11", "red_i",
        "eco_i", "ecr_i", "other", "private"
    ]
}
```

## Brief Explanation of Flag types

In the context of network traffic analysis, particularly in intrusion detection systems (IDS) and the KDD Cup 1999 dataset, flags are indicators that represent the state of a connection or session. Each flag provides insights into the nature

of the connection, whether it's normal, successful, or indicative of an attack or abnormal behavior. Here's a breakdown of the flags you mentioned:

## 1. SF (Successful Finish)

- **Description:** Indicates a successful completion of a connection.
- **Use:** Commonly seen in normal connection terminations. It signifies that the connection was established and ended without issues.

## 2. S0 (Syn Flood)

- **Description:** Represents a connection that was initiated (SYN packet sent) but never completed the handshake.
- **Use:** Often associated with SYN flood attacks, where an attacker sends a large number of SYN requests without completing the handshake, overwhelming the server.

## 3. REJ (Reject)

- **Description:** Indicates that a connection attempt was rejected.
- **Use:** This flag shows that the request to establish a connection was not allowed, possibly due to firewall rules or other security mechanisms.

## 4. RSTR (Reset)

- **Description:** Signifies that the connection was forcibly reset by one of the parties.
- **Use:** This can happen when a session is abruptly terminated or if there's an error in the communication.

## 5. SH (SHUTDOWN)

- **Description:** Indicates that a connection is being shut down.
- **Use:** This flag typically marks the closing of a connection in a controlled manner, allowing for cleanup of resources.

## 6. RSTO (Reset Other)

- **Description:** This flag is used to indicate that a reset is being sent in response to an unexpected packet.
- **Use:** It can signal that a connection was reset because of an unexpected event, often used in responses to invalid requests.

## 7. S1 (Syn-Sent)

- **Description:** Denotes that a SYN packet has been sent and the sender is waiting for a response.
- **Use:** This is part of the connection establishment process in the TCP handshake.

## 8. RSTOS0 (Reset Other/SYN Flood)

- **Description:** Indicates a reset that occurs in the context of an incomplete connection attempt.
- **Use:** Similar to S0, it can signal potential SYN flood activity where a reset is sent in response to numerous incomplete requests.

## 9. S3 (Syn-Received)

- **Description:** Indicates that the server has received a SYN request and has sent a SYN-ACK response back.
- **Use:** Part of the TCP handshake process, showing that the connection is in the middle of establishment.

## 10. S2 (Established)

- **Description:** Indicates that the connection has been fully established.
- **Use:** This flag signifies that both sides of the connection are ready to transmit data.

## 11. OTH (Other)

- **Description:** This flag is used for any connection status that does not fall into the defined categories.
- **Use:** It acts as a catch-all for other, less common states that may occur during network communication.

## Flag Categories

Using ChatGPT and Networking knowledge, Flags are classified into following 5 categories.  
(This categorisation is subjective in nature)

```
In [ ]: flag_categories_dict = {
    "Success Flag": ["SF"],
    "S Flag": ["S0", "S1", "S2", "S3"],
    "R Flag": ["REJ"],
    "Reset Flag": ["RSTR", "RST0", "RSTOS0"],
    "SH&OTH Flag": ["SH", "OTH"]
}
```

## Range of values of all numerical columns

```
In [ ]: for i in nadp.columns:
    if not isinstance(nadp[i][0], str):
        print(f"Maximum of {i}", nadp[i].max())
        print(f"Minimum of {i}", nadp[i].min())
        print()
```

Maximum of duration 42908  
Minimum of duration 0

Maximum of srcbytes 1379963888  
Minimum of srcbytes 0

Maximum of dstbytes 1309937401  
Minimum of dstbytes 0

Maximum of land 1  
Minimum of land 0

Maximum of wrongfragment 3  
Minimum of wrongfragment 0

Maximum of urgent 3  
Minimum of urgent 0

Maximum of hot 77  
Minimum of hot 0

Maximum of numfailedlogins 5  
Minimum of numfailedlogins 0

Maximum of loggedin 1  
Minimum of loggedin 0

Maximum of numcompromised 7479  
Minimum of numcompromised 0

Maximum of rootshell 1  
Minimum of rootshell 0

Maximum of suattempted 2  
Minimum of suattempted 0

Maximum of numroot 7468  
Minimum of numroot 0

Maximum of numfilecreations 43  
Minimum of numfilecreations 0

Maximum of numshells 2  
Minimum of numshells 0

Maximum of numaccessfiles 9  
Minimum of numaccessfiles 0

Maximum of numoutboundcmds 0  
Minimum of numoutboundcmds 0

Maximum of ishostlogin 1  
Minimum of ishostlogin 0

Maximum of isguestlogin 1  
Minimum of isguestlogin 0

Maximum of count 511  
Minimum of count 0

Maximum of srvcount 511  
Minimum of srvcount 0

Maximum of serrorrate 1.0  
Minimum of serrorrate 0.0

Maximum of svrerrorrate 1.0  
Minimum of svrerrorrate 0.0

Maximum of rerrorrate 1.0  
Minimum of rerrorrate 0.0

Maximum of svrerrorrate 1.0  
Minimum of svrerrorrate 0.0

```
Maximum of samesrvrate 1.0
Minimum of samesrvrate 0.0

Maximum of diffsrvrate 1.0
Minimum of diffsrvrate 0.0

Maximum of srvdifffhostrate 1.0
Minimum of srvdifffhostrate 0.0

Maximum of dsthostcount 255
Minimum of dsthostcount 0

Maximum of dsthostsrvcount 255
Minimum of dsthostsrvcount 0

Maximum of dsthostsamesrvrate 1.0
Minimum of dsthostsamesrvrate 0.0

Maximum of dsthostdiffsrvrate 1.0
Minimum of dsthostdiffsrvrate 0.0

Maximum of dsthostsamesrcportrate 1.0
Minimum of dsthostsamesrcportrate 0.0

Maximum of dsthostsrvdifffhostrate 1.0
Minimum of dsthostsrvdifffhostrate 0.0

Maximum of dsthosterrorrate 1.0
Minimum of dsthosterrorrate 0.0

Maximum of dsthostsrvserrorrate 1.0
Minimum of dsthostsrvserrorrate 0.0

Maximum of dsthostrerrorrate 1.0
Minimum of dsthostrerrorrate 0.0

Maximum of dsthostsrvrerrorrate 1.0
Minimum of dsthostsrvrerrorrate 0.0

Maximum of lastflag 21
Minimum of lastflag 0
```

### Observation

After handling the Outliers, Scaling is required to apply because of various ranges.

## Value counts of all columns with nunique <= 70

```
In [ ]: for i in nadp.columns:
    if nadp[i].nunique()<=70:
        print("Value Counts of {}".format(i),end="\n\n")
        print(nadp[i].value_counts(dropna= False),end="\n\n")
```

Value Counts of protocoltype

```
protocoltype
tcp      102689
udp      14993
icmp     8291
Name: count, dtype: int64
```

Value Counts of service

```
service
http      40338
private    21853
domain_u   9043
smtp       7313
ftp_data   6860
...
tftp_u     3
http_8001  2
aol        2
harvest    2
http_2784  1
Name: count, Length: 70, dtype: int64
```

Value Counts of flag

```
flag
SF      74945
S0      34851
REJ     11233
RSTR    2421
RSTO    1562
S1      365
SH      271
S2      127
RSTOS0  103
S3      49
OTH     46
Name: count, dtype: int64
```

Value Counts of land

```
land
0      125948
1      25
Name: count, dtype: int64
```

Value Counts of wrongfragment

```
wrongfragment
0      124883
3      884
1      206
Name: count, dtype: int64
```

Value Counts of urgent

```
urgent
0      125964
1      5
2      3
3      1
Name: count, dtype: int64
```

Value Counts of hot

```
hot
0      123302
2      1037
1      369
28     277
30     256
4      173
6      140
```

```
5      76
24     68
19     57
22     55
3      54
18     45
14     30
20     9
7      5
15     4
11     3
9      2
25     2
44     2
17     1
77     1
12     1
10     1
8      1
21     1
33     1
Name: count, dtype: int64
```

Value Counts of numfailedlogins

```
numfailedlogins
0    125851
1     104
2      9
3      5
4      3
5      1
Name: count, dtype: int64
```

Value Counts ofloggedin

```
loggedin
0    76121
1    49852
Name: count, dtype: int64
```

Value Counts of rootshell

```
rootshell
0    125804
1     169
Name: count, dtype: int64
```

Value Counts of suattempted

```
sualtempted
0    125893
2      59
1      21
Name: count, dtype: int64
```

Value Counts of numfilecreations

```
numfilecreations
0    125686
1     151
2      41
4      13
8      5
15     5
5      5
17     5
3      5
10     5
11     4
12     4
7      4
18     4
40     3
```

```
25      3
14      3
20      3
6       3
26      3
9       2
23      2
13      2
21      1
29      1
19      1
27      1
28      1
16      1
38      1
33      1
22      1
43      1
36      1
34      1
Name: count, dtype: int64
```

Value Counts of numshells

```
numshells
0    125926
1     42
2     5
Name: count, dtype: int64
```

Value Counts of numaccessfiles

```
numaccessfiles
0    125602
1     313
2     29
3      8
5      6
4      5
6      4
8      3
7      2
9      1
Name: count, dtype: int64
```

Value Counts of numoutboundcmds

```
numoutboundcmds
0    125973
Name: count, dtype: int64
```

Value Counts of ishostlogin

```
ishostlogin
0    125972
1      1
Name: count, dtype: int64
```

Value Counts of isguestlogin

```
isguestlogin
0    124786
1    1187
Name: count, dtype: int64
```

Value Counts of srvrerrorrate

```
srvrerrorrate
0.00    109767
1.00    14827
0.50     244
0.33     160
0.25     114
...
```

```
0.55      1  
0.95      1  
0.13      1  
0.58      1  
0.30      1  
Name: count, Length: 62, dtype: int64
```

Value Counts of srvdifffostrate

srvdifffostrate

```
0.00    97574  
1.00    8143  
0.01    2865  
0.50    982  
0.67    975  
0.12    904  
0.33    790  
0.02    771  
0.11    732  
0.25    724  
0.10    721  
0.14    673  
0.08    653  
0.15    623  
0.40    620  
0.09    618  
0.17    617  
0.29    587  
0.20    584  
0.18    581  
0.22    524  
0.06    520  
0.07    519  
0.13    461  
0.05    325  
0.19    246  
0.75    235  
0.27    221  
0.21    218  
0.03    218  
0.16    193  
0.04    187  
0.60    178  
0.43    178  
0.30    170  
0.38    166  
0.23    165  
0.24    85  
0.31    75  
0.36    71  
0.80    60  
0.44    53  
0.57    33  
0.28    28  
0.26    17  
0.45    14  
0.56    12  
0.42    11  
0.71    9  
0.32    8  
0.35    8  
0.83    7  
0.62    7  
0.47    3  
0.37    3  
0.55    2  
0.46    2  
0.54    2  
0.88    1  
0.41    1
```

Name: count, dtype: int64

Value Counts of attack

```
attack
normal          67343
neptune         41214
satan           3633
ipsweep          3599
portsweep        2931
smurf            2646
nmap              1493
back              956
teardrop          892
warezclient       890
pod                201
guess_passwd       53
buffer_overflow      30
warezmaster        20
land               18
imap               11
rootkit             10
loadmodule          9
ftp_write            8
multihop             7
phf                 4
perl                 3
spy                  2
Name: count, dtype: int64
```

Value Counts of lastflag

```
lastflag
21    62557
18    20667
20    19339
19    10284
15    3990
17    3074
16    2393
12    729
14    674
11    641
13    451
10    253
9     194
7     118
8     106
6      96
5      81
4      79
0      66
3      65
1      62
2      54
Name: count, dtype: int64
```

## Observations

- | land, wrongfragment, urgent, hot, numfailedlogins, rootshell, suattempted, numfilecreations, numshells, numaccessfiles, ishostlogin, isguestlogin Features are very high right skewed features because there are greater than 120000 records in these features as zero.
- | `numoutboundcmds` feature can be removed.
- | `attack` type distribution is not balanced. so while splitting the data for test or validation, It is better to use Stratified sampling based on distribution.

## FEATURE ENGINEERING

Creating a deep copy for adding new features

```
In [ ]: nadp_add = nadp.copy(deep=True)
```

## Creating attack\_category, service\_category, flag\_category Features using respective dictionaries

```
In [ ]: # Create a reverse mapping dictionaries for easier Lookup
attack_to_category = {attack: attack_category for attack_category, attacks in attack}
service_to_category = {service: service_category for service_category, services in s}
flag_to_category = {flag: flag_category for flag_category, flags in flag_categories_}

# Function to map flags to categories
def map_attack_to_category(attack):
    return attack_to_category.get(attack, "NAN") # Default to "NAN"
def map_service_to_category(service):
    return service_to_category.get(service, "NAN") # Default to "NAN"
def map_flag_to_category(flag):
    return flag_to_category.get(flag, "NAN") # Default to "NAN"

# Apply the mapping function to create a new feature
nadp_add['attack_category'] = nadp_add['attack'].apply(map_attack_to_category)
nadp_add['service_category'] = nadp_add['service'].apply(map_service_to_category)
nadp_add['flag_category'] = nadp_add['flag'].apply(map_flag_to_category)
nadp_add['attack_or_normal'] = nadp_add['attack'].apply(lambda x: 0 if x == "normal"
```

```
In [ ]: nadp_add[nadp_add["flag_category"] == "NAN"]
```

```
Out[ ]: duration protocoltype service flag srcbytes dstbytes land wrongfragment urgent h
```

## Multiplying rate features with its respective count/srvcount/dsthostcount/dsthostsrvcount features

```
In [ ]: nadp_add['serrors_count'] = nadp['serrorrate']*nadp['count']
nadp_add['rerrors_count'] = nadp['rerrorrate']*nadp['count']

nadp_add['samesrv_count'] = nadp['samesrvrate']*nadp['count']
nadp_add['diffsrv_count'] = nadp['diffsrvrate']*nadp['count']

nadp_add['serrors_srvcount'] = nadp['srvserrorrate']*nadp['srvcount']
nadp_add['rerrors_srvcount'] = nadp['srvrerrorrate']*nadp['srvcount']

nadp_add['srvdifffhost_srvcount'] = nadp['srvdifffhostrate']*nadp['srvcount']
```

```
In [ ]: nadp_add['dsthost_serrors_count'] = nadp['dsthosterrorrate']*nadp['dsthostcount']
nadp_add['dsthost_rerrors_count'] = nadp['dsthostrrorrate']*nadp['dsthostcount']

nadp_add['dsthost_samesrv_count'] = nadp['dsthostsamesrvrate']*nadp['dsthostcount']
nadp_add['dsthost_diffsrv_count'] = nadp['dsthostdiffsrvrate']*nadp['dsthostcount']

nadp_add['dsthost_serrors_srvcount'] = nadp['dsthostsrvserrorrate']*nadp['dsthostsrv']
nadp_add['dsthost_rerrors_srvcount'] = nadp['dsthostsrvrrorrate']*nadp['dsthostsrv']

nadp_add['dsthost_samesrcport_srvcount'] = nadp['dsthostsamesrcportrate']*nadp['dsth']
nadp_add['dsthost_srvidffhost_srvcount'] = nadp['dsthostsrvdifffhostrate']*nadp['dsth']
```

## Remove numoutboundcmds feature

```
In [ ]: nadp_add = nadp_add.drop(["numoutboundcmds"],axis = 1)
```

## Add Data Speed features by Dividing bytes by duration

```
In [ ]: nadp_add['srcbytes/sec'] = nadp_add.apply(
    lambda row: row['srcbytes'] / row['duration'] if row['duration'] != 0 else row['
    axis=1
)
nadp_add['dstbytes/sec'] = nadp_add.apply(
    lambda row: row['dstbytes'] / row['duration'] if row['duration'] != 0 else row['
```

```
    axis=1  
)
```

Modify suattempted such that it is binary

```
In [ ]: nadp_add["suattempted"] = nadp_add["suattempted"].apply(lambda x: 0 if x == 0 else 1)
```

```
In [ ]: display_all(nadp.head())
```

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urgent	h
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0

◀ ▶

```
In [ ]: display_all(nadp_add.head())
```

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urgent	h
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0

◀ ▶

## DATA VISUALISATION USING PYTHON

### Univariate Analysis

```
In [ ]: cont_cols = ['srcbytes', 'dstbytes','srcbytes/sec','dstbytes/sec',
                   'duration', 'wrongfragment', 'urgent', 'hot',
                   'numfailedlogins', 'numcompromised', 'numroot', 'numfilecreations',
                   'numshells', 'numaccessfiles','count', 'srvcount',
                   'serrorrate','rerrorrate', 'samesrvrate','diffsrvrate',
                   'srvserrorrate', 'srverrorrate','srvidffhostrate','srvidffhost_srvcoun
                   'dsthostcount', 'dsthostsrvcount', 'dsthosterrorrate','dsthosterrorra
                   'dsthostsamesrvrate', 'dsthostdiffsrvrate', 'dsthostsrvserrorrate', 'dst
                   'dsthostsamesrcportrate', 'dsthostsrvdiffhostrate','serrors_count', 're
                   'samesrv_count','diffsrv_count', 'serrors_srvcount', 'rerrors_srvcount'
                   'dsthost_serrors_count','dsthost_errors_count','dsthost_samesrv_count'
                   'dsthost_serrors_srvcount','dsthost_errors_srvcount','dsthost_samesrcp
```

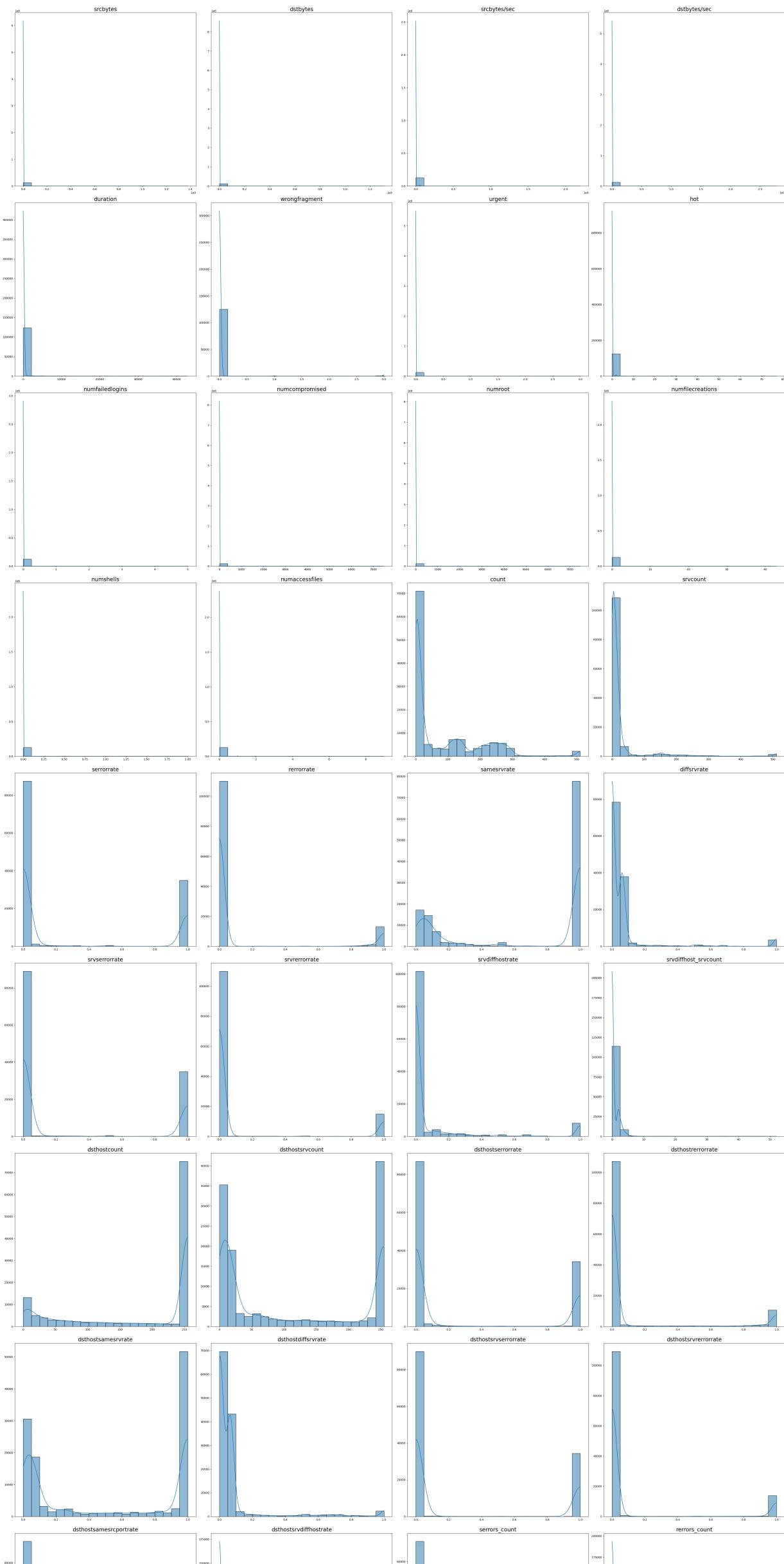
```
cat_cols = ['protocoltype', 'service', 'flag','land','loggedin', 'rootshell',
            'suattempted','ishostlogin', 'isguestlogin','attack', 'lastflag',
            'attack_category', 'service_category', 'flag_category','attack_or_normal']
```

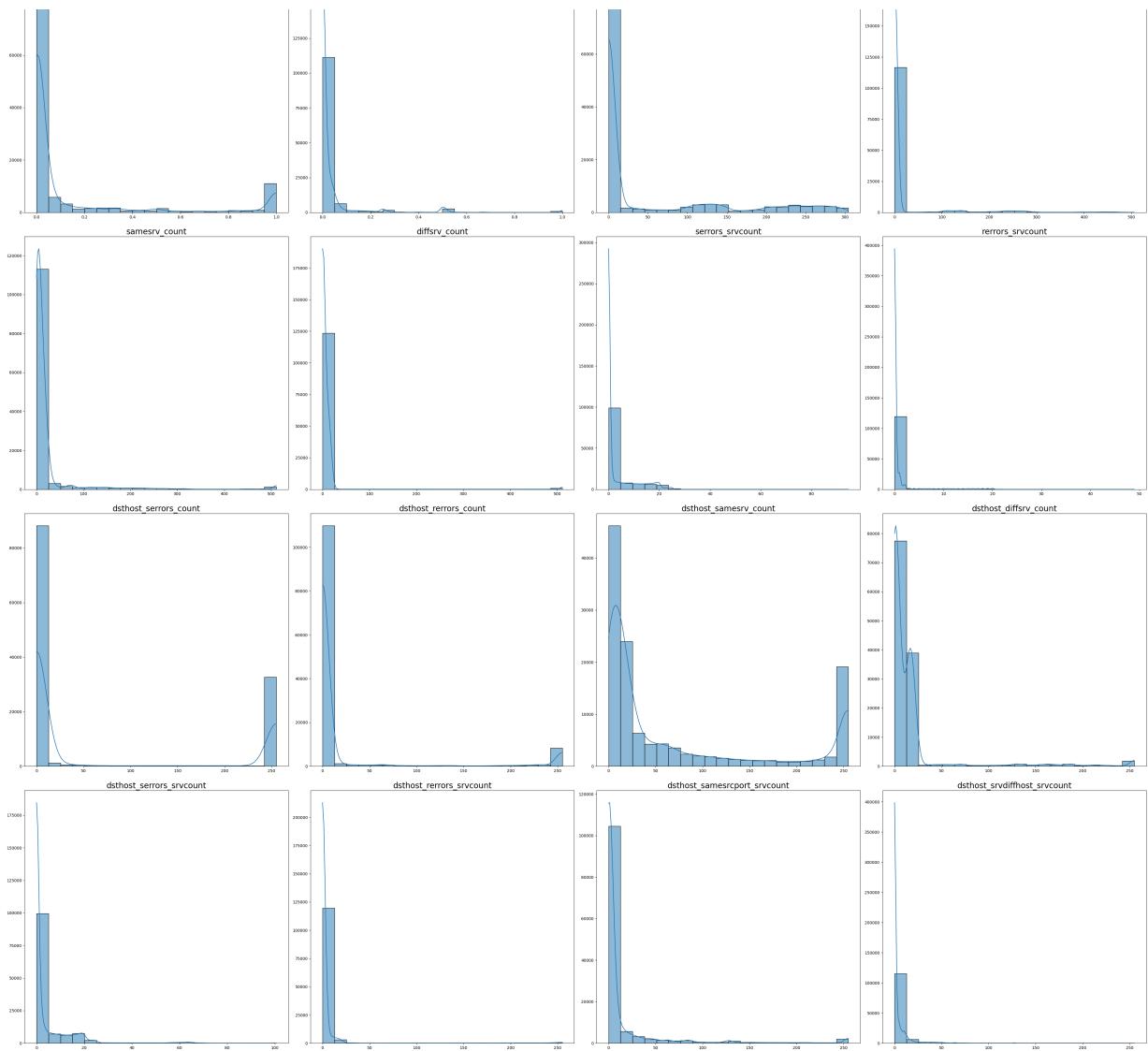
### Distribution plots of all Numerical Columns

```
In [ ]: num_cols = 4 #number of columns
fig = plt.figure(figsize = (num_cols*10,len(cont_cols)*10/num_cols))
plt.suptitle("hist plots for all numerical columns\n",fontsize=24)
```

```
k = 1
for i in cont_cols:
    plt.subplot(math.ceil(len(cont_cols)/num_cols),num_cols,k)
    plt.title("{}".format(i),fontsize = 20)
    k += 1
    plot = sns.histplot(data=nadp_add,x = i,kde = True,bins = 20)
    plt.xlabel("") # No xlabel to keep plots clean
    plt.ylabel("") # No ylabel to keep plots clean
    warnings.filterwarnings('ignore')
plt.tight_layout()
plt.subplots_adjust(top=0.96)
plt.show()
warnings.filterwarnings('ignore')
```

hist plots for all numerical columns





## Observation

Highly Right Skewed Distributions with single significal peak at 0 are - All Basic and Content Related Features

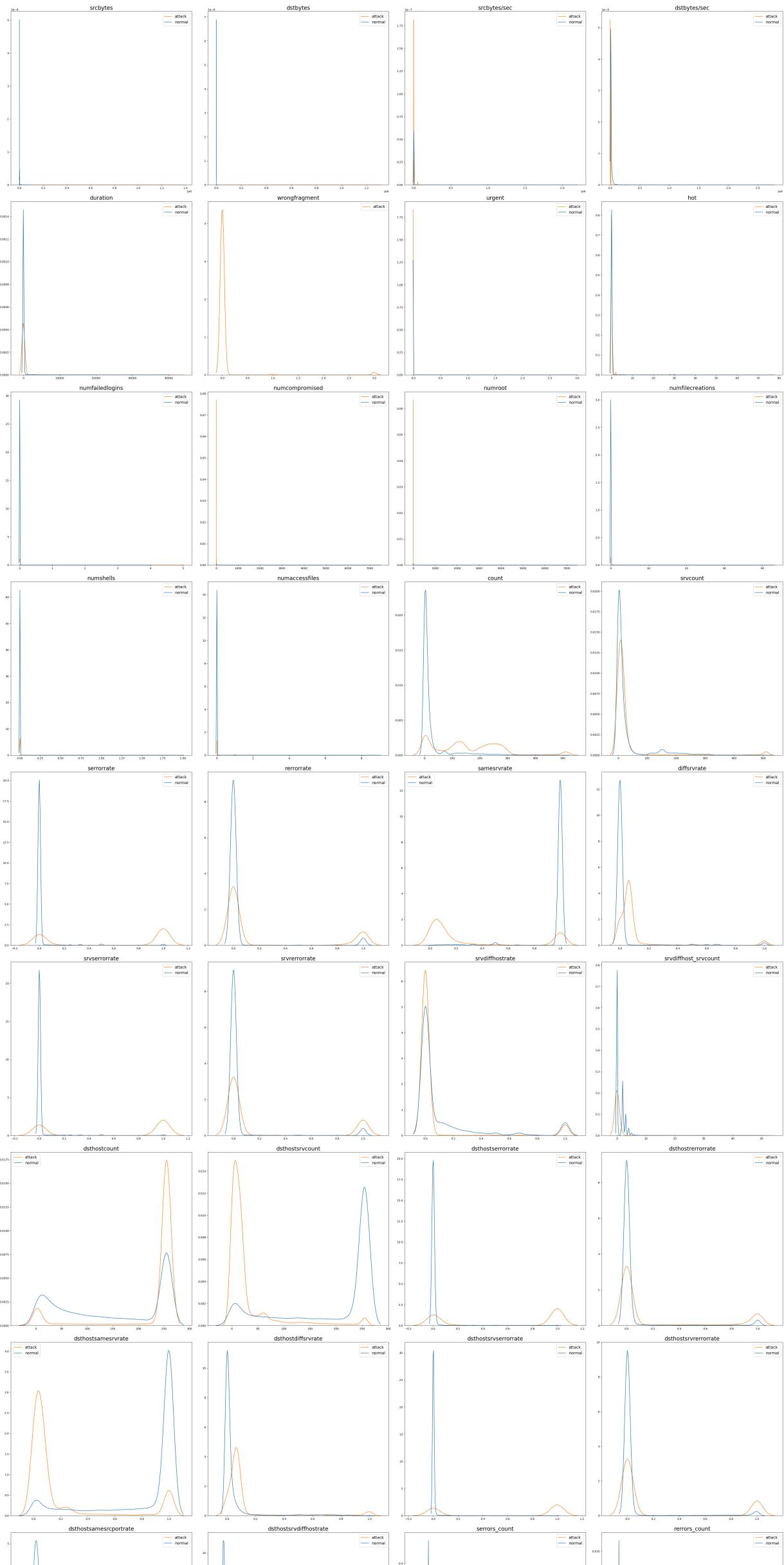
**Count** & **Srvcount** are having slightly better Right Skewed distribution with some dispersion

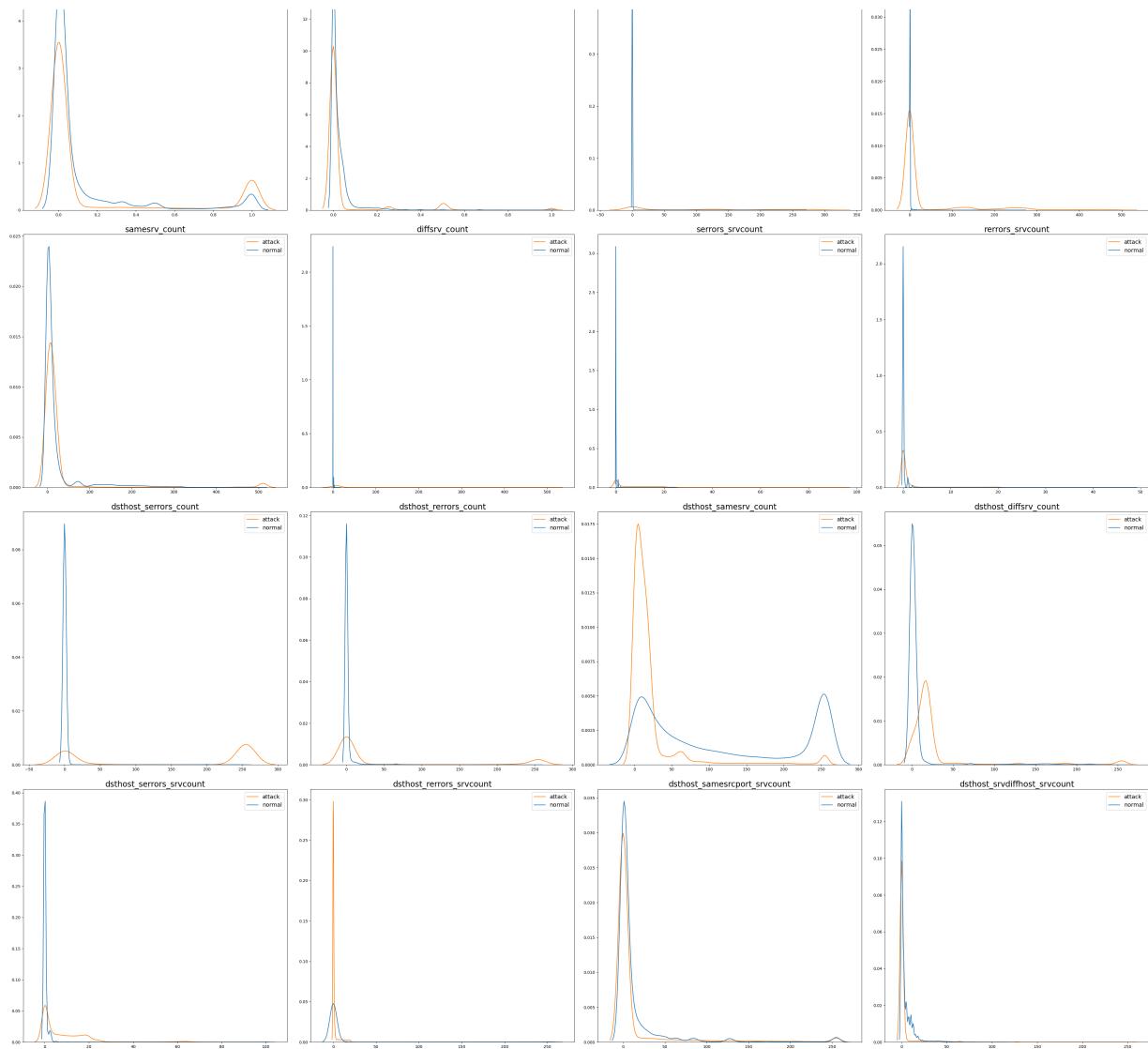
`dsthostcount` & `dsthostsrvcount` are having Left Skewed Distribution with two peaks. One at max & one at min. Similar distribution can be observed in `dsthostsamesrvrate`

Most of the rate related features are having two peaks. One at min and one at max.

```
In [ ]: num_cols = 4
fig = plt.figure(figsize = (num_cols*10,len(cont_cols)*10/num_cols))
plt.suptitle("kde plots for all numerical columns with attack_or_normal as hue\n",fo
k = 1
for i in cont_cols:
    plt.subplot(math.ceil(len(cont_cols)/num_cols),num_cols,k)
    plt.title("{}".format(i),fontsize = 20)
    k += 1
    plot = sns.kdeplot(data=nadp_add,x = i,hue = "attack_or_normal")
    plt.xlabel("") # No xlabel to keep plots clean
    plt.ylabel("") # No ylabel to keep plots clean
    plt.legend(plot.get_legend_handles_labels,labels = ["attack","normal"],fontsize
    warnings.filterwarnings('ignore')
plt.tight_layout()
plt.subplots_adjust(top=0.96)
plt.show()
warnings.filterwarnings('ignore')
```

kde plots for all numerical columns with attack\_or\_normal as hue





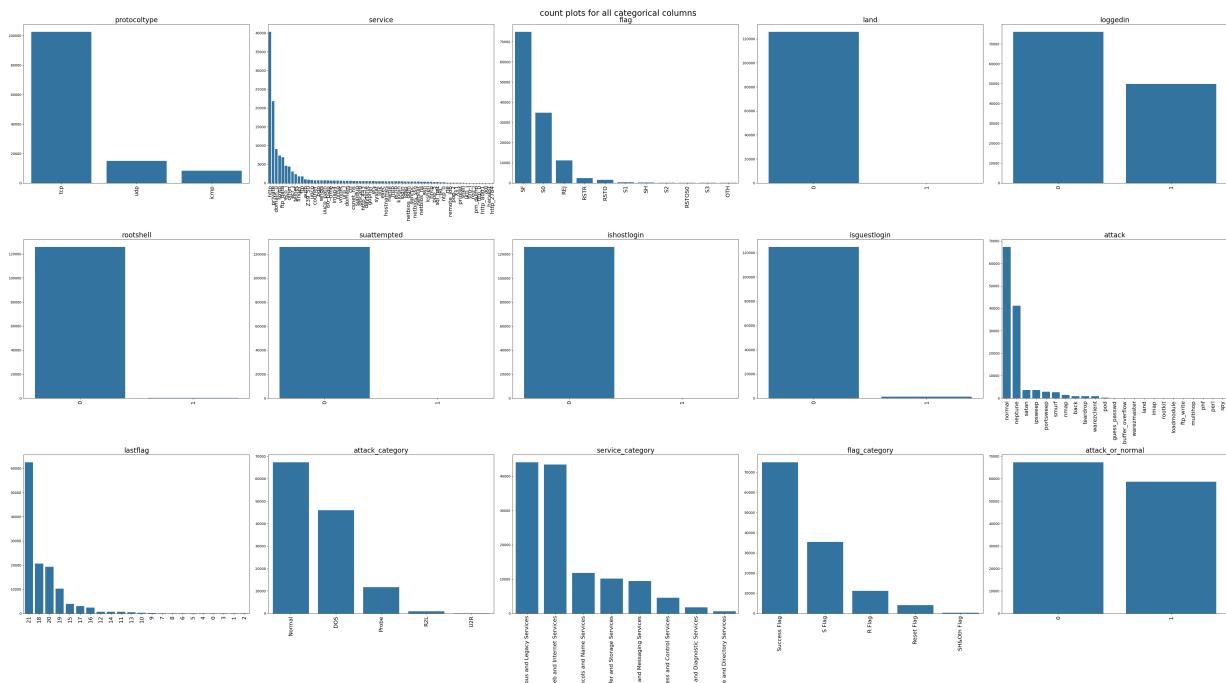
## Observation

**Wrong Fragments** is a very important KPI based on attack vs normal kde distribution. All Postive wrong fragments are from attack categories only.

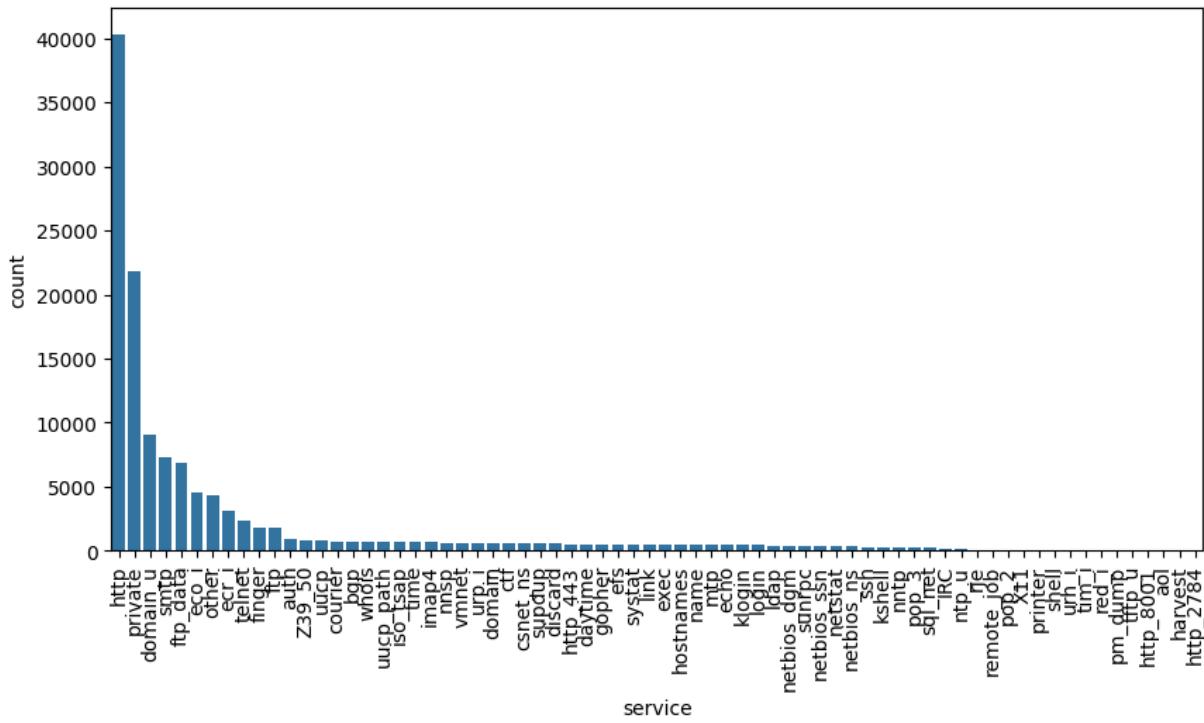
**All Time related and host related features** are having clear distinction between attack and normal. These features may have high feature importance. Among them , **Count, Serrorrate, Rerrorrate, samesrvrate, diffsrvrate, svrerrorrate, srvrerrorrate, dsthostcount, dsthostsrvcount, dsthosterrorrate, dsthosterrorrate, dsthost\_samesrv\_count, dsthost\_diffsvr\_count** are having significant difference between attack and normal.

## Count plots of all categorical columns

```
In [ ]: num_cols = 5
fig = plt.figure(figsize = (num_cols*10,len(cat_cols)*10/num_cols))
plt.suptitle("count plots for all categorical columns\n", fontsize=24)
k = 1
for i in cat_cols:
    plt.subplot(math.ceil(len(cat_cols)/num_cols),num_cols,k)
    plt.title("{}\n".format(i), fontsize = 20)
    k += 1
    category_order = nadp_add[i].value_counts().index
    plot = sns.countplot(data=nadp_add,x = i,order=category_order)
    plt.xticks(rotation = 90,fontsize = 16)
    plt.xlabel("") # No xlabel to keep plots clean
    plt.ylabel("") # No ylabel to keep plots clean
    warnings.filterwarnings('ignore')
plt.tight_layout()
plt.subplots_adjust(top=0.96)
plt.show()
warnings.filterwarnings('ignore')
```



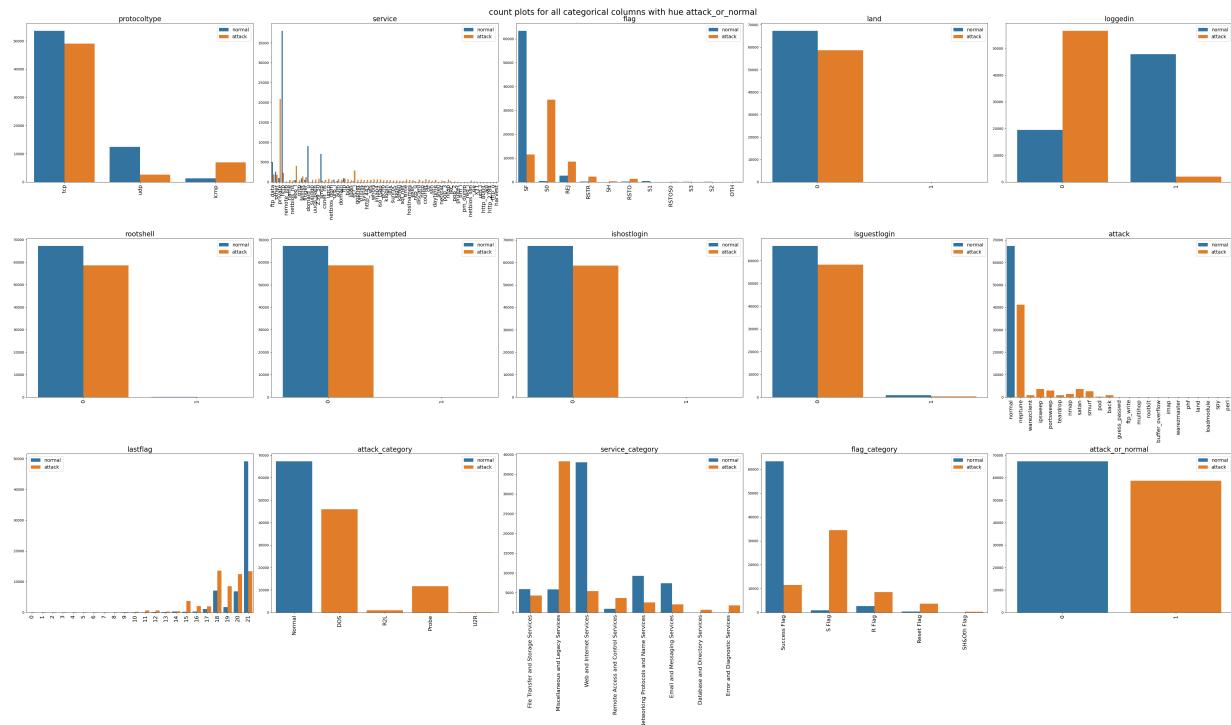
```
In [ ]: fig = plt.figure(figsize=(10,5))
category_order = nadp_add["service"].value_counts().index
sns.countplot(data=nadp_add,x = "service",order=category_order)
plt.xticks(rotation = 90,fontsize = 10)
plt.show()
```



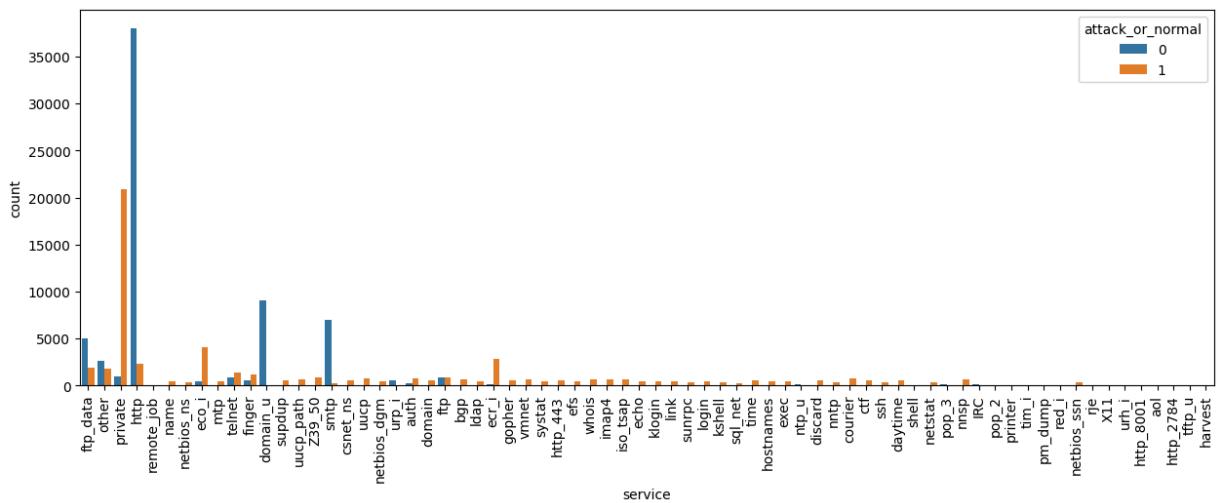
## Observation

- tcp dominates in prototype
- http and private dominates in service
- SF dominates in flag
- neptune dominates in attack types
- u2r , r21 has very less number of rows. Highly imbalanced in case of 5 class classification
- Miscellaneous and Legacy Services & Web& Internet services dominates in service\_category
- attack\_or\_normal has better balanced dataset. So Binary classification may work better than 5 class multi class classification.

```
In [ ]: num_cols = 5
fig = plt.figure(figsize = (num_cols*10,len(cat_cols)*10/num_cols))
plt.suptitle("count plots for all categorical columns with hue attack_or_normal\n", fontweight='bold')
k = 1
for i in cat_cols:
    plt.subplot(math.ceil(len(cat_cols)/num_cols),num_cols,k)
    plt.title("{}\n".format(i), fontsize = 20)
    k += 1
    plot = sns.countplot(data=nadp_add,x = i,hue = "attack_or_normal")
    plt.legend(plot.get_legend_handles_labels,labels = ["normal","attack"],fontsize=16)
    plt.xticks(rotation = 90,fontsize = 16)
    plt.xlabel("") # No xlabel to keep plots clean
    plt.ylabel("") # No ylabel to keep plots clean
    warnings.filterwarnings('ignore')
plt.tight_layout()
plt.subplots_adjust(top=0.96)
plt.show()
warnings.filterwarnings('ignore')
```



```
In [ ]: fig = plt.figure(figsize=(15,5))
sns.countplot(data=nadp_add,x = "service",hue="attack_or_normal")
plt.xticks(rotation = 90,fontsize = 10)
plt.show()
```



## Observation

attack dominates normal in icmp protocoltype , All Flags except SF , when no login in , private service type and Miscellaneous and legacy Services

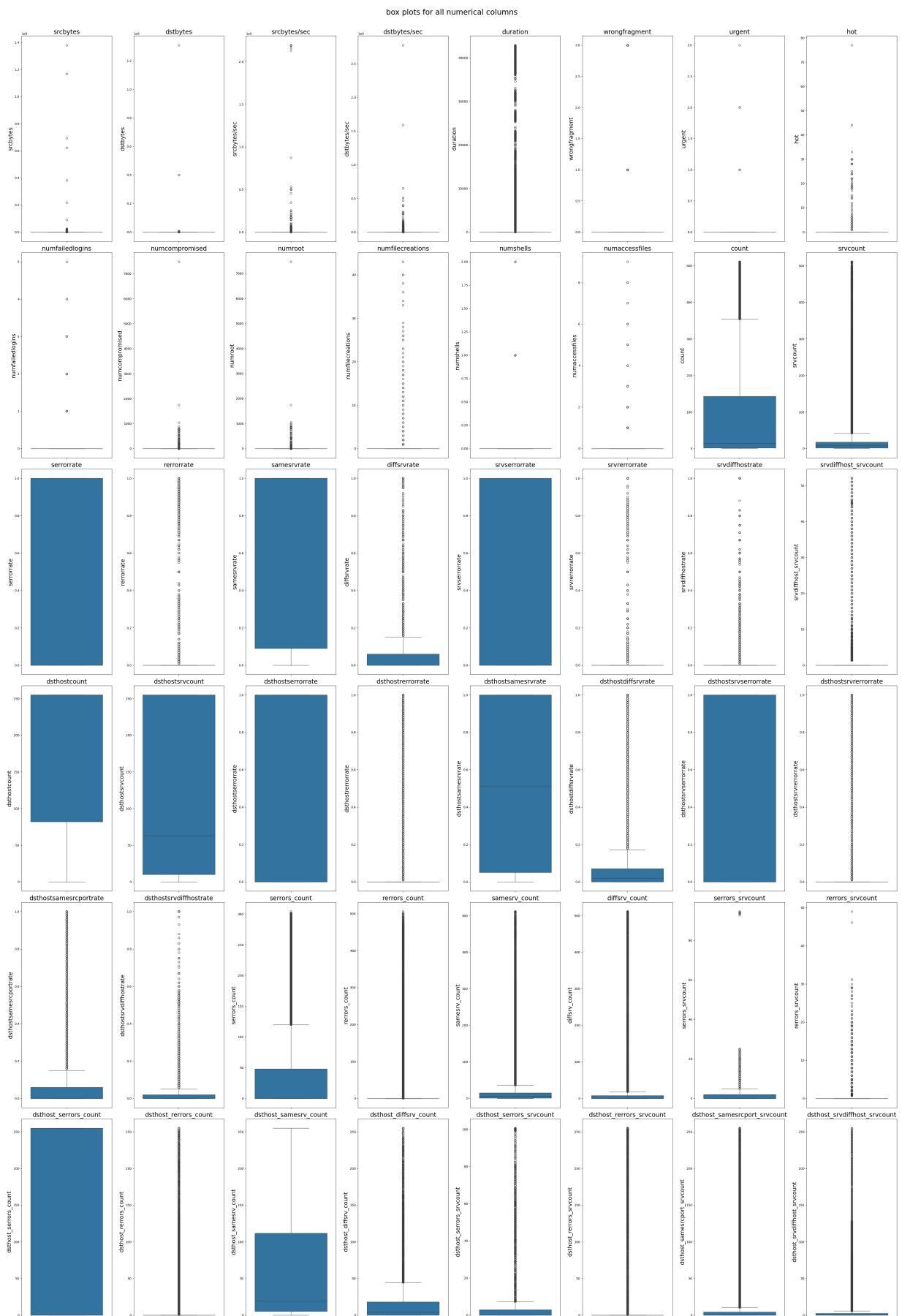
## Outlier Detection

## Box plots of all Numerical columns

```
In [ ]: fig = plt.figure(figsize = (8*5,len(cont_cols)*5/(8/2)))
plt.suptitle("box plots for all numerical columns\n", fontsize=24)
k = 1
for i in cont_cols:
    plt.subplot(math.ceil(len(cont_cols)/8),8,k)
    plt.title("{}\n".format(i), fontsize = 20)
    k += 1
    plot = sns.boxplot(data=nadp_add,y = i)

    # Increase label and legend font sizes
    plot.set_xlabel(plot.get_xlabel(), fontsize=18)
    plot.set_ylabel(plot.get_ylabel(), fontsize=18)

warnings.filterwarnings('ignore')
plt.tight_layout()
plt.subplots_adjust(top=0.96)
plt.show()
warnings.filterwarnings('ignore')
```



## Observation

May be because of lot of zeros in the distribution, Box plots are showing lot of outliers

Lets remove the zeros and plot them again

```
In [ ]: fig = plt.figure(figsize = (8*5,len(cont_cols)*5/(8/2)))
plt.suptitle("box plots for all numerical columns\n", fontsize=24)
k = 1
for i in cont_cols:
    plt.subplot(math.ceil(len(cont_cols)/8),8,k)
    plt.title("{}\n".format(i), fontsize = 20)
    k += 1
    plot = sns.boxplot(data=nadp_add[nadp_add[i] != 0],y = i)

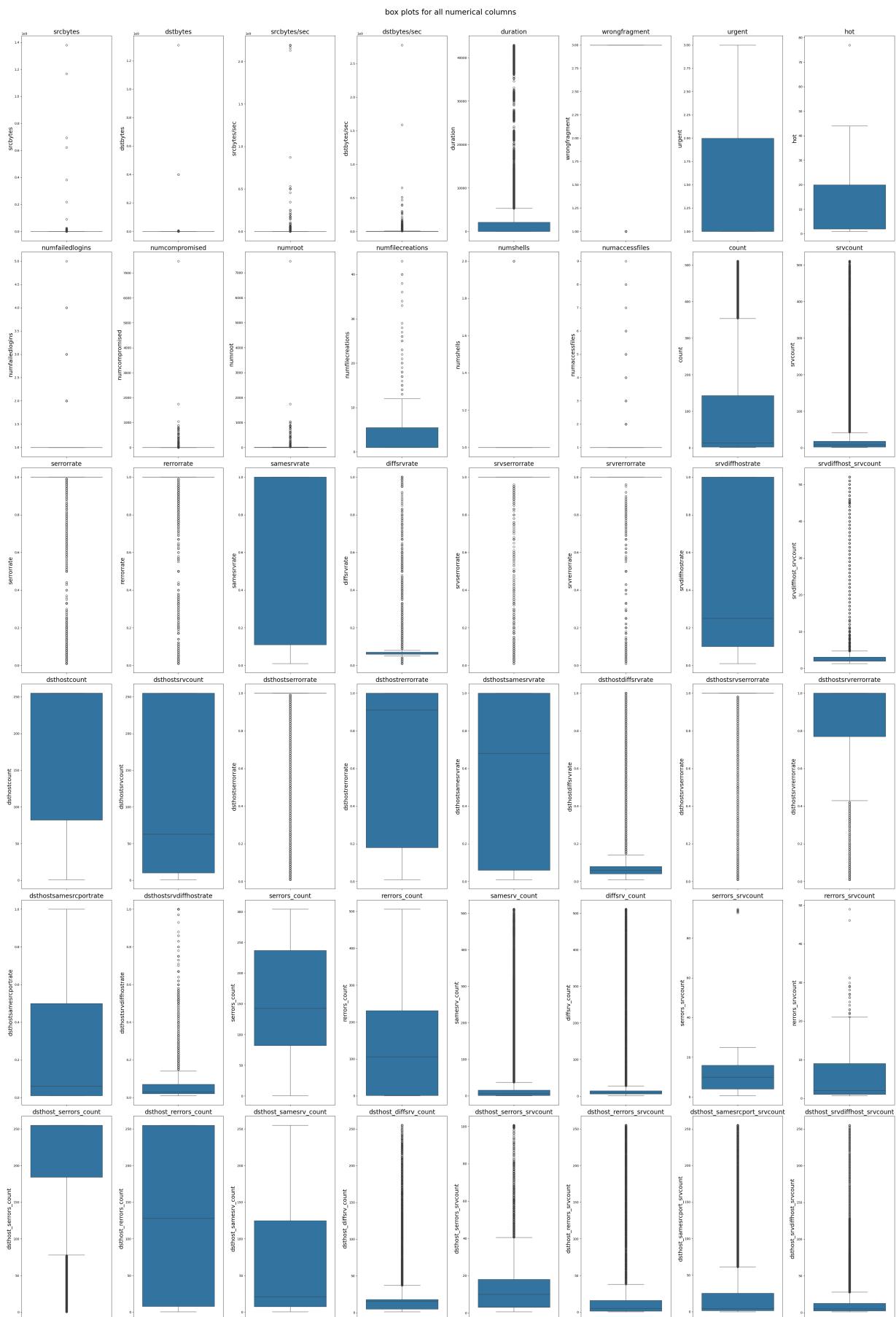
    # Increase Label and Legend font sizes
    plot.set_xlabel(plot.get_xlabel(), fontsize=18)
```

```

plot.set_ylabel(plot.get_ylabel(), fontsize=18)

warnings.filterwarnings('ignore')
plt.tight_layout()
plt.subplots_adjust(top=0.96)
plt.show()
warnings.filterwarnings('ignore')

```



## Observation

Even after removing zeros, There are significant number of outliers.

## Boxcox transformation

```

In [ ]: # Initialize transformed DataFrame and outlier count
nadp_boxcox = nadp_add.copy(deep=True)
outlier_counts = pd.Series(0, index=nadp_add.index)

```

```

# Set up the plot figure
fig = plt.figure(figsize=(8 * 5, len(cont_cols) * 5 / (8 / 2)))
plt.suptitle("Box plots for all numerical columns after Box-Cox transformation\n", f
k = 1

for col in cont_cols:
    # Apply Box-Cox transformation
    transformed_data, _ = stats.boxcox(nadp_add[col] + 0.001) # Avoid zero by addin
    nadp_boxcox[col] = transformed_data # Store transformed data in nadp_boxcox

    # Identify outliers using IQR
    Q1 = nadp_boxcox[col].quantile(0.25)
    Q3 = nadp_boxcox[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Flag outliers for this column
    is_outlier = (nadp_boxcox[col] < lower_bound) | (nadp_boxcox[col] > upper_bound)
    outlier_counts += is_outlier.astype(int) # Increment outlier count for each row

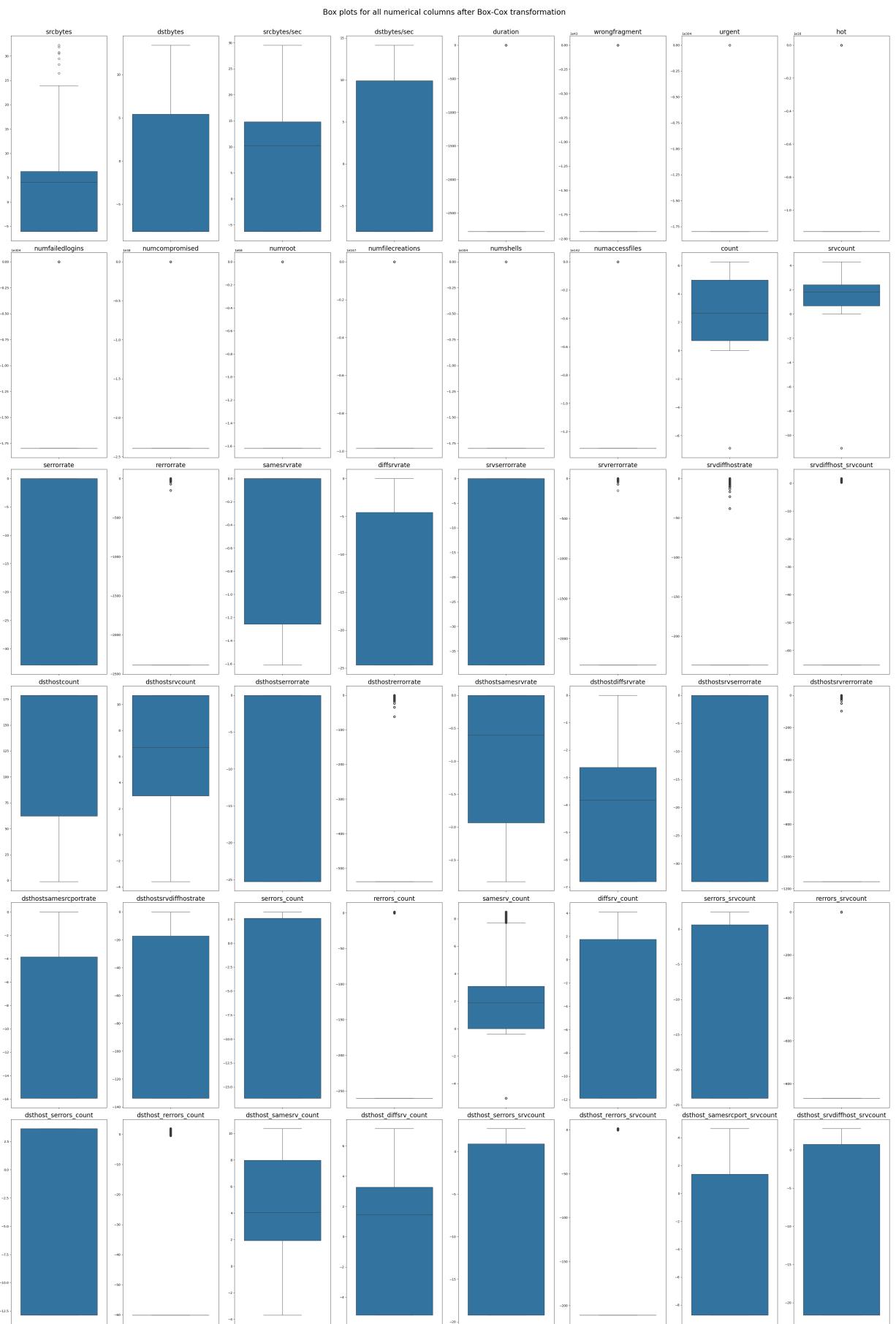
    # Plotting
    plt.subplot(math.ceil(len(cont_cols) / 8), 8, k)
    plt.title(col, fontsize=20)
    k += 1

    # Create the boxplot of transformed data
    sns.boxplot(y=nadp_boxcox[col])
    plt.xlabel("") # No xlabel to keep plots clean
    plt.ylabel("") # No ylabel to keep plots clean

    # Add outlier count as a new feature
    nadp_boxcox["number_of_features_identified_as_outlier"] = outlier_counts

    # Finalize and display plot
    warnings.filterwarnings('ignore')
    plt.tight_layout()
    plt.subplots_adjust(top=0.96)
    plt.show()
    warnings.filterwarnings('ignore')

```



```
In [ ]: display_all(nadp_boxcox.head())
```

	duration	protoptype	service	flag	srcbytes	dstbytes	land	wrongfragment		
0	-2773.840834		tcp	ftp_data	SF	6.987472	-8.145222	0	-1.924712e+43	-1.7
1	-2773.840834		udp	other	SF	5.487203	-8.145222	0	-1.924712e+43	-1.7
2	-2773.840834		tcp	private	S0	-6.074883	-8.145222	0	-1.924712e+43	-1.7
3	-2773.840834		tcp	http	SF	6.051939	7.359132	0	-1.924712e+43	-1.7
4	-2773.840834		tcp	http	SF	5.863742	5.266603	0	-1.924712e+43	-1.7

◀ ▶

```
In [ ]: nadp_boxcox[nadp_boxcox["attack_or_normal"] == 0]["number_of_features_identified_as_
```

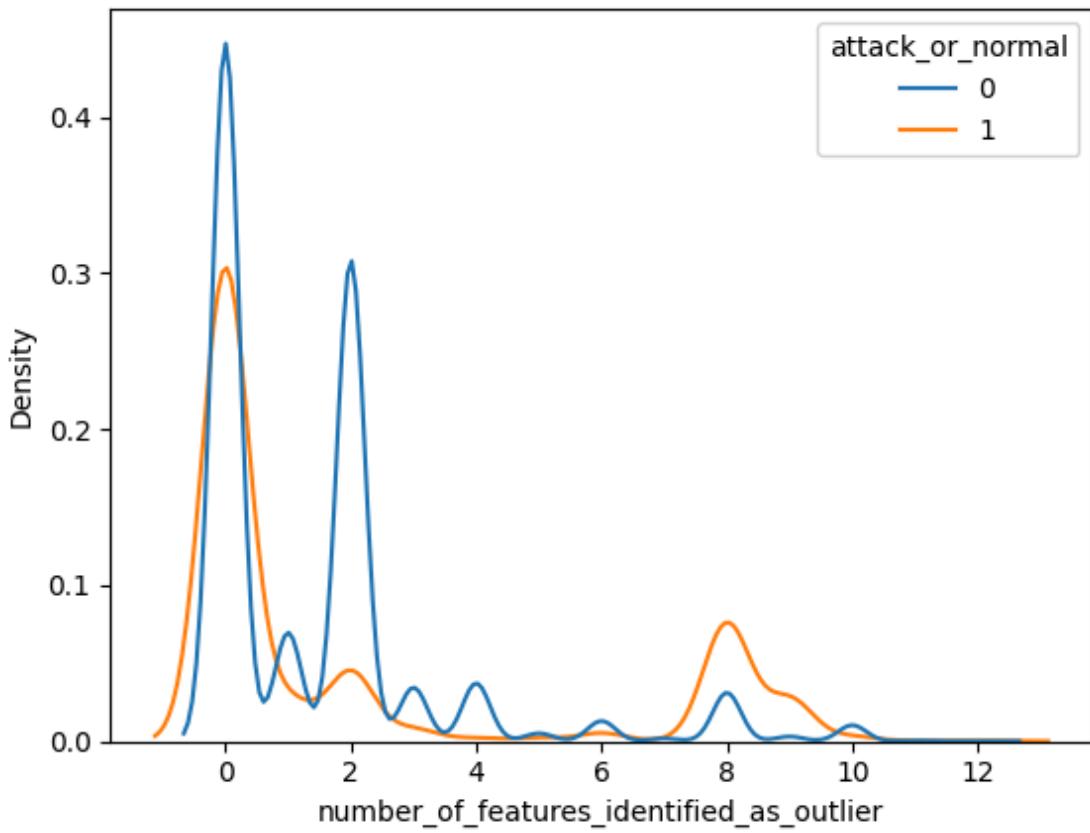
```
Out[ ]: number_of_features_identified_as_outlier
0      31340
2      21609
1      4875
4      2603
3      2409
8      2178
6      905
10     718
5      350
9      206
7      122
11     24
12     4
Name: count, dtype: int64
```

```
In [ ]: nadp_boxcox[nadp_boxcox["attack_or_normal"] == 1]["number_of_features_identified_as_
```

```
Out[ ]: number_of_features_identified_as_outlier
0      35870
8      8928
2      5275
9      3177
1      2925
3      903
6      611
10     353
5      247
4      236
12     43
7      35
11     27
Name: count, dtype: int64
```

```
In [ ]: sns.kdeplot(data=nadp_boxcox,x = "number_of_features_identified_as_outlier",hue = "a
```

```
Out[ ]: <Axes: xlabel='number_of_features_identified_as_outlier', ylabel='Density'>
```



### Observation

We can clearly identify that Boxcox transformation helps to remove the significant number of outliers .

But it is not advisable to remove the outliers. Because Outliers are identified as Anomalies .

We have created nadp\_boxcox dataframe to compare the results with & without boxcox transformation to get an understanding.

`number_of_features_identified_as_outlier` is additional feature in nadp\_boxcox. It represents how many features identified that particular row as outlier.

If we observe top 5 value counts for attack vs normal, For normal --> "0 2 1 4 3", For Attack --> "0 8 2 1 9". It indicates that among the detected outliers (that means ignore 0), Attack category are identified more number of times than normal category. We can observe this kde plot at number 8 and 9 with high peak for attack category.

## Bi-Variate Analysis

### Categorical Vs Categorical

```
In [ ]: num_cols = 3
imp_cat_features = ['protocoltype', 'service_category', 'flag_category', 'attack_cat']
cat_perm = list(permutations(imp_cat_features, 2))

fig = plt.figure(figsize=(num_cols * 8, len(cat_perm) * 8 / num_cols))
plt.suptitle("Stacked hist plots of imp_categorical_features permutation\n", fontsize=16)

for p, q in cat_perm:
    plt.subplot(math.ceil(len(cat_perm) / num_cols), num_cols, k)
    plt.title(f"Cross tab between {p} and {q}\nnormalizing about {q} in percentages")
    k += 1

# Create the cross-tabulated data for plotting
plot = nadp_add.groupby([p])[q].value_counts(normalize=True).mul(100).reset_index()
```

```

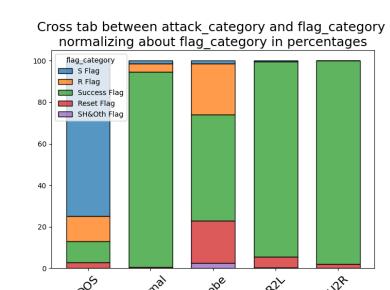
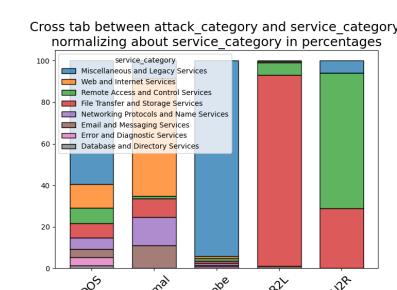
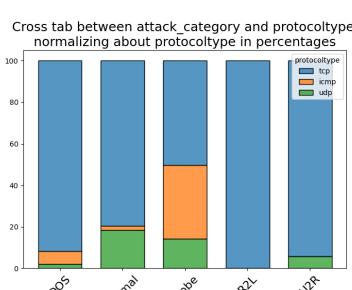
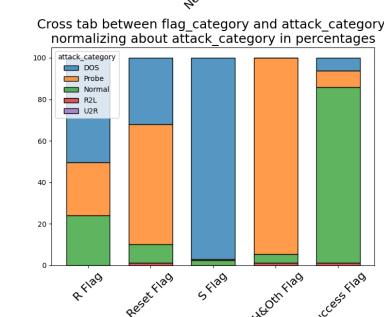
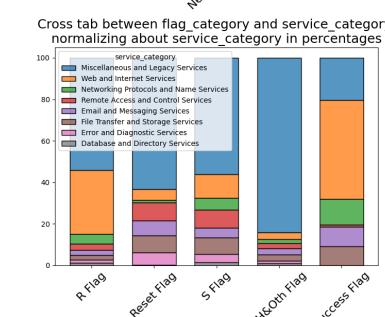
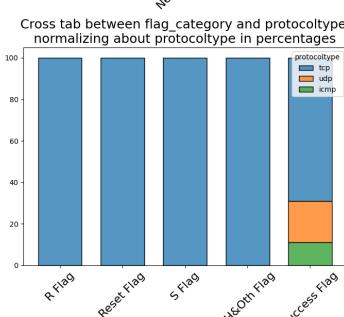
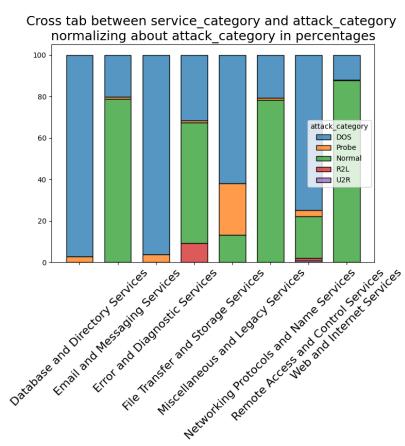
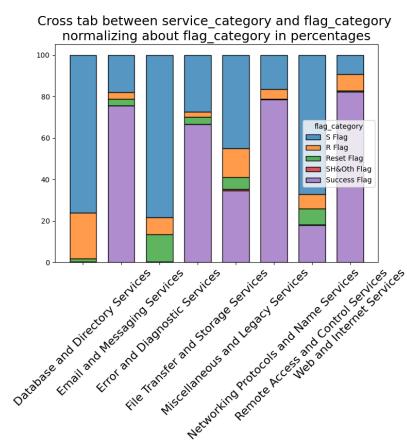
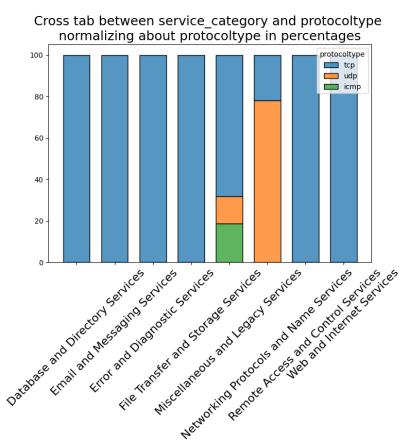
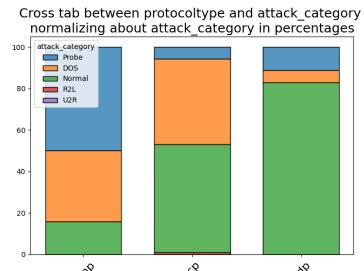
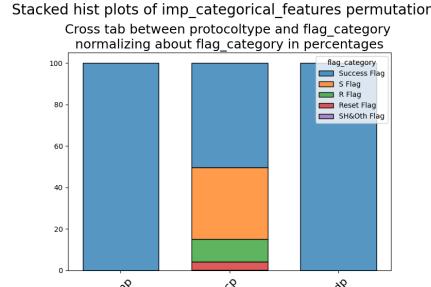
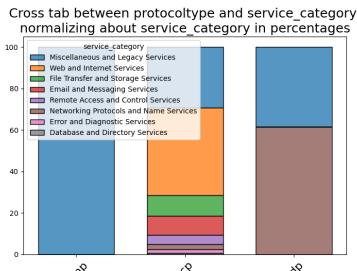
# Plot the histogram with stacked bars and store the axis object
ax = sns.histplot(x=p, hue=q, weights='percentage', multiple='stack', data=plot,
                   palette='magma')

# Set x-axis tick Label font size
plt.xticks(rotation=45, fontsize=16)
plt.xlabel("") # No xlabel to keep plots clean
plt.ylabel("") # No ylabel to keep plots clean

warnings.filterwarnings('ignore')

plt.tight_layout()
plt.subplots_adjust(top=0.95)
plt.show()
warnings.filterwarnings('ignore')

```



## Observation

icmp has dominated by "Miscellaneous and Legacy Services" only.

udp has only two types of services "Miscellaneous and Legacy Services and File Transfer and Storage Services".

icmp has totally "success flag" but it also has mostly 'attacked'.

Error Flag distribution is high in Database and Directory Services, Error and Diagnostic services, Remote access and control services . That means, System able to flag properly in these services.

Database and Directory Services, Error and Diagnostic services has dominated by attack categories only.

Unable to flag properly in udp & icmp protocoltype.

S Flag has mostly DOS attack category. SH&OTH Flag has mostly Probe attack category.

R2L uses only tcp protocol

## Numerical Vs Categorical

### Basic and Content Related features

```
In [ ]: imp_cat_features = ['protocoltype', 'service_category', 'flag_category', 'attack_cat
imp_cont_features = ['duration', 'srcbytes', 'dstbytes', 'wrongfragment', 'urgent', 'hot
                    'numcompromised', 'numroot', 'numfilecreations', 'numshells', 'num
Cat_Vs_cont = []
for i in range(len(imp_cat_features)):
    for j in range(len(imp_cont_features)):
        if (nadp_add[imp_cat_features[i]].nunique()<50):
            Cat_Vs_cont.append((imp_cat_features[i], imp_cont_features[j]))
print(Cat_Vs_cont)
print(len(Cat_Vs_cont))
```

[('protocoltype', 'duration'), ('protocoltype', 'srcbytes'), ('protocoltype', 'dstbyt
es'), ('protocoltype', 'wrongfragment'), ('protocoltype', 'urgent'), ('protocoltype',
'hot'), ('protocoltype', 'numfailedlogins'), ('protocoltype', 'numcompromised'), ('pr
otocoltype', 'numroot'), ('protocoltype', 'numfilecreations'), ('protocoltype', 'num
shells'), ('protocoltype', 'numaccessfiles'), ('service\_category', 'duration'), ('serv
ice\_category', 'srcbytes'), ('service\_category', 'dstbytes'), ('service\_category', 'w
rongfragment'), ('service\_category', 'urgent'), ('service\_category', 'hot'), ('servic
e\_category', 'numfailedlogins'), ('service\_category', 'numcompromised'), ('service\_ca
talog', 'numroot'), ('service\_category', 'numfilecreations'), ('service\_category',
'numshells'), ('service\_category', 'numaccessfiles'), ('flag\_category', 'duration'),
('flag\_category', 'srcbytes'), ('flag\_category', 'dstbytes'), ('flag\_category', 'wron
gfragment'), ('flag\_category', 'urgent'), ('flag\_category', 'hot'), ('flag\_category',
'numfailedlogins'), ('flag\_category', 'numcompromised'), ('flag\_category', 'numroo
t'), ('flag\_category', 'numfilecreations'), ('flag\_category', 'numshells'), ('flag\_ca
talog', 'numaccessfiles'), ('attack\_category', 'duration'), ('attack\_category', 'src
bytes'), ('attack\_category', 'dstbytes'), ('attack\_category', 'wrongfragment'), ('att
ack\_category', 'urgent'), ('attack\_category', 'hot'), ('attack\_category', 'numfailde
logins'), ('attack\_category', 'numcompromised'), ('attack\_category', 'numroot'), ('att
ack\_category', 'numfilecreations'), ('attack\_category', 'numshells'), ('attack\_categ
ory', 'numaccessfiles')]

48

```
In [ ]: num_cols = 4
fig = plt.figure(figsize=(num_cols * 8, len(Cat_Vs_cont) * 8 / num_cols))
plt.suptitle("Barplot of Mean Values of Basic & Content Numerical Features with resp
k = 1

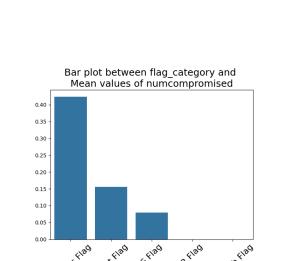
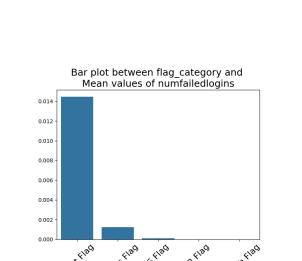
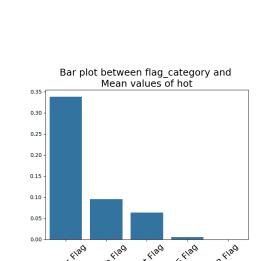
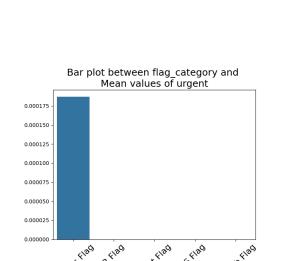
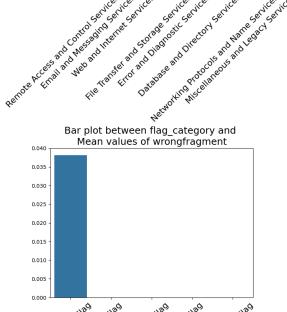
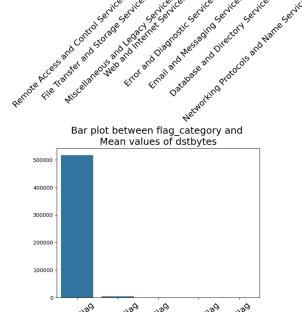
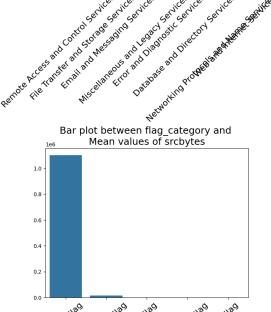
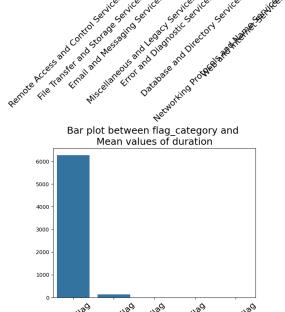
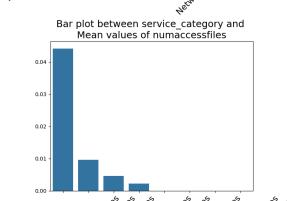
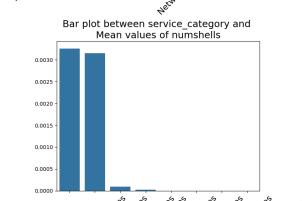
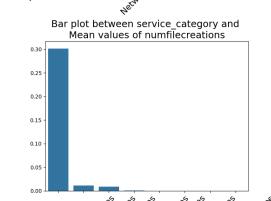
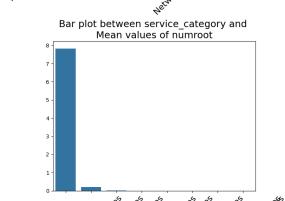
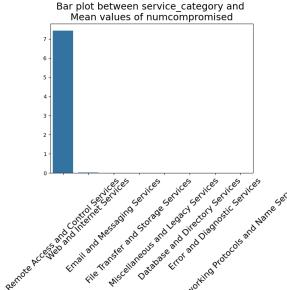
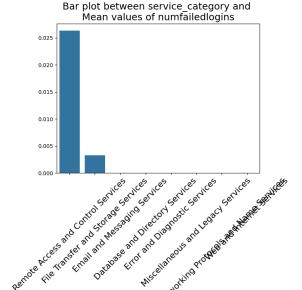
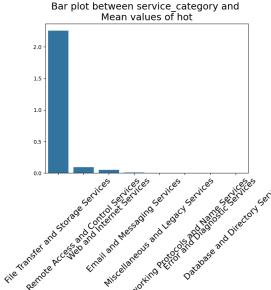
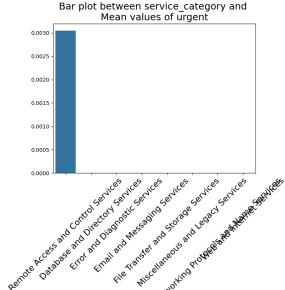
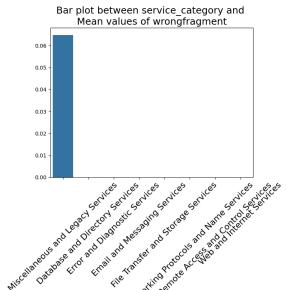
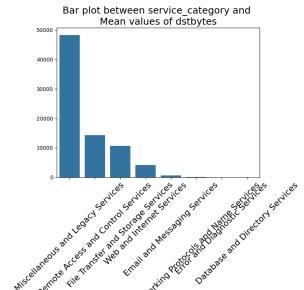
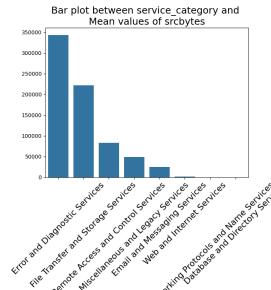
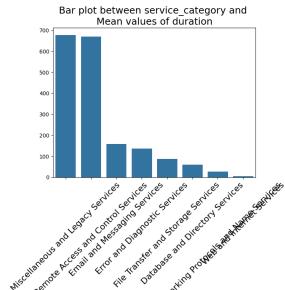
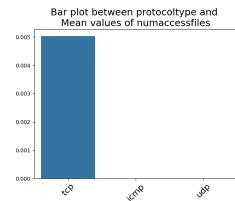
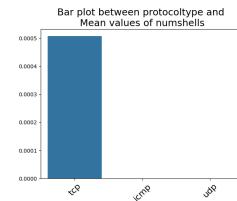
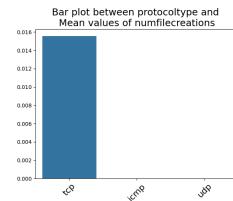
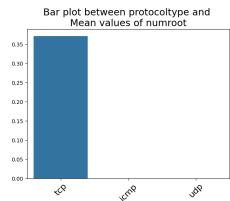
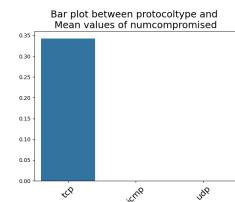
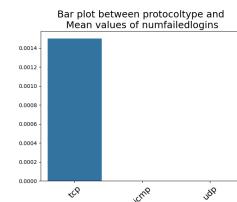
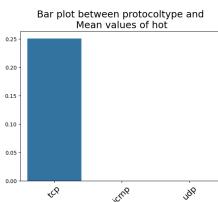
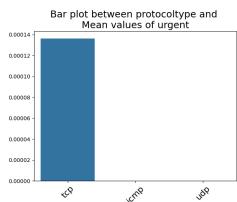
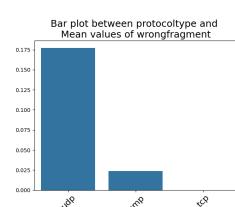
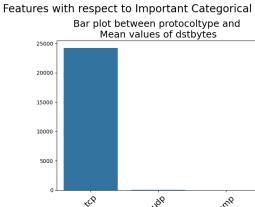
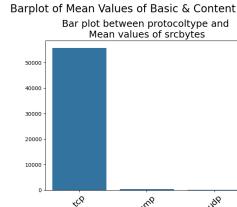
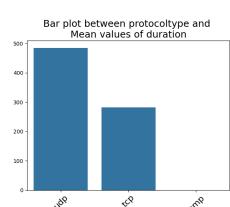
for p, q in Cat_Vs_cont:
    plt.subplot(math.ceil(len(Cat_Vs_cont) / num_cols), num_cols, k)
    plt.title(f"Bar plot between {p} and \nMean values of {q}", fontsize=18)
    k += 1

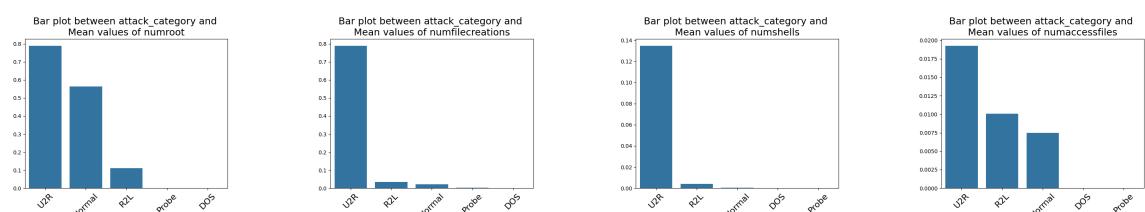
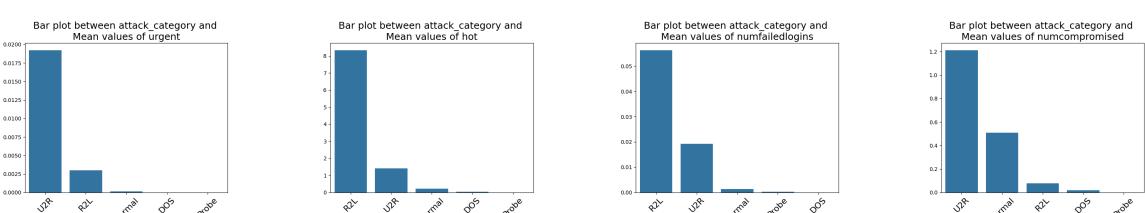
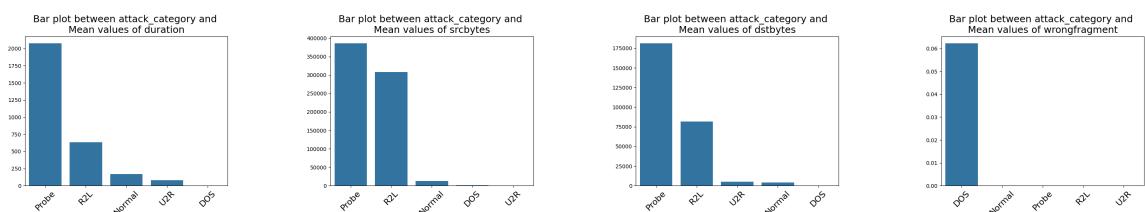
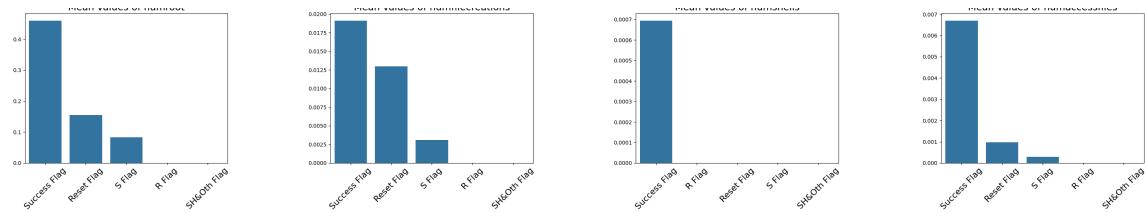
    # Plot the histogram with stacked bars and store the axis object
    df = pd.DataFrame(nadp_add.groupby([p])[q].mean().reset_index())
    sns.barplot(data = df,x = p,y= q,order = df.sort_values(q,ascending = False)[p])

    # Set x-axis tick Label font size
    plt.xticks(rotation=45, fontsize=16)
    plt.xlabel("") # No xlabel to keep plots clean
    plt.ylabel("") # No ylabel to keep plots clean
```

```
warnings.filterwarnings('ignore')

plt.tight_layout()
plt.subplots_adjust(top=0.97)
plt.show()
warnings.filterwarnings('ignore')
```





## Observation

`udp` dominates in `avg duration & avg wrongfragments`. In Remaining all numerical features, `tcp` dominates

`Miscalleneous and legacy Services` dominates in `avg duration, avg dstbytes, avg wrong fragments`. `Error and Diagnostic Services` dominates in `avg srcbytes`. `Remote Access and Control Services` dominates in `urgent packets, numfailedlogins, numcompromised, numroot, numfilecreations, numshells, numaccessfiles`. `File Transfer and Storage Services` dominates in `hot` indicators`.

`Reset Flag` dominates in `avg duration, avg srcbytes, avg dstbytes, avg numfailedlogins`. Remaining all has numerical features are dominated by `Success flag`.

`Probe` dominates in `avg duration, avg srcbytes, avg dstbytes`. `DOS` dominates in `wrong fragmants`. `U2R` dominates in `urgent, numcompromised, numroot, numfilecreations, numshells, numaccessfiles`. `R2L` dominates in `hot, numfailedlogins`.

## Time and Host Related Features

```
In [ ]: imp_cat_features = ['protocoltype', 'service_category', 'flag_category', 'attack_cat'
imp_cont_features2 = ['dsthostcount', 'dsthost_serrors_count','dsthost_reerrors_co
                      'dsthostsrvcount','dsthost_serrors_srvcount','dsthost_reerro
                      'count', 'serrors_count', 'rerrors_count', 'samesrv_count', 's
                      'rvcount','serrors_srvcount', 'rerrors_srvcount','srvidffho
Cat_Vs_cont2 = []
for i in range(len(imp_cat_features)):
    for j in range(len(imp_cont_features2)):
        if (nadb_add[imp_cat_features[i]].nunique()<50):
            Cat_Vs_cont2.append((imp_cat_features[i], imp_cont_features2[j]))
```

```

print(Cat_Vs_cont2)
print(len(Cat_Vs_cont2))

[('protocoltype', 'dsthostcount'), ('protocoltype', 'dsthost_serrors_count'), ('proto
coltype', 'dsthost_rerrors_count'), ('protocoltype', 'dsthost_samesrv_count'), ('proto
coltype', 'dsthost_diffsrv_count'), ('protocoltype', 'dsthostsrvcount'), ('proto
coltype', 'dsthost_serrors_srvcount'), ('protocoltype', 'dsthost_rerrors_srvcount'), ('proto
coltype', 'dsthost_samesrcport_srvcount'), ('protocoltype', 'dsthost_srvdiffhost_s
rvcount'), ('protocoltype', 'count'), ('protocoltype', 'serrors_count'), ('proto
coltype', 'rerrors_count'), ('protocoltype', 'samesrv_count'), ('protocoltype', 'diffsrv_c
ount'), ('protocoltype', 'srvcount'), ('protocoltype', 'serrors_srvcount'), ('proto
coltype', 'rerrors_srvcount'), ('protocoltype', 'srvidfhost_srvcount'), ('service_cate
gory', 'dsthostcount'), ('service_category', 'dsthost_serrors_count'), ('service_cate
gory', 'dsthost_rerrors_count'), ('service_category', 'dsthost_samesrv_count'), ('ser
vice_category', 'dsthost_diffsrv_count'), ('service_category', 'dsthostsrvcount'),
('service_category', 'dsthost_serrors_srvcount'), ('service_category', 'dsthost_rero
rs_srvcount'), ('service_category', 'dsthost_samesrcport_srvcount'), ('service_catego
ry', 'dsthost_srvdiffhost_srvcount'), ('service_category', 'count'), ('service_catego
ry', 'serrors_count'), ('service_category', 'rerrors_count'), ('service_category', 's
amesrv_count'), ('service_category', 'diffsrv_count'), ('service_category', 'srvcoun
t'), ('service_category', 'serrors_srvcount'), ('service_category', 'rerrors_srvcoun
t'), ('service_category', 'srvidfhost_srvcount'), ('flag_category', 'dsthostcount'),
('flag_category', 'dsthost_serrors_count'), ('flag_category', 'dsthost_rerors_coun
t'), ('flag_category', 'dsthost_samesrv_count'), ('flag_category', 'dsthost_diffsrv_c
ount'), ('flag_category', 'dsthostsrvcount'), ('flag_category', 'dsthost_serrors_srvc
ount'), ('flag_category', 'dsthost_rerors_srvcount'), ('flag_category', 'dsthost_sam
esrcport_srvcount'), ('flag_category', 'dsthost_srvdiffhost_srvcount'), ('flag_catego
ry', 'count'), ('flag_category', 'serrors_count'), ('flag_category', 'rerrors_coun
t'), ('flag_category', 'samesrv_count'), ('flag_category', 'diffsrv_count'), ('flag_c
ategory', 'srvcount'), ('flag_category', 'serrors_srvcount'), ('flag_category', 'rerr
ors_srvcount'), ('flag_category', 'srvidfhost_srvcount'), ('attack_category', 'dstho
stcount'), ('attack_category', 'dsthost_serrors_count'), ('attack_category', 'dsthost
_rerors_count'), ('attack_category', 'dsthost_samesrv_count'), ('attack_category',
'dsthost_diffsrv_count'), ('attack_category', 'dsthostsrvcount'), ('attack_catego
ry', 'dsthost_serrors_srvcount'), ('attack_category', 'dsthost_rerors_srvcount'),
('attac
k_category', 'dsthost_samesrcport_srvcount'), ('attack_category', 'dsthost_srvdiffhos
t_srvcount'), ('attack_category', 'count'), ('attack_category', 'serrors_count'), ('a
ttack_category', 'rerrors_count'), ('attack_category', 'samesrv_count'), ('attack_cat
egory', 'diffsrv_count'), ('attack_category', 'srvcount'), ('attack_category', 'serro
rs_srvcount'), ('attack_category', 'rerrors_srvcount'), ('attack_category', 'srvidf
host_srvcount')]

```

76

```

In [ ]: num_cols = 5
fig = plt.figure(figsize=(num_cols * 8, len(Cat_Vs_cont2) * 8 / num_cols))
plt.suptitle("Barplot of Mean Values of Time & Host Related Numerical Features with
k = 1

for p, q in Cat_Vs_cont2:
    plt.subplot(math.ceil(len(Cat_Vs_cont2) / num_cols), num_cols, k)
    plt.title(f"Bar plot between {p} and \nMean values of {q}", fontsize=18)
    k += 1

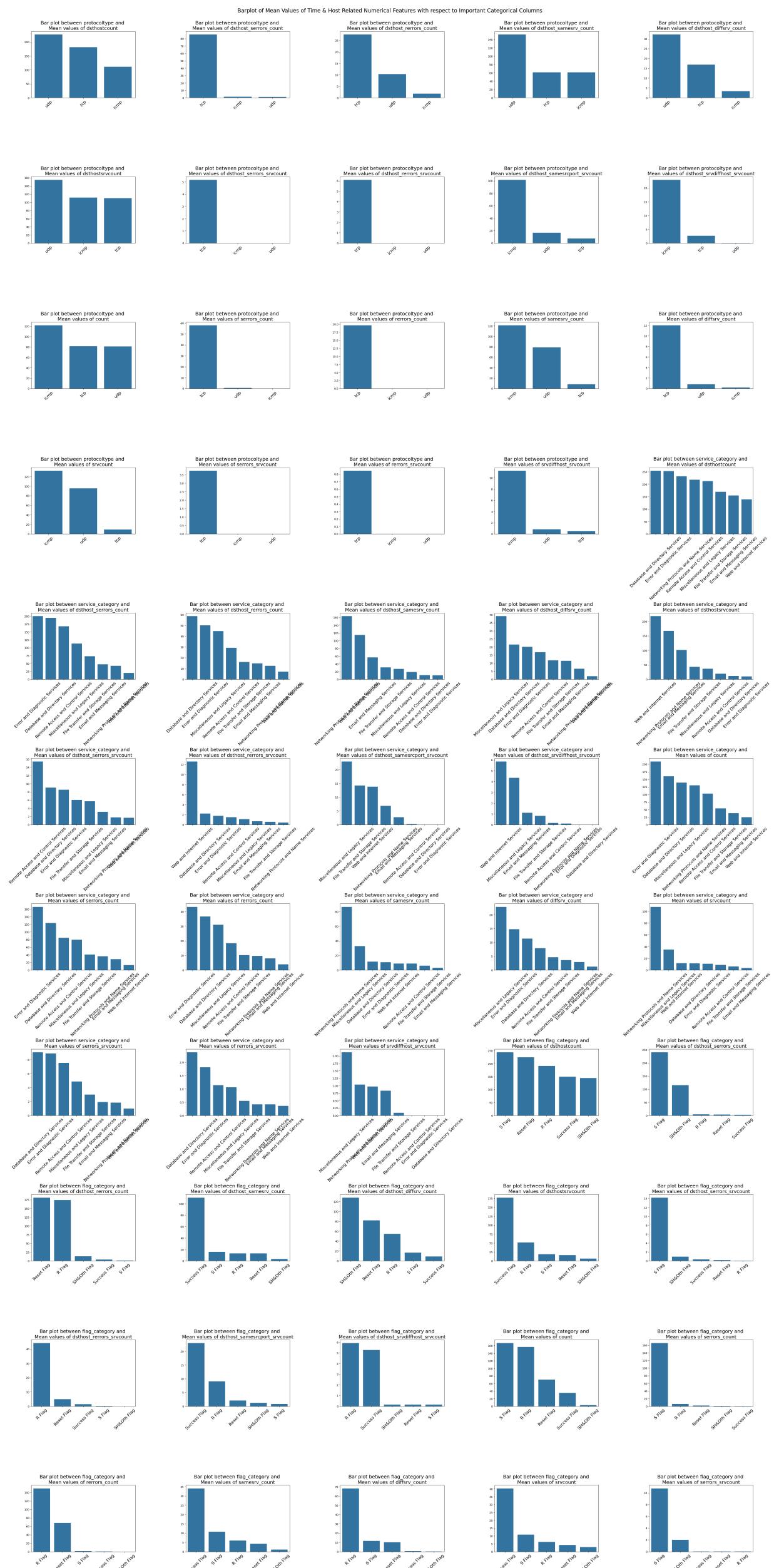
    # Plot the histogram with stacked bars and store the axis object
    df = pd.DataFrame(nadp_add.groupby([p])[q].mean().reset_index())
    sns.barplot(data = df,x = p,y= q,order = df.sort_values(q,ascending = False)[p])

    # Set x-axis tick label font size
    plt.xticks(rotation=45, fontsize=16)
    plt.xlabel("") # No xlabel to keep plots clean
    plt.ylabel("") # No ylabel to keep plots clean

    warnings.filterwarnings('ignore')

plt.tight_layout()
plt.subplots_adjust(top=0.97)
plt.show()
warnings.filterwarnings('ignore')

```





## Observation

udp dominates in dsthostcount & dsthostsrvcount and dsthost\_samesrv\_count, dsthost\_diffhost\_count . icmp dominates in count & srvcnt and dsthost\_samesrvport\_count, dsthost\_srvidffhost\_srvcount,samesrvcount,srvidffhost\_srvcount . tcp dominates in all types of serrors and rerrors and diffsvrcount .

Database and Directory Service dominates in dsthostcount, dsthost\_rerrorcount, serrorcount, rerrorcount . Error and Diagnostic Services dominates in dsthost\_serrorcount, count, serros\_count, rerrorcount . Networking Protocol & Name services dominates in dsthost\_samesrv\_count, samesrv\_count, svrcount . Miscellaneous and Legacy Services dominates in dsthost\_diffsvr\_count, dsthost\_samesrvport\_count, diffsvr\_count, srvidffhost\_srvcount . Web and Internet services dominates in dsthostsrvcount, dsthost\_rerrorcount, dsthost\_srvidffhost\_srvcount .

S Flag dominates in dsthostcount, dsthost\_serrorcount, dsthost\_serrorcount, count, serrorcount , serrorcount . Reset Flag dominates in dsthost\_rerrorcount . R Flag dominates in dsthost\_rerrorcount, dsthost\_srvidffhost\_srvcount, rerrorcount, diffsvr\_count, rerrorcount . SH&Oth Flag dominates in dsthost\_diffsvr\_count . Success Flag dominates in remaining features.

DOS attack type dominates in dsthostcount, dsthost\_serrorcount, dsthost\_serrorcount, count, serrorcount, samesrv\_count,

```
srcvcount, serrors_srcvcount, rerrors_srcvcount . Probe dominates in
dsthost_rerrors_count, dsthost_diffsrv_count,
dsthost_samesrcport_srcvcount, dsthost_srvdiffhost_srcvcount,
rerrors_count, diffsrv_count . Normal Flag dominates in remaining
features.
```

## Multi-Variate Analysis

### Numerical Vs Categorical Vs Target

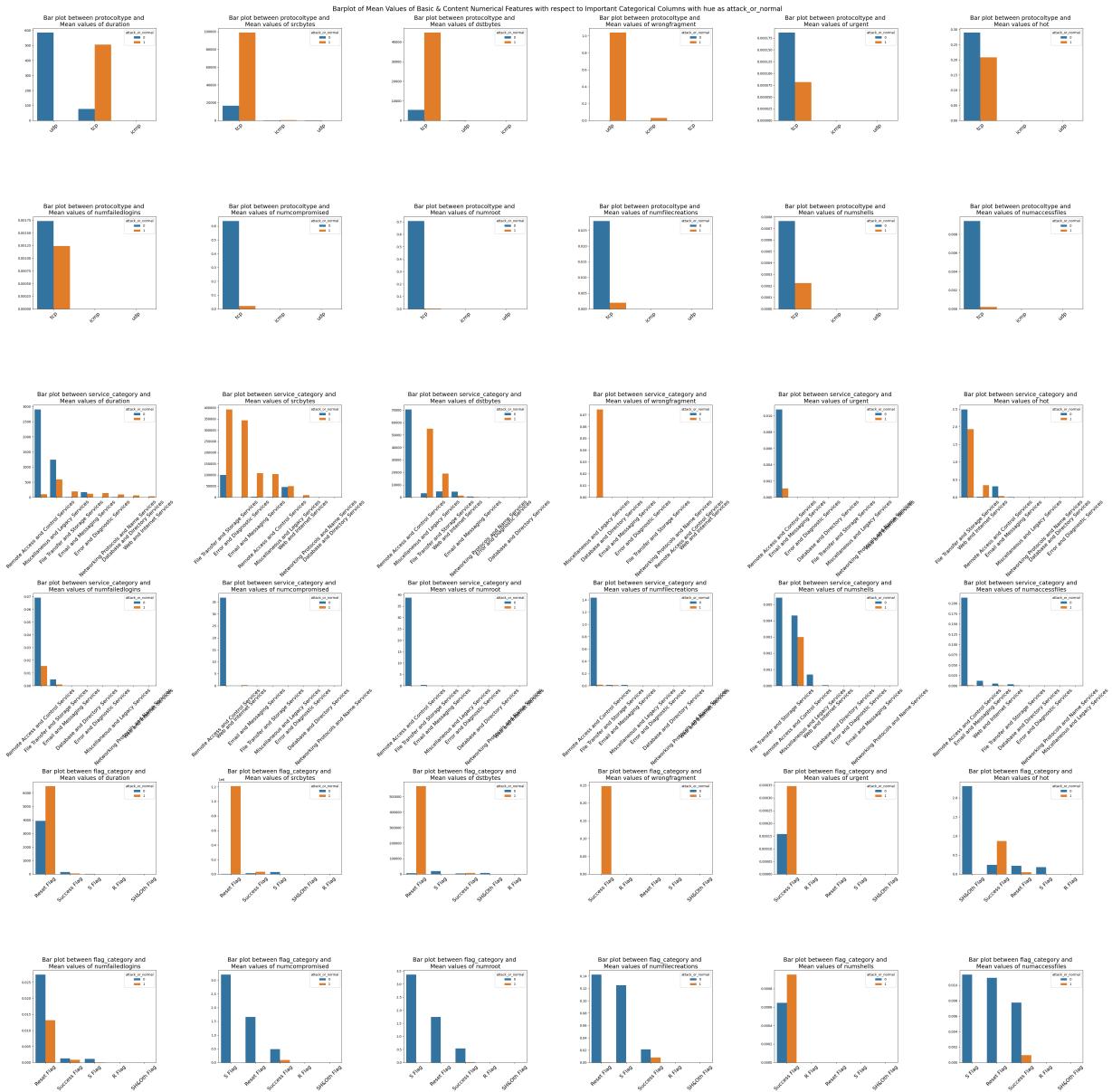
```
In [ ]: num_cols = 6
fig = plt.figure(figsize=(num_cols * 8, len(Cat_Vs_cont[:-12]) * 8 / num_cols))
plt.suptitle("Barplot of Mean Values of Basic & Content Numerical Features with resp
k = 1

for p, q in Cat_Vs_cont[:-12]:
    plt.subplot(math.ceil(len(Cat_Vs_cont[:-12]) / num_cols), num_cols, k)
    plt.title(f"Bar plot between {p} and \nMean values of {q}", fontsize=18)
    k += 1

# Plot the histogram with stacked bars and store the axis object
df = pd.DataFrame(nadp_add.groupby([p, "attack_or_normal"])[[q]].mean().reset_index())
sns.barplot(data = df,x = p,y= q,hue = "attack_or_normal",order = df.sort_values(
    # Set x-axis tick Label font size
    plt.xticks(rotation=45, fontsize=16)
    plt.xlabel("") # No xlabel to keep plots clean
    plt.ylabel("") # No ylabel to keep plots clean

    warnings.filterwarnings('ignore')

plt.tight_layout()
plt.subplots_adjust(top=0.96)
plt.show()
warnings.filterwarnings('ignore')
```



## **Observation**

These bar plots helps to understand the attack distribution too.

Attack is significant due to bar plots related to `srcbytes`, `dstbytes`, `wrongfragment`, `urgent`, `numshells`.

```
In [ ]: num_cols = 5
fig = plt.figure(figsize=(num_cols * 8, len(Cat_Vs_cont2[:-19]) * 8 / num_cols))
plt.suptitle("Barplot of Mean Values of Time & Host Related Numerical Features with k = 1

for p, q in Cat_Vs_cont2[:-19]:
    plt.subplot(math.ceil(len(Cat_Vs_cont2[:-19]) / num_cols), num_cols, k)
    plt.title(f"Bar plot between {p} and \nMean values of {q}", fontsize=18)
    k += 1

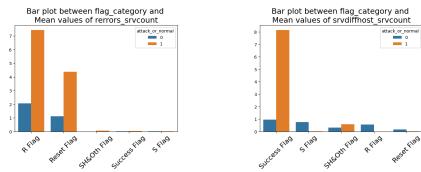
    # Plot the histogram with stacked bars and store the axis object
    df = pd.DataFrame(nadp_add.groupby([p,"attack_or_normal"])[[q]].mean().reset_index())
    sns.barplot(data = df,x = p,y= q,hue="attack_or_normal",order = df.sort_values(q

        # Set x-axis tick label font size
        plt.xticks(rotation=45, fontsize=16)
        plt.xlabel("") # No xlabel to keep plots clean
        plt.ylabel("") # No ylabel to keep plots clean

    warnings.filterwarnings('ignore')

    plt.tight_layout()
    plt.subplots_adjust(top=0.97)
    plt.show()
    warnings.filterwarnings('ignore')
```





## Observation

These bar plots helps to understand the attack distribution too.

Attack is significant in bar plots related to all types of errors and errors counts .

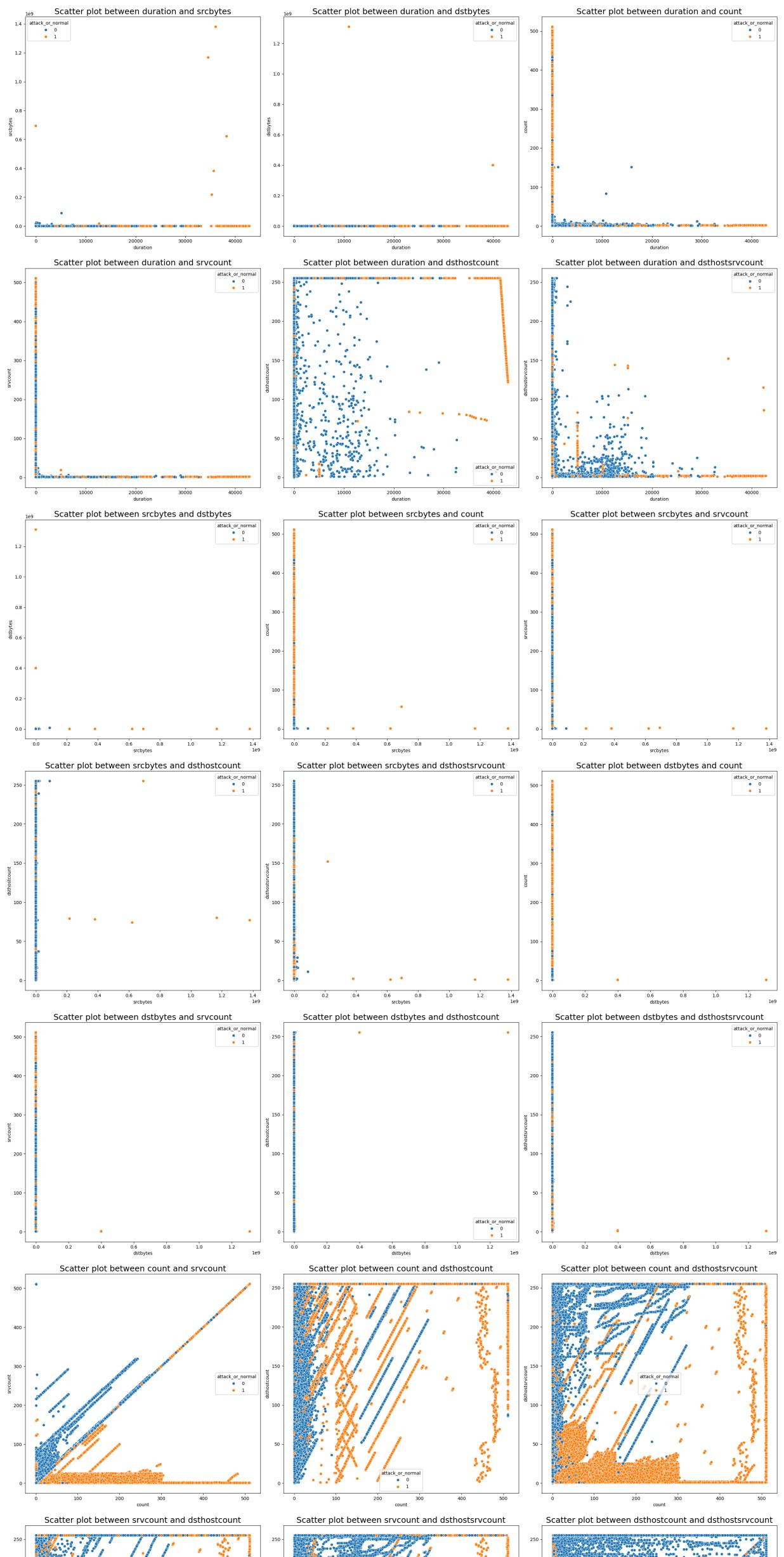
## Numerical Vs Numerical Vs Target

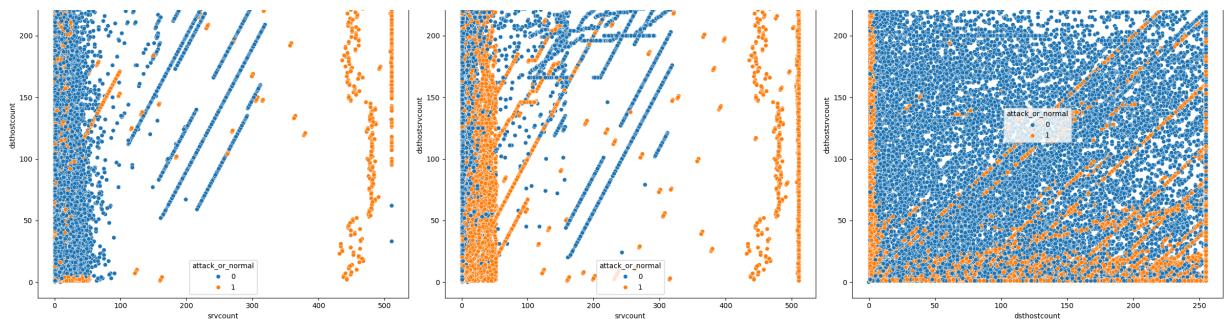
```
In [ ]: imp_cont_cols = ['duration', 'srcbytes', 'dstbytes', 'count', 'srvcount', 'dsthostcount']
cont_comb = list(combinations(imp_cont_cols, 2))
num_cols = 3
fig = plt.figure(figsize=(num_cols * 8, len(cont_comb) * 8 / num_cols))
plt.suptitle("Scatter plot between Important Numerical Features with hue as attack_type")
k = 1

for p, q in cont_comb:
    plt.subplot(math.ceil(len(cont_comb) / num_cols), num_cols, k)
    plt.title(f"Scatter plot between {p} and {q}", fontsize=18)
    k += 1
    sns.scatterplot(data = nadt_add, x=p, y=q, hue = "attack_or_normal")
    warnings.filterwarnings('ignore')

plt.tight_layout()
plt.subplots_adjust(top=0.96)
plt.show()
warnings.filterwarnings('ignore')
```

Scatter plot between Important Numerical Features with hue as attack\_or\_normal





## Observation

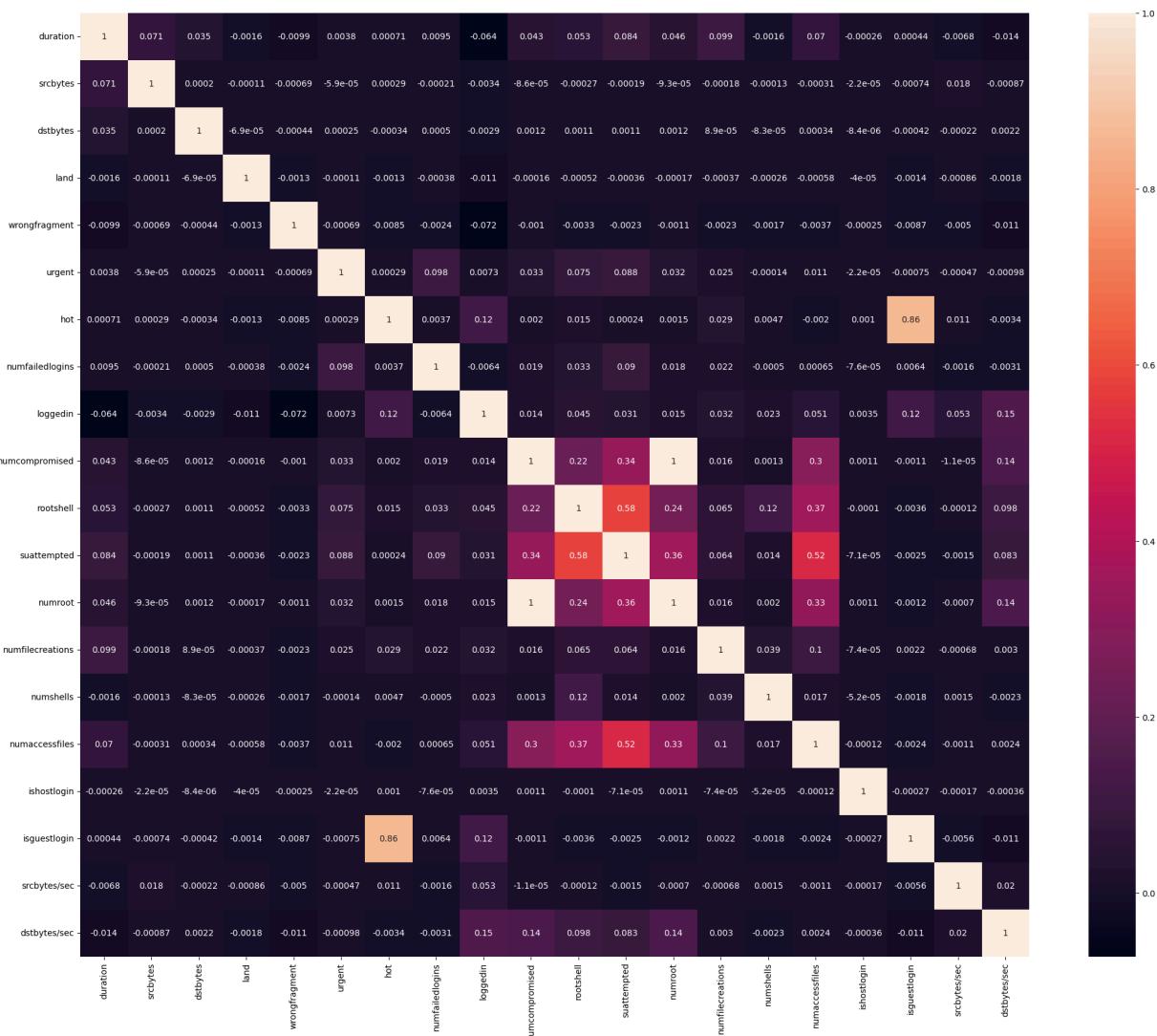
Among all the plots, Clear distinction between attack and normal can be seen in count vs srvcount

## Correlation Heat Maps

### Basic and Content Features Correlation

```
In [ ]: cols = ['duration', 'protocoltype', 'service', 'flag', 'srcbytes', 'dstbytes',
            'land', 'wrongfragment', 'urgent', 'hot', 'numfailedlogins', 'loggedin',
            'numcompromised', 'rootshell', 'suattempted', 'numroot',
            'numfilecreations', 'numshells', 'numaccessfiles', 'ishostlogin',
            'isguestlogin', 'srcbytes/sec', 'dstbytes/sec']

corr = nadp_add[cols].corr(numeric_only=True)
plt.figure(figsize=(25,20))
sns.heatmap(corr, annot=True)
plt.show()
```



## Observation

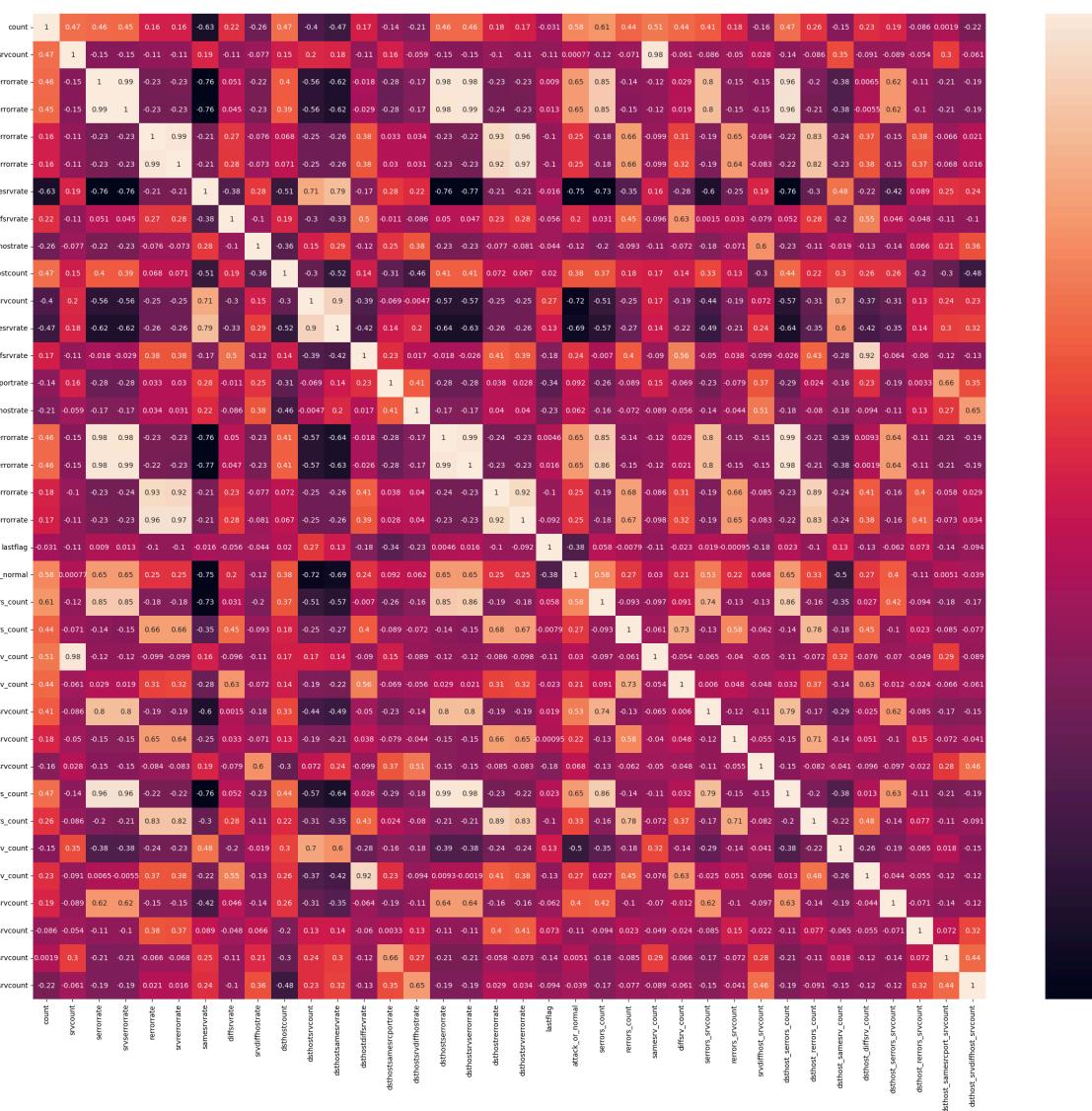
Correlation > 0.8 is observed between isguestlogin & hot

Remaining all combination of features in Basic and content related features has no significant correlation.

## Time and Host Related Features Correlation

```
In [ ]: cols = ['count', 'srvcount', 'serrorrate', 'srvserrorrate',
    'rerrorrate', 'svrerrorrate', 'samesrvrate', 'diffsrvrate',
    'srvdiffhostrate', 'dsthostcount', 'dsthostsrvcount',
    'dsthostsamesrvrate', 'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
    'dsthostsrvdiffhostrate', 'dsthosterrorrate', 'dsthostsrvserrorrate',
    'dsthostrrorrate', 'dsthostsrvrrorrate', 'attack', 'lastflag',
    'attack_category', 'service_category', 'flag_category',
    'attack_or_normal', 'serrors_count', 'rerrors_count', 'samesrv_count',
    'diffsrv_count', 'serrors_srvcount', 'rerrors_srvcount',
    'srvdiffhost_srvcount', 'dsthost_serrors_count',
    'dsthost_rrrors_count', 'dsthost_samesrv_count',
    'dsthost_diffsrv_count', 'dsthost_serrors_srvcount',
    'dsthost_rrrors_srvcount', 'dsthost_samesrcport_srvcount',
    'dsthost_srvdiffhost_srvcount']

corr = nadp_add[cols].corr(numeric_only=True)
plt.figure(figsize=(30,25))
sns.heatmap(corr, annot=True)
plt.show()
```



## Observation

1. srvserrorrate and serrorrate correlation: 0.99
2. svrerrorrate and rerrorrate correlation: 0.99
3. dsthosterrorrate and serrorrate correlation: 0.98
4. dsthosterrorrate and srvserrorrate correlation: 0.98
5. dsthostsrvserrorrate and serrorrate correlation: 0.98
6. dsthostsrvserrorrate and srvserrorrate correlation: 0.99
7. dsthostsrvserrorrate and dsthosterrorrate correlation: 0.99
8. dsthostrrorrate and rerrorrate correlation: 0.93
9. dsthostrrorrate and svrerrorrate correlation: 0.92
10. dsthostsrvrrorrate and rerrorrate correlation: 0.96
11. dsthostsrvrrorrate and svrerrorrate correlation: 0.97

```
12. dsthostsvrerrorrate and dsthosterrorrate correlation: 0.92
13. samesrv_count and srvcount correlation: 0.98
14. dsthost_serrors_count and serrorrate correlation: 0.96
15. dsthost_serrors_count and svsserrorrate correlation: 0.96
16. dsthost_serrors_count and dsthosterrorrate correlation: 0.99
17. dsthost_serrors_count and dsthostsrvserrorrate correlation: 0.98
18. dsthost_diffsrv_count and dsthostdiffsrvrate correlation: 0.92`
```

Negative correlation < -0.7 is observed between

Positive correlation > 0.9 is observed between

```
1. Samesrvrate and serrorrate correlation: -0.76
2. samesrvrate and svsserrorrate correlation: -0.76
3. dsthosterrorrate and samesrvrate correlation: -0.76
4. dsthostsrvserrorrate and samesrvrate correlation: -0.77
5. attack_or_normal and samesrvrate correlation: -0.75
6. attack_or_normal and dsthostsvrcount correlation: -0.72
7. serrors_count and samesrvrate correlation: -0.73
8. dsthost_serrors_count and samesrvrate correlation: -0.76
```

## CHAPTER 3: HYPOTHESIS TESTING

```
In [ ]: nadp.to_csv("nadp",index=False)
nadp_add.to_csv("nadp_add",index = False)
nadp_boxcox.to_csv("nadp_boxcox",index = False)
```

```
In [ ]: nadp_add = pd.read_csv("./nadp_add")
```

```
In [ ]: nadp_boxcox = pd.read_csv("./nadp_boxcox")
```

### Network Traffic Volume and Anomalies

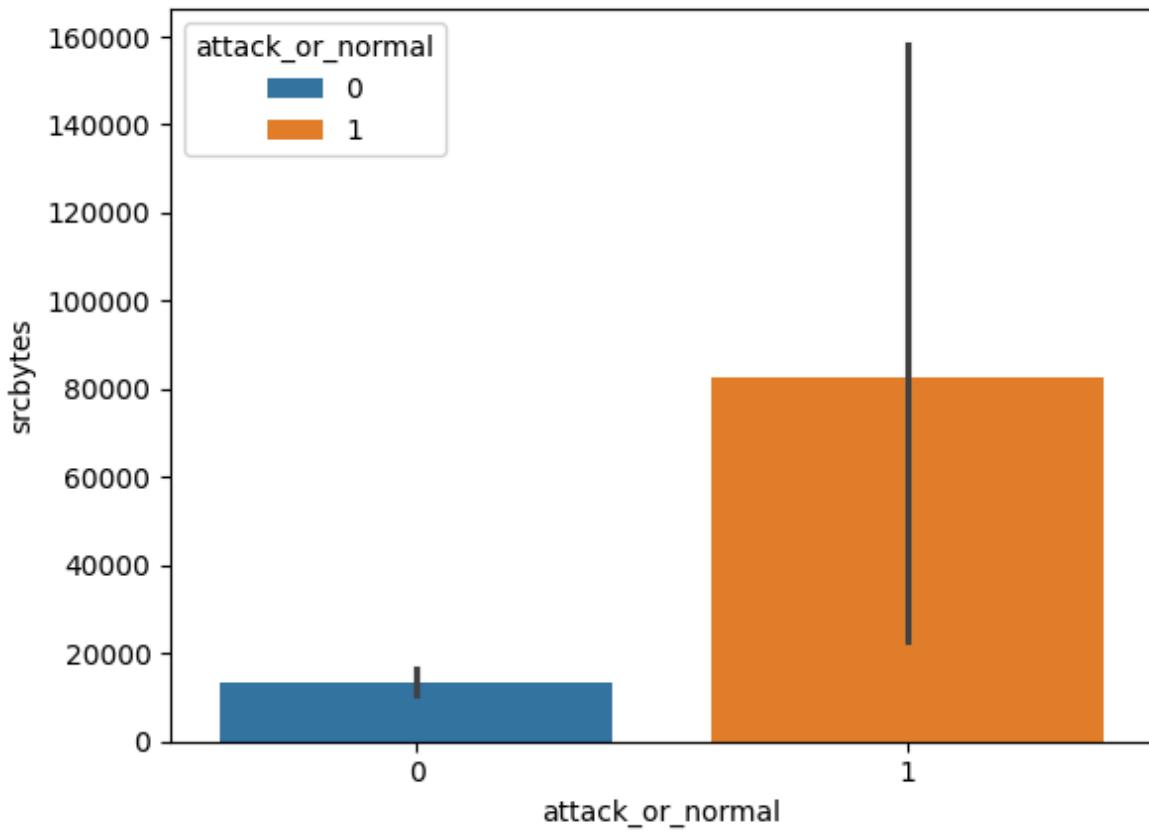
**Does network connections with unusually high or low traffic volumes (bytes transferred) are more likely to be anomalous?**

#### srcbytes vs attack\_or\_normal

```
In [ ]: # data groups
srcbytes_normal = nadp_add[nadp_add["attack_or_normal"] == 0]["srcbytes"]
srcbytes_attack = nadp_add[nadp_add["attack_or_normal"] == 1]["srcbytes"]
```

#### Visual Analysis

```
In [ ]: sns.barplot(data = nadp_add, x = "attack_or_normal",y = "srcbytes",estimator="mean",
Out[ ]: <Axes: xlabel='attack_or_normal', ylabel='srcbytes'>
```



### Observation

- Graph indicates that there is significant difference between srcbytes transferred during attack vs during normal.
- We can observe error bar has very high and unequal dispersions. So It is difficult to judge by bar plot. Let's use hypothesis testing.
- We can frame alternate hypothesis as the means of (srcbytes\_normal != srcbytes\_attack) or (srcbytes\_normal < srcbytes\_attack)

### Selection of Appropriate Test

- `attack_or_normal` is categorical column with two categories - 0 and 1 (normal and attack). `srcbytes` column is providing number of srcbytes transferred from source to destination in a network connection. `srcbytes` is numerical column.
- So here two independent samples are tested based on number of srcbytes transferred from source to destination.
- Comparing means of two independent samples is required. So `ttest_ind` is appropriate if distributions are normal. `Mann Whitney U test` is appropriate if distributions are non-normal

### ttest\_ind Hypothesis Formulation

#### two tailed\_t test

Null Hypothesis H0:

$$\mu_{srcbytes\_normal} = \mu_{srcbytes\_attack} \quad (1)$$

Alternate Hypothesis Ha:

$$\mu_{srcbytes\_normal} \neq \mu_{srcbytes\_attack} \quad (2)$$

two-tailed test

Significance level: 0.05

```
In [ ]: alpha = 0.05
```

```
In [ ]: print("variance of srcbytes_normal",np.var(srcbytes_normal),"\nvariance of srcbytes_"
variance of srcbytes_normal 174815997085.91464
variance of srcbytes_attack 73838812345841.42
```

```
In [ ]: tstat,p_val = ttest_ind(a=srcbytes_normal, b=srcbytes_attack, equal_var=False,altern
print("ttest_ind t_stat:", tstat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of ttest_ind test i.e.,
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of ttest_ind test
```

ttest\_ind t\_stat: -1.9616326188727324 p-value: 0.049809977020307705  
As p\_value <= 0.05, We reject the null hypothesis of ttest\_ind test i.e., the mean of srcbytes\_normal is not equal to the mean of srcbytes\_attack

### one\_tailed\_t\_test

Null Hypothesis H0:

$$\mu_{srcbytes\_normal} \geq \mu_{srcbytes\_attack} \quad (3)$$

Alternate Hypothesis Ha:

$$\mu_{srcbytes\_normal} < \mu_{srcbytes\_attack} \quad (4)$$

one-tailed test

Significance level: 0.05

```
In [ ]: tstat,p_val = ttest_ind(a=srcbytes_normal, b=srcbytes_attack, equal_var=False,altern
print("ttest_ind t_stat:", tstat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of ttest_ind test i.e.,
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of ttest_ind test
```

ttest\_ind t\_stat: -1.9616326188727324 p-value: 0.02490498851015385  
As p\_value <= 0.05, We reject the null hypothesis of ttest\_ind test i.e., the mean of srcbytes\_normal is less than the mean of srcbytes\_attack

### Check the assumptions of ttest\_ind

The two-samples independent t-test assume the following characteristics about the data:

**Independence** : The observations in each sample must be independent of each other. This means that the value of one observation should not be related to the value of any other observation in the same sample.

**Normality** : The data within each group should follow a normal distribution. However, the t-test is quite robust to violations of normality, especially for large sample sizes (typically n>30).

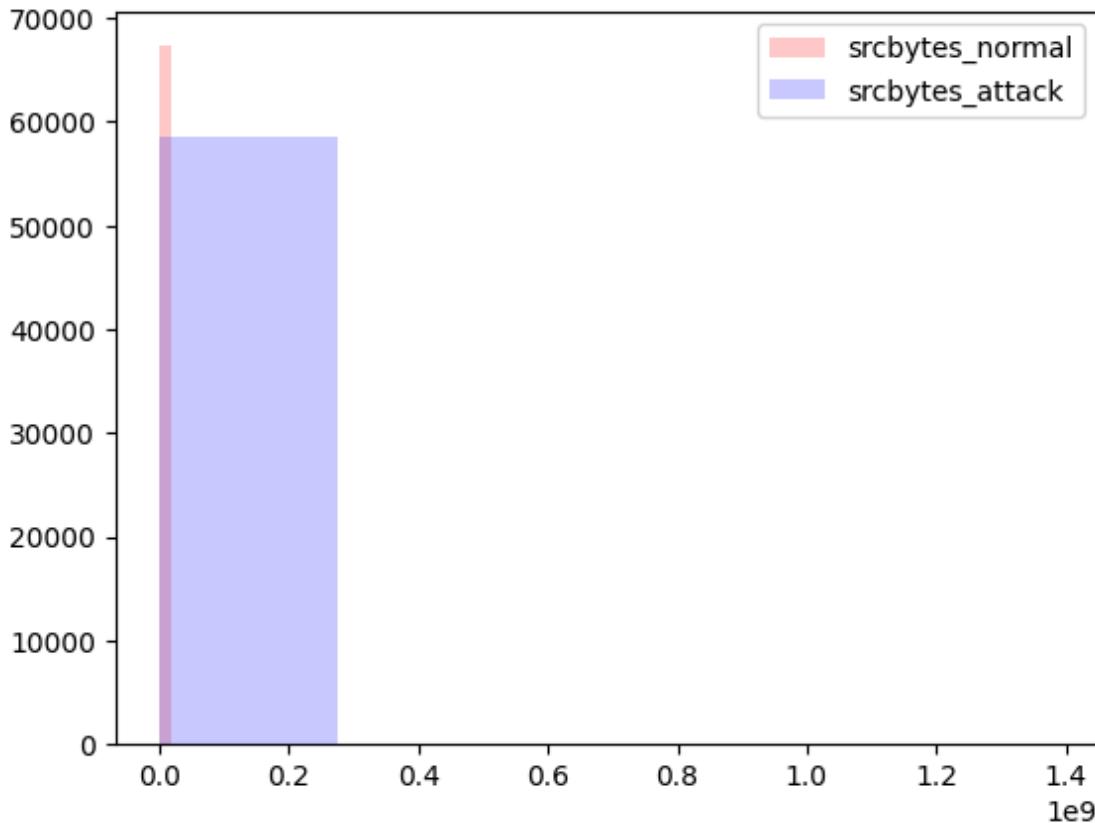
**Homogeneity of Variances** : The variances of the two groups should be equal.

### Independence

srcbytes\_normal and srcbytes\_attack are independent groups.

### Normality

```
In [ ]: ## Checking the distribution of the two
plt.hist(srcbytes_normal, bins=5, alpha=0.2, color='r', label='srcbytes_normal')
plt.hist(srcbytes_attack, bins=5, alpha=0.2, color='b', label='srcbytes_attack')
plt.legend()
plt.show()
```



NOTE:

Lets try with shapiro test for normality check.

```
In [ ]: ## Running the shapiro test
shapiro_stat1, shapiro_pval1 = shapiro(srcbytes_normal)
shapiro_stat2, shapiro_pval2 = shapiro(srcbytes_attack)

print("shapiro test for Normality:")
print("srcbytes_normal - Statistic:", shapiro_stat1, "p-value:", shapiro_pval1)
print("srcbytes_attack - Statistic:", shapiro_stat2, "p-value:", shapiro_pval2)

if shapiro_pval1 <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of shapiro test i.e., srcbytes_normal is not normal distributed")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of shapiro test i.e., srcbytes_normal is normal distributed")

if shapiro_pval2 <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of shapiro test i.e., srcbytes_attack is not normal distributed")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of shapiro test i.e., srcbytes_attack is normal distributed")
```

```
shapiro test for Normality:
srcbytes_normal - Statistic: 0.009855350719979783 p-value: 1.9375197153584948e-172
srcbytes_attack - Statistic: 0.001502592092851085 p-value: 1.4839287848635966e-168
As p_value <= 0.05, We reject the null hypothesis of shapiro test i.e., srcbytes_normal is not normal distributed
As p_value <= 0.05, We reject the null hypothesis of shapiro test i.e., srcbytes_attack is not normal distributed
```

NOTE:

We cannot use shapiro test because it is suited for sample size < 5000. Let's use anderson test because it is suited for large datasets.

```
In [ ]: # Running the Anderson test for both groups
anderson_result1 = anderson(srcbytes_normal, dist="norm")
anderson_result2 = anderson(srcbytes_attack, dist="norm")
```

```

# Displaying the results
print("Anderson test for Normality:")
print("srcbytes_normal - Statistic:", anderson_result1.statistic)
print("srcbytes_normal - Critical Values:", anderson_result1.critical_values)
print("srcbytes_attack - Statistic:", anderson_result2.statistic)
print("srcbytes_attack - Critical Values:", anderson_result2.critical_values)

# Define alpha
alpha = 0.05

# Check for normality in srcbytes_normal
if anderson_result1.statistic > anderson_result1.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")

# Check for normality in srcbytes_attack
if anderson_result2.statistic > anderson_result2.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")

```

Anderson test for Normality:

srcbytes\_normal - Statistic: 25385.264019912633  
srcbytes\_normal - Critical Values: [0.576 0.656 0.787 0.918 1.092]  
srcbytes\_attack - Statistic: 22587.708867528985  
srcbytes\_attack - Critical Values: [0.576 0.656 0.787 0.918 1.092]

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., srcbytes\_normal is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., srcbytes\_attack is not normally distributed.

### Checking Homogeneity of Variances

NOTE:

Levene test is better than Bartlett test with non-normally distributed datasets.

```

In [ ]: stat, p_value = levene(srcbytes_normal, srcbytes_attack)
print(f"levene test between srcbytes_normal and srcbytes_attack\nstat : {stat}\np_value : {p_value}")
if p_value <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of levene test i.e., srcbytes_normal and srcbytes_attack has unequal variances")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of levene test i.e., srcbytes_normal and srcbytes_attack has equal variances")

```

levene test between srcbytes\_normal and srcbytes\_attack  
stat : 4.43204234679084  
p\_value : 0.03527225420248859  
As p\_value <= 0.05, We reject the null hypothesis of levene test i.e., srcbytes\_attack and srcbytes\_normal has unequal variances

### Observation

as srcbytes\_normal and srcbytes\_attack groups are violating normality and homogeneity of variances assumptions, we cannot use ttest\_ind (parametric test).

So we need to apply non-parametric test - Mann Whitney U Test (also called as Wilcoxon Rank sum test)

### Mann Whitney U Test Hypothesis Formulation

#### **two tailed\_mannwhitneyu\_test**

Null Hypothesis H0:

$$Distribution_{srcbytes\_normal} = Distribution_{srcbytes\_attack} \quad (5)$$

Alternate Hypothesis Ha:

$$Distribution_{srcbytes\_normal} \neq Distribution_{srcbytes\_attack} \quad (6)$$

two-tailed test

Significance level: 0.05

```
In [ ]: stat,p_val = mannwhitneyu(x=srcbytes_normal, y=srcbytes_attack,alternative = "two-sided")
print("mannwhitneyu stat:", stat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of mannwhitneyu test i.e., the distribution of srcbytes_normal is stochastically not equal to the distribution of srcbytes_attack")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of mannwhitneyu test i.e., the distribution of srcbytes_normal is stochastically equal to the distribution of srcbytes_attack")
```

### one\_tailed\_mannwhitneyu\_test

Null Hypothesis H0:

$$Distribution_{srcbytes\_normal} \geq Distribution_{srcbytes\_attack} \quad (7)$$

Alternate Hypothesis Ha:

$$Distribution_{srcbytes\_normal} < Distribution_{srcbytes\_attack} \quad (8)$$

one-tailed test

Significance level: 0.05

```
In [ ]: stat,p_val = mannwhitneyu(x=srcbytes_normal, y=srcbytes_attack,alternative = "less")
print("mannwhitneyu stat:", stat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of mannwhitneyu test i.e., the distribution of srcbytes_normal is stochastically greater than or equal to the distribution of srcbytes_attack")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of mannwhitneyu test i.e., the distribution of srcbytes_normal is stochastically less than the distribution of srcbytes_attack")
```

**Observation**

As the ttest\_ind assumptions are not satisfied by srcbytes, We use mann\_whitney u test.

Two tailed test suggests that the distribution of srcbytes\_normal is stochastically not equal to the distribution of srcbytes\_attack.

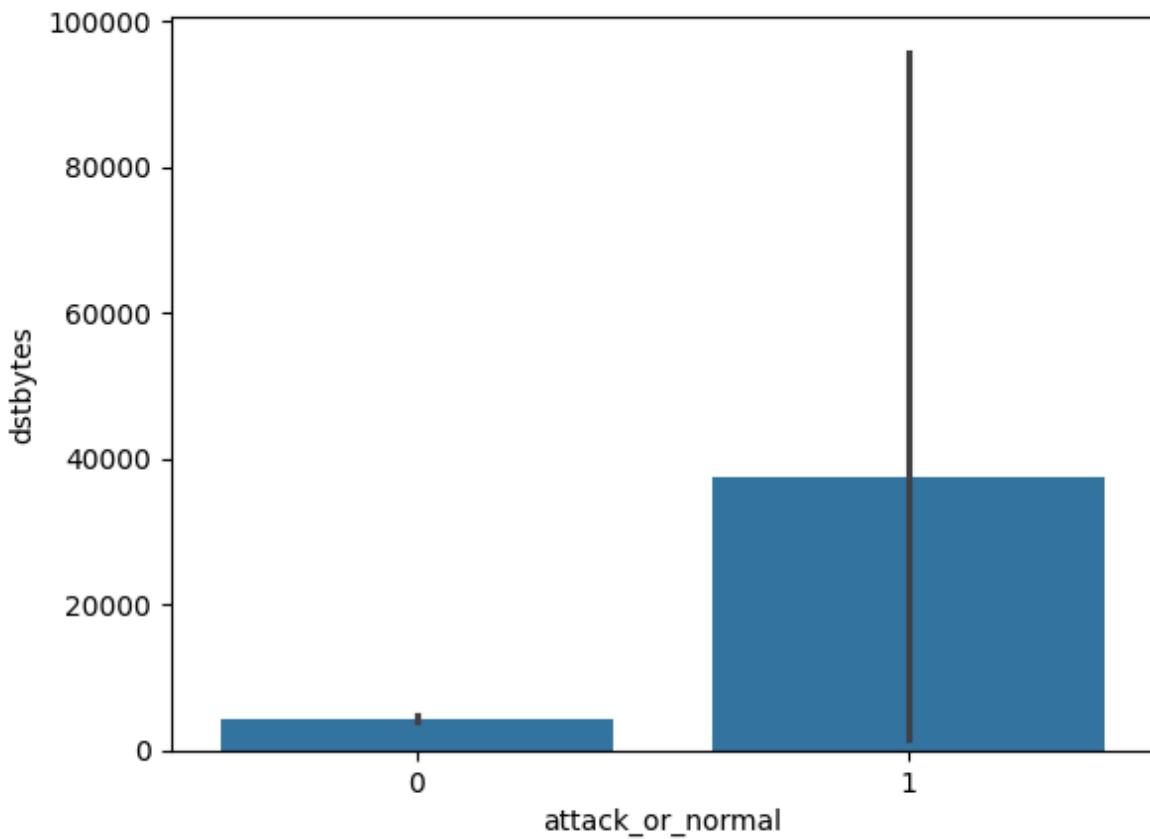
One tailed test suggests that the distribution of srcbytes\_normal is stochastically greater than or equal to the distribution of srcbytes\_attack

### dstbytes vs attack\_or\_normal

```
In [ ]: # data groups
dstbytes_normal = nadp_add[nadp_add["attack_or_normal"] == 0]["dstbytes"]
dstbytes_attack = nadp_add[nadp_add["attack_or_normal"] == 1]["dstbytes"]
```

### Visual Analysis

```
In [ ]: sns.barplot(data = nadp_add, x = "attack_or_normal",y = "dstbytes",estimator="mean")
Out[ ]: <Axes: xlabel='attack_or_normal', ylabel='dstbytes'>
```



### Observation

- Graph indicates that there is significant difference between dstbytes transferred during attack vs during normal.
- We can observe error bar has very high and unequal dispersions. So It is difficult to judge by bar plot. Let's use hypothesis testing.
- We can frame alternate hypothesis as the means of (dstbytes\_normal != dstbytes\_attack) or (dstbytes\_normal < dstbytes\_attack)

### Selection of Appropriate Test

- `attack_or_normal` is categorical column with two categories - 0 and 1 (normal and attack). `dstbytes` column is providing number of dstbytes transferred from destination to source in a network connection. `dstbytes` is numerical column.
- So here two independent samples are tested based on number of dstbytes transferred from destination to source.
- Comparing means of two independent samples is required. So `ttest_ind` is appropriate if distributions are normal. `Mann Whitney U test` is appropriate if distributions are non-normal

### `ttest_ind` Hypothesis Formulation

#### **two tailed\_t test**

Null Hypothesis H0:

$$\mu_{dstbytes\_normal} = \mu_{dstbytes\_attack} \quad (9)$$

Alternate Hypothesis Ha:

$$\mu_{dstbytes\_normal} \neq \mu_{dstbytes\_attack} \quad (10)$$

two-tailed test

Significance level: 0.05

```
In [ ]: print("variance of dstbytes_normal",np.var(dstbytes_normal),"\nvariance of dstbytes_"
variance of dstbytes_normal 4285316866.4747543
variance of dstbytes_attack 34738536668105.723
```

```
In [ ]: tstat,p_val = ttest_ind(a=dstbytes_normal, b=dstbytes_attack, equal_var=True,alterna
print("ttest_ind t_stat:", tstat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of ttest_ind test i.e.,
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of ttest_ind test
```

ttest\_ind t\_stat: -1.4614241258205836 p-value: 0.14390157812640422  
As p\_value > 0.05, We cannot reject the null hypothesis of ttest\_ind test i.e., the mean of dstbytes\_normal is equal to the mean of dstbytes\_attack

### one\_tailed\_t\_test

Null Hypothesis H0:

$$\mu_{dstbytes\_normal} \geq \mu_{dstbytes\_attack} \quad (11)$$

Alternate Hypothesis Ha:

$$\mu_{dstbytes\_normal} < \mu_{dstbytes\_attack} \quad (12)$$

one-tailed test

Significance level: 0.05

```
In [ ]: tstat,p_val = ttest_ind(a=dstbytes_normal, b=dstbytes_attack, equal_var=True,alterna
print("ttest_ind t_stat:", tstat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of ttest_ind test i.e.,
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of ttest_ind test
```

ttest\_ind t\_stat: -1.4614241258205836 p-value: 0.07195078906320211  
As p\_value > 0.05, We cannot reject the null hypothesis of ttest\_ind test i.e., the mean of dstbytes\_normal is greater than or equal to the mean of dstbytes\_attack

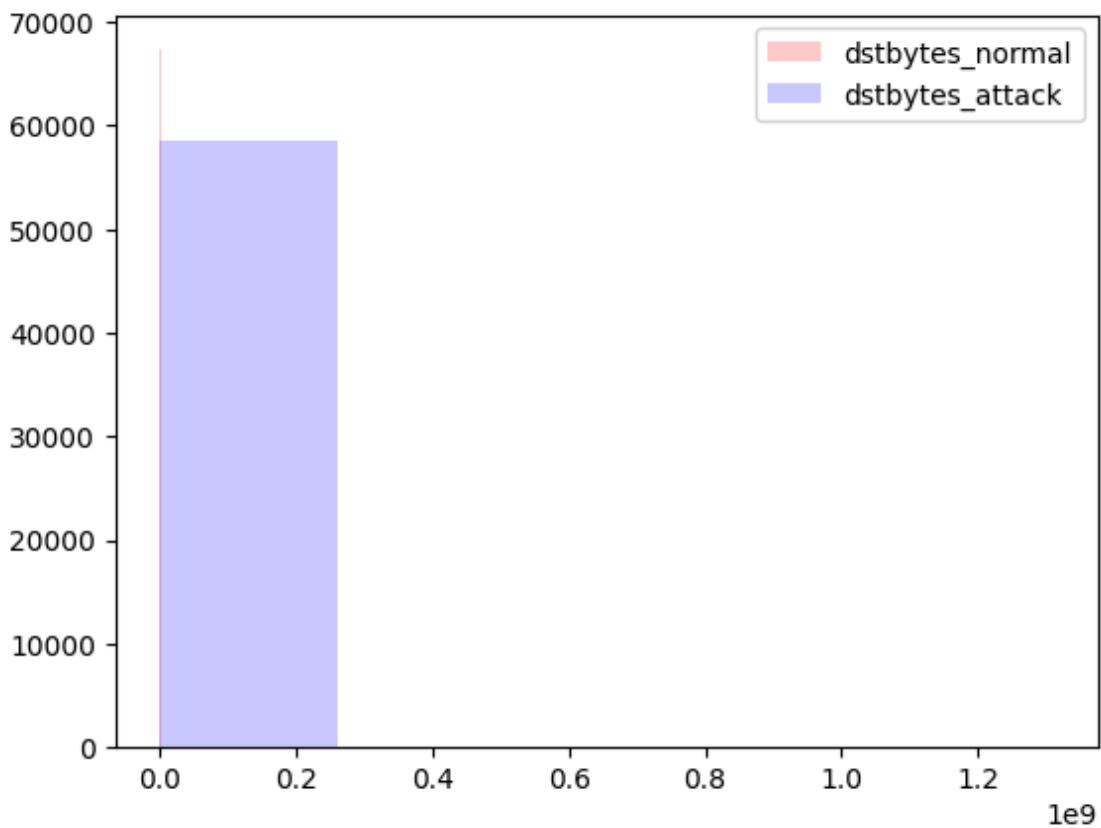
Check the assumptions of ttest\_ind

### Independence

dstbytes\_normal and dstbytes\_attack are independent groups.

### Normality

```
In [ ]: ## Checking the distribution of the two
plt.hist(dstbytes_normal, bins=5, alpha=0.2, color='r', label='dstbytes_normal')
plt.hist(dstbytes_attack, bins=5, alpha=0.2, color='b', label='dstbytes_attack')
plt.legend()
plt.show()
```



NOTE:

We cannot use shapiro test because it is suited for sample size < 5000. Let's use anderson test because it is suited for large datasets.

```
In [ ]: # Running the Anderson test for both groups
anderson_result1 = anderson(dstbytes_normal, dist="norm")
anderson_result2 = anderson(dstbytes_attack, dist="norm")

# Displaying the results
print("Anderson test for Normality:")
print("dstbytes_normal - Statistic:", anderson_result1.statistic)
print("dstbytes_normal - Critical Values:", anderson_result1.critical_values)
print("dstbytes_attack - Statistic:", anderson_result2.statistic)
print("dstbytes_attack - Critical Values:", anderson_result2.critical_values)

# Define alpha
alpha = 0.05

# Check for normality in dstbytes_normal
if anderson_result1.statistic > anderson_result1.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")

# Check for normality in dstbytes_attack
if anderson_result2.statistic > anderson_result2.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")
```

Anderson test for Normality:  
dstbytes\_normal - Statistic: 22908.983665953216  
dstbytes\_normal - Critical Values: [0.576 0.656 0.787 0.918 1.092]  
dstbytes\_attack - Statistic: 22628.74707960493  
dstbytes\_attack - Critical Values: [0.576 0.656 0.787 0.918 1.092]  
As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., dstbytes\_normal is not normally distributed.  
As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., dstbytes\_attack is not normally distributed.

### Checking Homogeneity of Variances

NOTE:

Levene test is better than Bartlett test with non-normally distributed datasets.

```
In [ ]: stat, p_value = levene(dstbytes_normal, dstbytes_attack)
print(f"levene test between dstbytes_normal and dstbytes_attack\nstat : {stat}\n")
if p_value <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of levene test i.e., dstbytes_normal and dstbytes_attack has equal variances")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of levene test i.e., dstbytes_normal and dstbytes_attack has equal variances")
```

levene test between dstbytes\_normal and dstbytes\_attack  
 stat : 2.152620133647356  
 p\_value : 0.1423293162438743  
 As p\_value > 0.05, We cannot reject the null hypothesis of levene test i.e., dstbytes\_normal and dstbytes\_attack has equal variances

### Observation

as dstbytes\_normal and dstbytes\_attack groups are violating normality assumption, we cannot use ttest\_ind (parametric test).

So we need to apply non-parametric test - Mann Whitney U Test (also called as Wilcoxon Rank sum test)

### Mann Whitney U Test Hypothesis Formulation

#### **two tailed\_mannwhitneyu\_test**

Null Hypothesis H0:

$$Distribution_{dstbytes\_normal} = Distribution_{dstbytes\_attack} \quad (13)$$

Alternate Hypothesis Ha:

$$Distribution_{dstbytes\_normal} \neq Distribution_{dstbytes\_attack} \quad (14)$$

two-tailed test

Significance level: 0.05

```
In [ ]: stat,p_val = mannwhitneyu(x=dstbytes_normal, y=dstbytes_attack,alternative = "two-sided")
print("mannwhitneyu stat:", stat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of mannwhitneyu test i.e., the distribution of dstbytes_normal is stochastically not equal to the distribution of dstbytes_attack")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of mannwhitneyu test i.e., the distribution of dstbytes_normal is stochastically equal to the distribution of dstbytes_attack")
```

mannwhitneyu stat: 3551587496.5 p-value: 0.0  
 As p\_value <= 0.05, We reject the null hypothesis of mannwhitneyu test i.e., the distribution of dstbytes\_normal is stochastically not equal to the distribution of dstbytes\_attack

#### **one\_tailed\_mannwhitneyu\_test**

Null Hypothesis H0:

$$Distribution_{dstbytes\_normal} \geq Distribution_{dstbytes\_attack} \quad (15)$$

Alternate Hypothesis Ha:

$$Distribution_{dstbytes\_normal} < Distribution_{dstbytes\_attack} \quad (16)$$

one-tailed test

Significance level: 0.05

```
In [ ]: stat,p_val = mannwhitneyu(x=dstbytes_normal, y=dstbytes_attack,alternative = "less")
print("mannwhitneyu t_stat:", stat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of mannwhitneyu test i.e., the distribution of dstbytes_normal is stochastically less than the distribution of dstbytes_attack")
```

```

else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of mannwhitneyu t
mannwhitneyu t_stat: 3551587496.5 p-value: 1.0
As p_value > 0.05, We cannot reject the null hypothesis of mannwhitneyu test i.e., th
e distribution of dstbytes_normal is stochastically greater than or equal to the dis
tribution of dstbytes_attack

```

### Observation

- As the ttest\_ind assumptions are not satisfied by dstbytes, We use mann\_whitney u test.
- Two tailed test suggests that the distribution of dstbytes\_normal is stochastically not equal to the distribution of dstbytes\_attack.
- One tailed test suggests that the distribution of dstbytes\_normal is stochastically greater than or equal to the distribution of dstbytes\_attack

### srcbytes vs attack\_categories

```

In [ ]: nadp_add["attack_category"].unique()

Out[ ]: array(['Normal', 'DOS', 'R2L', 'Probe', 'U2R'], dtype=object)

In [ ]: # data groups
srcbytes_Normal = nadp_add[nadp_add["attack_category"] == "Normal"]["srcbytes"]
srcbytes_DOS = nadp_add[nadp_add["attack_category"] == "DOS"]["srcbytes"]
srcbytes_R2L = nadp_add[nadp_add["attack_category"] == "R2L"]["srcbytes"]
srcbytes_Probe = nadp_add[nadp_add["attack_category"] == "Probe"]["srcbytes"]
srcbytes_U2R = nadp_add[nadp_add["attack_category"] == "U2R"]["srcbytes"]

```

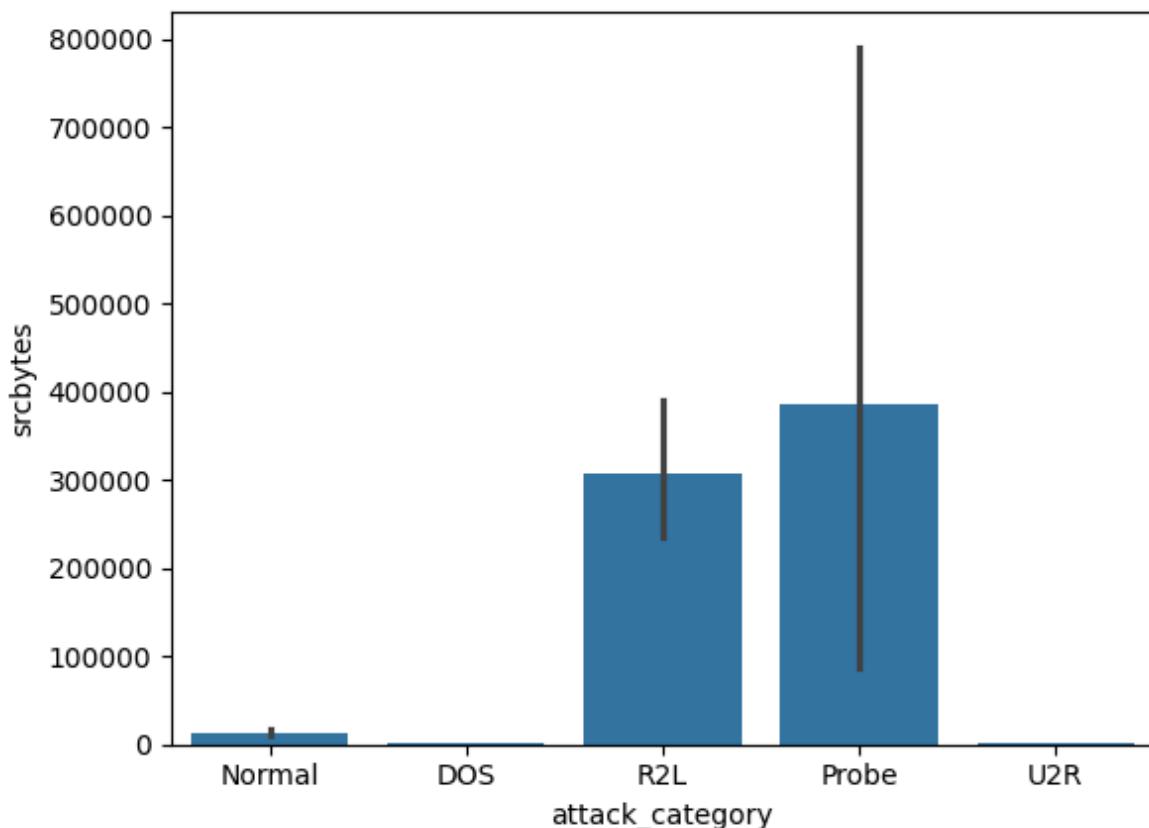
### Visual Analysis

```

In [ ]: sns.barplot(data = nadp_add, x = "attack_category",y = "srcbytes",estimator="mean")

Out[ ]: <Axes: xlabel='attack_category', ylabel='srcbytes'>

```



### Observation

- Graph indicates that there is significant difference between srcbytes transferred during different attack categories.

We can observe error bar has very high and unequal dispersions. So It is difficult to judge by bar plot. Let's use hypothesis testing.

We can frame alternate hypothesis as the atleast one of the attack category srcbytes average is different from others.

### Selection of Appropriate Test

`attack_category` is categorical column with five categories. `srcbytes` column is providing number of srcbytes transferred from source to destination in a network connection. `srcbytes` is numerical column.

So here five independent samples are tested based on number of srcbytes transferred from source to destination.

Comparing means of five independent samples is required. So

`f_oneway(anova)` is appropriate if distributions are normal.

`Kruskal(anova)` is appropriate if distributions are non-normal.

### `f_oneway` Hypothesis Formulation

Null Hypothesis H0: The means of srcbytes across all 5 attack categories are equal.

Alternate Hypothesis Ha: Atleast one of the attack category srcbytes mean is different from the others.

Significance level: 0.05

```
In [ ]: alpha = 0.05
```

```
In [ ]: print("variance of srcbytes_Normal",np.var(srcbytes_Normal),"\nvariance of srcbytes_DOS",np.var(srcbytes_DOS),"\nvariance of srcbytes_R2L",np.var(srcbytes_R2L),"\nvariance of srcbytes_Probe",np.var(srcbytes_Probe),"\nvariance of srcbytes_U2R",np.var(srcbytes_U2R))
```

variance of srcbytes\_Normal 174815997085.91464  
variance of srcbytes\_DOS 59075159.17597193  
variance of srcbytes\_R2L 1474659751379.953  
variance of srcbytes\_Probe 371162801803574.75  
variance of srcbytes\_U2R 1384808.0621301776

```
In [ ]: fstat,p_val = f_oneway(srcbytes_Normal,srcbytes_DOS,srcbytes_R2L,srcbytes_Probe,srcbytes_U2R)  
print("f_oneway f_stat:", fstat, "p-value:", p_val)  
if p_val <= alpha:  
    print("As p_value <= 0.05, We reject the null hypothesis of f_oneway test i.e.,")  
else:  
    print("As p_value > 0.05, We cannot reject the null hypothesis of f_oneway test")
```

f\_oneway f\_stat: 11.452740605522365 p-value: 2.706640046281792e-09  
As p\_value <= 0.05, We reject the null hypothesis of f\_oneway test i.e., Atleast one of the attack category srcbytes mean is different from the others.

### Check the assumptions of `f_oneway`

The two-samples independent t-test assume the following characteristics about the data:

`Independence` : The observations in each sample must be independent of each other. This means that the value of one observation should not be related to the value of any other observation in the same sample.

`Normality` : The data within each group should follow a normal distribution.

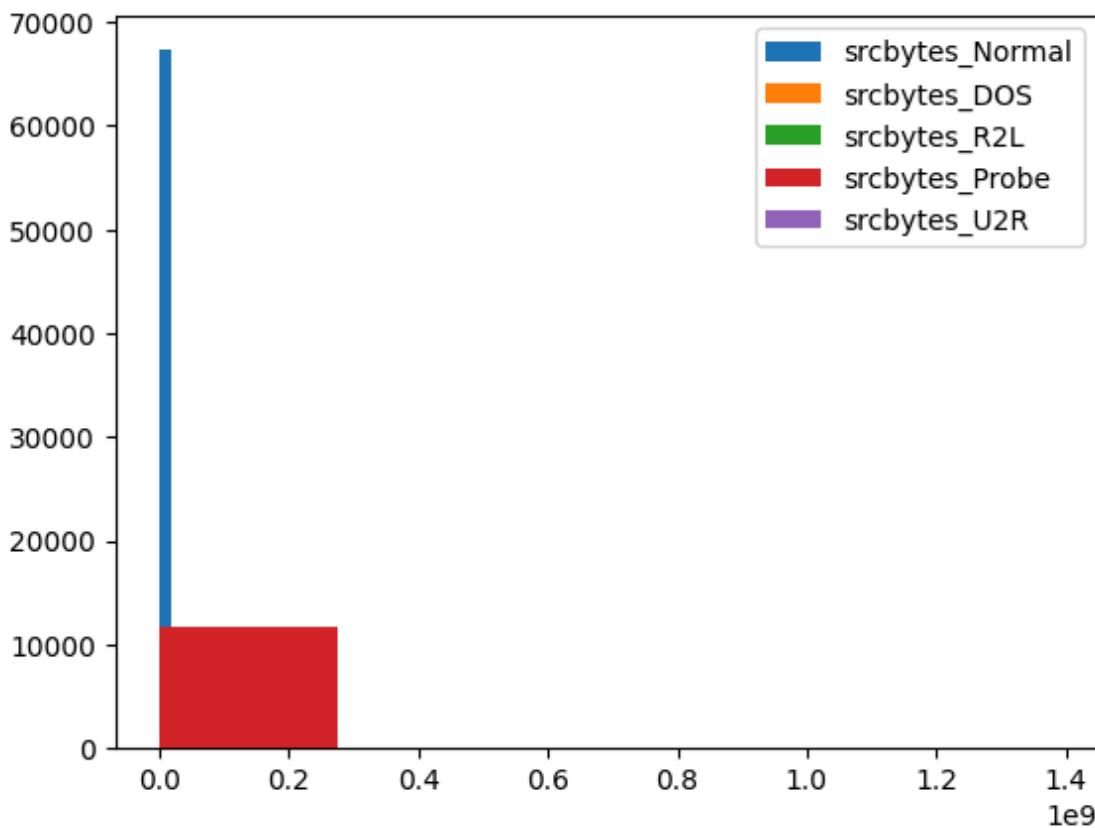
`Homogeneity of Variances` : The variances of the groups should be equal.

### `Independence`

srcbytes\_Normal, srcbytes\_DOS, srcbytes\_R2L, srcbytes\_Probe and srcbytes\_U2R are independent groups.

## Normality

```
In [ ]: ## Checking the distribution of the Five groups
plt.hist(srcbytes_Normal, bins=5, label='srcbytes_Normal')
plt.hist(srcbytes_DOS, bins=5, label='srcbytes_DOS')
plt.hist(srcbytes_R2L, bins=5, label='srcbytes_R2L')
plt.hist(srcbytes_Probe, bins=5, label='srcbytes_Probe')
plt.hist(srcbytes_U2R, bins=5, label='srcbytes_U2R')
plt.legend()
plt.show()
```



NOTE:

We cannot use shapiro test because it is suited for sample size < 5000. Let's use anderson test because it is suited for large datasets.

```
In [ ]: # Running the Anderson test for both groups
anderson_result1 = anderson(srcbytes_Normal, dist="norm")
anderson_result2 = anderson(srcbytes_DOS, dist="norm")
anderson_result3 = anderson(srcbytes_R2L, dist="norm")
anderson_result4 = anderson(srcbytes_Probe, dist="norm")
anderson_result5 = anderson(srcbytes_U2R, dist="norm")

# Displaying the results
print("Anderson test for Normality:")
print("srcbytes_Normal - Statistic:", anderson_result1.statistic)
print("srcbytes_Normal - Critical Values:", anderson_result1.critical_values)
print("srcbytes_DOS - Statistic:", anderson_result2.statistic)
print("srcbytes_DOS - Critical Values:", anderson_result2.critical_values)
print("srcbytes_R2L - Statistic:", anderson_result3.statistic)
print("srcbytes_R2L - Critical Values:", anderson_result3.critical_values)
print("srcbytes_Probe - Statistic:", anderson_result4.statistic)
print("srcbytes_Probe - Critical Values:", anderson_result4.critical_values)
print("srcbytes_U2R - Statistic:", anderson_result5.statistic)
print("srcbytes_U2R - Critical Values:", anderson_result5.critical_values)

# Define alpha
alpha = 0.05

# Check for normality in srcbytes_Normal
if anderson_result1.statistic > anderson_result1.critical_values[2]: # Use the 5% c
```

```

        print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
    else:
        print("As the statistic does not exceed the 5% critical value, we cannot reject")

    # Check for normality in srcbytes_DOS
    if anderson_result2.statistic > anderson_result2.critical_values[2]: # Use the 5% critical values
        print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
    else:
        print("As the statistic does not exceed the 5% critical value, we cannot reject")

    # Check for normality in srcbytes_R2L
    if anderson_result3.statistic > anderson_result3.critical_values[2]: # Use the 5% critical values
        print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
    else:
        print("As the statistic does not exceed the 5% critical value, we cannot reject")

    # Check for normality in srcbytes_Probe
    if anderson_result4.statistic > anderson_result4.critical_values[2]: # Use the 5% critical values
        print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
    else:
        print("As the statistic does not exceed the 5% critical value, we cannot reject")

    # Check for normality in srcbytes_U2R
    if anderson_result5.statistic > anderson_result5.critical_values[2]: # Use the 5% critical values
        print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
    else:
        print("As the statistic does not exceed the 5% critical value, we cannot reject")

```

Anderson test for Normality:

```

srcbytes_Normal - Statistic: 25385.264019912633
srcbytes_Normal - Critical Values: [0.576 0.656 0.787 0.918 1.092]
srcbytes_DOS - Statistic: 16819.1670102691
srcbytes_DOS - Critical Values: [0.576 0.656 0.787 0.918 1.092]
srcbytes_R2L - Statistic: 352.1482776365201
srcbytes_R2L - Critical Values: [0.574 0.653 0.784 0.914 1.088]
srcbytes_Probe - Statistic: 4496.639246542389
srcbytes_Probe - Critical Values: [0.576 0.656 0.787 0.918 1.092]
srcbytes_U2R - Statistic: 4.191363283649558
srcbytes_U2R - Critical Values: [0.539 0.614 0.737 0.86 1.023]

```

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., srcbytes\_Normal is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., srcbytes\_DOS is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., srcbytes\_R2L is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., srcbytes\_Probe is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., srcbytes\_U2R is not normally distributed.

### Checking Homogeneity of Variances

NOTE:

Levene test is better than Bartlett test with non-normally distributed datasets.

```
In [ ]: stat, p_value = levene(srcbytes_Normal, srcbytes_DOS, srcbytes_R2L, srcbytes_Probe,
print(f"levene test between srcbytes_Normal, srcbytes_DOS, srcbytes_R2L, srcbytes_Probe")
if p_value <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of levene test i.e., srcbytes_U2R")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of levene test i.e., srcbytes_U2R")
```

```
levene test between srcbytes_Normal, srcbytes_DOS, srcbytes_R2L, srcbytes_Probe, srcbytes_U2R
stat : 11.45546600342747
p_value : 2.6925457417182913e-09
```

As p\_value <= 0.05, We reject the null hypothesis of levene test i.e., srcbytes\_Normal, srcbytes\_DOS, srcbytes\_R2L, srcbytes\_Probe, srcbytes\_U2R has unequal variances

### Observation

as srcbytes\_Normal, srcbytes\_DOS, srcbytes\_R2L, srcbytes\_Probe, srcbytes\_U2R groups are violating normality and homogeneity of variances assumptions, we cannot use f\_oneway (parametric test).

So we need to apply non-parametric test - Kruskal Wallis test.

### Kruskal Wallis Test Hypothesis Formulation

Null Hypothesis H0: The medians of srcbytes across all 5 attack categories are equal.

Alternate Hypothesis Ha: Atleast one of the attack category srcbytes median is different from the others.

Significance level: 0.05

```
In [ ]: fstat,p_val = kruskal(srcbytes_Normal,srcbytes_DOS,srcbytes_R2L,srcbytes_Probe,srcby  
print("kruskal f_stat:", fstat, "p-value:", p_val)  
if p_val <= alpha:  
    print("As p_value <= 0.05, We reject the null hypothesis of kruskal test i.e., A  
else:  
    print("As p_value > 0.05, We cannot reject the null hypothesis of kruskal test i
```

kruskal f\_stat: 68347.04983221697 p-value: 0.0  
As p\_value <= 0.05, We reject the null hypothesis of kruskal test i.e., Atleast one o  
f the attack category srcbytes median is different from the others.

### Observation

As the f\_oneway assumptions are not satisfied by srcbytes, We use kruskal test.

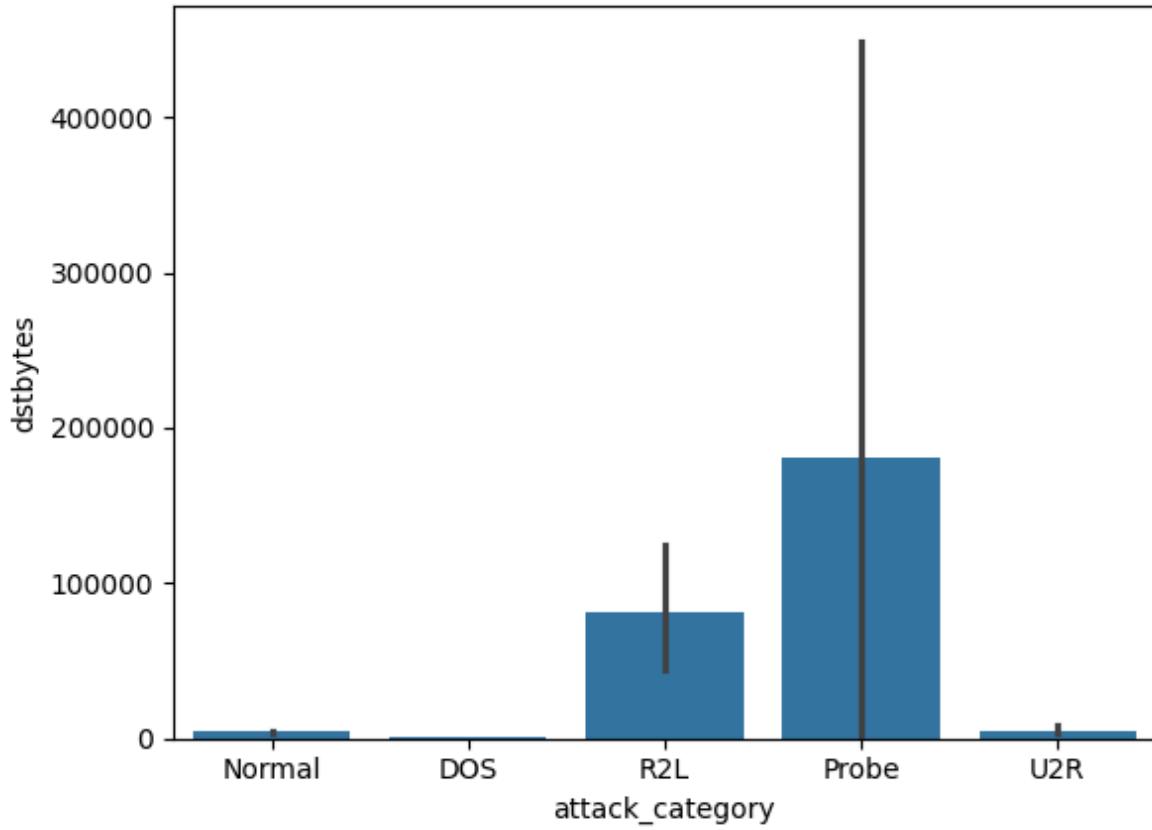
Kruskal test suggests that Atleast one of the attack category srcbytes median is different from the others.

### dstbytes vs attack\_categories

```
In [ ]: # data groups  
dstbytes_Normal = nadp_add[nadp_add["attack_category"] == "Normal"]["dstbytes"]  
dstbytes_DOS = nadp_add[nadp_add["attack_category"] == "DOS"]["dstbytes"]  
dstbytes_R2L = nadp_add[nadp_add["attack_category"] == "R2L"]["dstbytes"]  
dstbytes_Probe = nadp_add[nadp_add["attack_category"] == "Probe"]["dstbytes"]  
dstbytes_U2R = nadp_add[nadp_add["attack_category"] == "U2R"]["dstbytes"]
```

### Visual Analysis

```
In [ ]: sns.barplot(data = nadp_add, x = "attack_category",y = "dstbytes",estimator="mean")  
Out[ ]: <Axes: xlabel='attack_category', ylabel='dstbytes'>
```



### Observation

- Graph indicates that there is significant difference between dstbytes transferred during different attack categories.
- We can observe error bar has very high and unequal dispersions. So It is difficult to judge by bar plot. Let's use hypothesis testing.
- We can frame alternate hypothesis as the atleast one of the attack category dstbytes average is different from others.

### Selection of Appropriate Test

- `attack_category` is categorical column with five categories. `dstbytes` column is providing number of dstbytes transferred from source to destination in a network connection. `dstbytes` is numerical column.
- So here five independent samples are tested based on number of dstbytes transferred from source to destination.
- Comparing means of five independent samples is required. So `f_oneway(anova)` is appropriate if distributions are normal. `Kruskal(anova)` is appropriate if distributions are non-normal.

### `f_oneway` Hypothesis Formulation

- Null Hypothesis H0: The means of dstbytes across all 5 attack categories are equal.
- Alternate Hypothesis Ha: Atleast one of the attack category dstbytes mean is different from the others.
- Significance level: 0.05

```
In [ ]: alpha = 0.05
```

```
In [ ]: print("variance of dstbytes_Normal",np.var(dstbytes_Normal),"\nvariance of dstbytes_
```

```
variance of dstbytes_Normal 4285316866.4747543
variance of dstbytes_DOS 1364202.8034613945
variance of dstbytes_R2L 396347848449.0807
variance of dstbytes_Probe 174675676441514.1
variance of dstbytes_U2R 99682360.96005917
```

```
In [ ]: fstat,p_val = f_oneway(dstbytes_Normal,dstbytes_DOS,dstbytes_R2L,dstbytes_Probe,dstbytes_U2R)
print("f_oneway f_stat:", fstat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of f_oneway test i.e.,")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of f_oneway test")
```

f\_oneway f\_stat: 5.269882113259665 p-value: 0.00030559909673138076

As p\_value <= 0.05, We reject the null hypothesis of f\_oneway test i.e., Atleast one of the attack category dstbytes mean is different from the others.

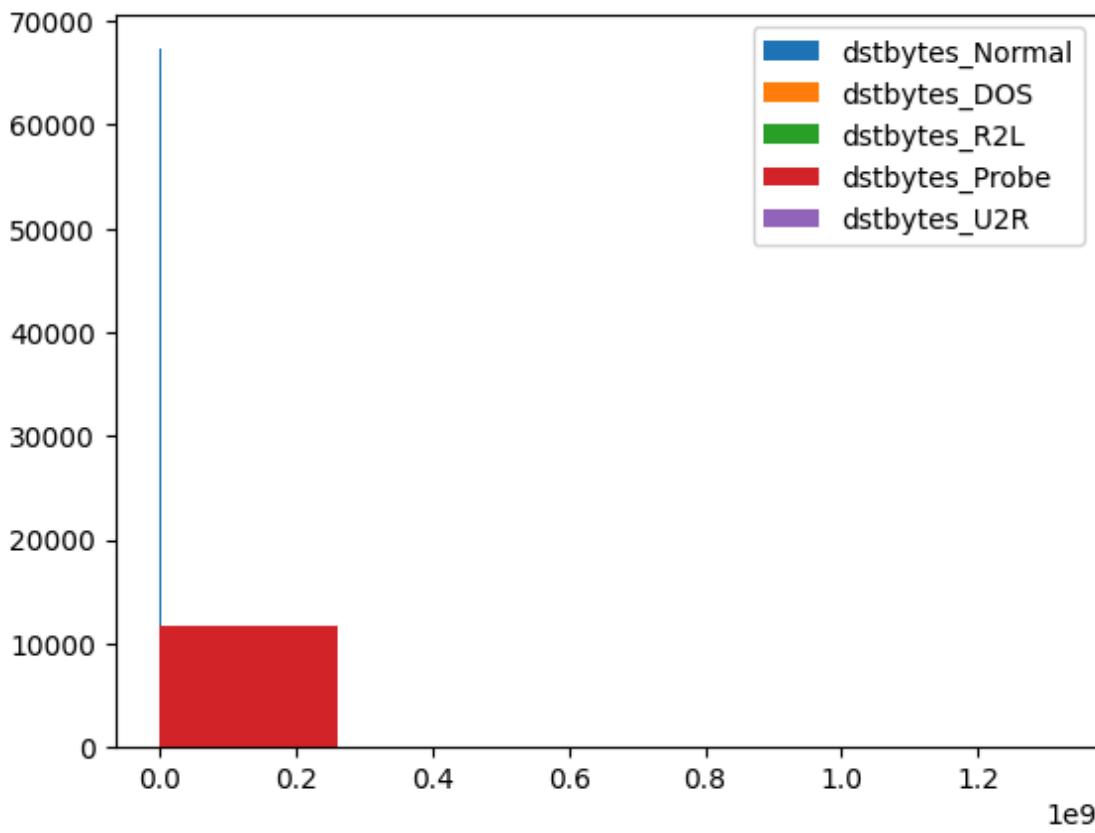
Check the assumptions of f\_oneway

### Independence

dstbytes\_Normal, dstbytes\_DOS, dstbytes\_R2L, dstbytes\_Probe and dstbytes\_U2R are independent groups.

### Normality

```
In [ ]: ## Checking the distribution of the Five groups
plt.hist(dstbytes_Normal, bins=5, label='dstbytes_Normal')
plt.hist(dstbytes_DOS, bins=5, label='dstbytes_DOS')
plt.hist(dstbytes_R2L, bins=5, label='dstbytes_R2L')
plt.hist(dstbytes_Probe, bins=5, label='dstbytes_Probe')
plt.hist(dstbytes_U2R, bins=5, label='dstbytes_U2R')
plt.legend()
plt.show()
```



NOTE:

We cannot use shapiro test because it is suited for sample size < 5000. Let's use anderson test because it is suited for large datasets.

```
In [ ]: # Running the Anderson test for both groups
anderson_result1 = anderson(dstbytes_Normal, dist="norm")
anderson_result2 = anderson(dstbytes_DOS, dist="norm")
anderson_result3 = anderson(dstbytes_R2L, dist="norm")
anderson_result4 = anderson(dstbytes_Probe, dist="norm")
```

```

anderson_result5 = anderson(dstbytes_U2R, dist="norm")

# Displaying the results
print("Anderson test for Normality:")
print("dstbytes_Normal - Statistic:", anderson_result1.statistic)
print("dstbytes_Normal - Critical Values:", anderson_result1.critical_values)
print("dstbytes_DOS - Statistic:", anderson_result2.statistic)
print("dstbytes_DOS - Critical Values:", anderson_result2.critical_values)
print("dstbytes_R2L - Statistic:", anderson_result3.statistic)
print("dstbytes_R2L - Critical Values:", anderson_result3.critical_values)
print("dstbytes_Probe - Statistic:", anderson_result4.statistic)
print("dstbytes_Probe - Critical Values:", anderson_result4.critical_values)
print("dstbytes_U2R - Statistic:", anderson_result5.statistic)
print("dstbytes_U2R - Critical Values:", anderson_result5.critical_values)

# Define alpha
alpha = 0.05

# Check for normality in dstbytes_Normal
if anderson_result1.statistic > anderson_result1.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")

# Check for normality in dstbytes_DOS
if anderson_result2.statistic > anderson_result2.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")

# Check for normality in dstbytes_R2L
if anderson_result3.statistic > anderson_result3.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")

# Check for normality in dstbytes_Probe
if anderson_result4.statistic > anderson_result4.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")

# Check for normality in dstbytes_U2R
if anderson_result5.statistic > anderson_result5.critical_values[2]: # Use the 5% critical value
    print("As the statistic exceeds the 5% critical value, we reject the null hypothesis")
else:
    print("As the statistic does not exceed the 5% critical value, we cannot reject the null hypothesis")

```

Anderson test for Normality:

dstbytes\_Normal - Statistic: 22908.983665953216  
dstbytes\_Normal - Critical Values: [0.576 0.656 0.787 0.918 1.092]  
dstbytes\_DOS - Statistic: 17311.759925765225  
dstbytes\_DOS - Critical Values: [0.576 0.656 0.787 0.918 1.092]  
dstbytes\_R2L - Statistic: 372.4592566912097  
dstbytes\_R2L - Critical Values: [0.574 0.653 0.784 0.914 1.088]  
dstbytes\_Probe - Statistic: 4500.543280635984  
dstbytes\_Probe - Critical Values: [0.576 0.656 0.787 0.918 1.092]  
dstbytes\_U2R - Statistic: 9.899747436556979  
dstbytes\_U2R - Critical Values: [0.539 0.614 0.737 0.86 1.023]

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., dstbytes\_Normal is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., dstbytes\_DOS is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., dstbytes\_R2L is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., dstbytes\_Probe is not normally distributed.

As the statistic exceeds the 5% critical value, we reject the null hypothesis of the Anderson test; i.e., dstbytes\_U2R is not normally distributed.

## Checking Homogeneity of Variances

NOTE:

Levene test is better than Bartlett test with non-normally distributed datasets.

```
In [ ]: stat, p_value = levene(dstbytes_Normal, dstbytes_DOS, dstbytes_R2L, dstbytes_Probe, dstbytes_U2R)
print("levene test between dstbytes_Normal, dstbytes_DOS, dstbytes_R2L, dstbytes_Probe, dstbytes_U2R")
if p_value <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of levene test i.e., dstbytes_Normal, dstbytes_DOS, dstbytes_R2L, dstbytes_Probe, dstbytes_U2R has unequal variances")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of levene test i.e., dstbytes_Normal, dstbytes_DOS, dstbytes_R2L, dstbytes_Probe, dstbytes_U2R has equal variances")
```

levene test between dstbytes\_Normal, dstbytes\_DOS, dstbytes\_R2L, dstbytes\_Probe, dstbytes\_U2R  
stat : 5.274135958643906  
p\_value : 0.00030323396798365246  
As p\_value <= 0.05, We reject the null hypothesis of levene test i.e., dstbytes\_Normal, dstbytes\_DOS, dstbytes\_R2L, dstbytes\_Probe, dstbytes\_U2R has unequal variances

### Observation

as dstbytes\_Normal, dstbytes\_DOS, dstbytes\_R2L, dstbytes\_Probe, dstbytes\_U2R groups are violating normality and homogeneity of variances assumptions, we cannot use f\_one\_way (parametric test).

So we need to apply non-parametric test - Kruskal Wallis test.

### Kruskal Wallis Test Hypothesis Formulation

Null Hypothesis H0: The medians of dstbytes across all 5 attack categories are equal.

Alternate Hypothesis Ha: Atleast one of the attack category dstbytes median is different from the others.

Significance level: 0.05

```
In [ ]: fstat,p_val = kruskal(dstbytes_Normal,dstbytes_DOS,dstbytes_R2L,dstbytes_Probe,dstbytes_U2R)
print("kruskal f_stat:", fstat, "p-value:", p_val)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of kruskal test i.e., Atleast one of the attack category dstbytes median is different from the others")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of kruskal test i.e., All attack category dstbytes median are equal")
```

kruskal f\_stat: 72122.47580082729 p-value: 0.0  
As p\_value <= 0.05, We reject the null hypothesis of kruskal test i.e., Atleast one of the attack category dstbytes median is different from the others.

### Observation

As the f\_oneway assumptions are not satisfied by dstbytes, We use kruskal test.

Kruskal test suggests that Atleast one of the attack category dstbytes median is different from the others.

## Impact of Protocol Type on Anomaly Detection

**Does Certain protocols are more frequently associated with network anomalies?**

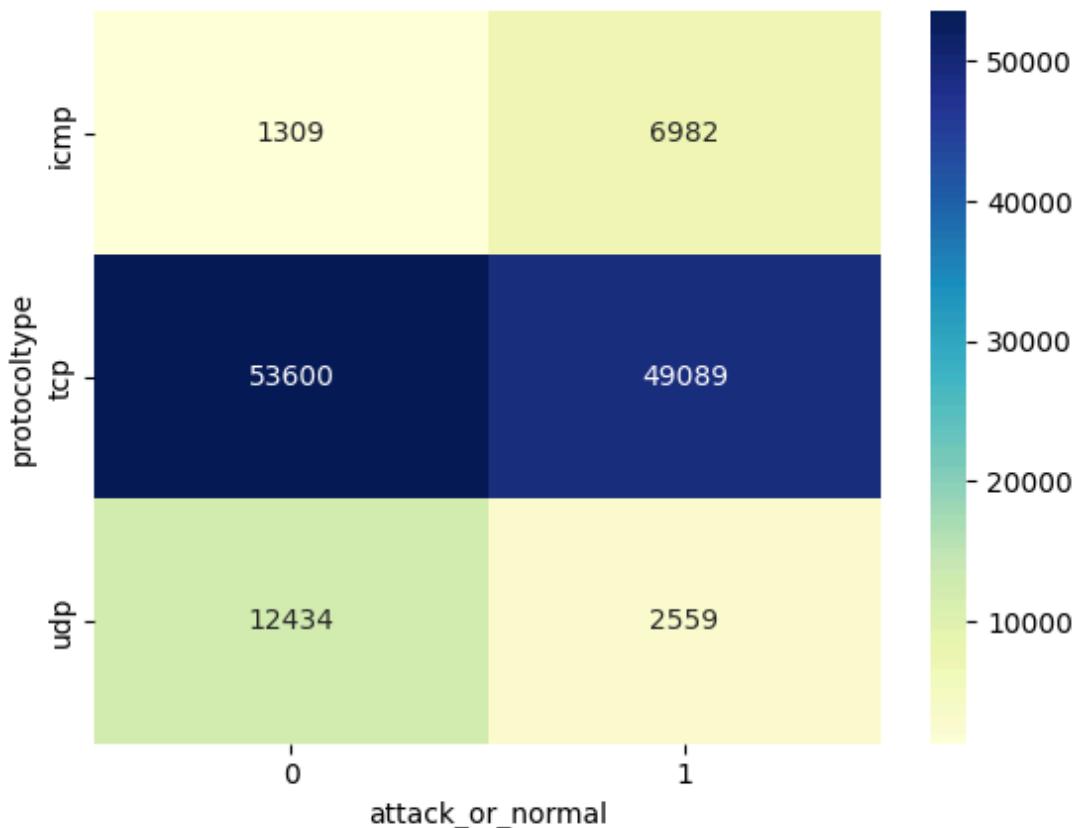
### protocoltype vs attack\_or\_normal

```
In [ ]: # data groups
Protocoltype = nadp_add["Protocoltype"]
Attack_or_normal = nadp_add["attack_or_normal"]
# Create a contingency table
contingency_table = pd.crosstab(Protocoltype, Attack_or_normal)
```

### Visual Analysis

```
In [ ]: sns.heatmap(contingency_table, annot=True, fmt="d", cmap="YlGnBu")
```

```
Out[ ]: <Axes: xlabel='attack_or_normal', ylabel='protocoltype'>
```



### Observation

It is difficult to judge the relation between two categorical features by contingency heat plot. Let's use hypothesis testing.

We can frame alternate hypothesis as protocoltype influences attack\_or\_normal categorization.

### Selection of Appropriate Test

`attack_or_normal` is categorical column with two categories - 0 and 1 (normal and attack). `protocoltype` column is providing type of network protocol used in a network connection. It is a categorical column.

So here the independence between two categorical features should be tested.

Comparing observed contingency table with expected contingency table is required. So `chi2_contingency` test is appropriate if atleast 20% cells in expected frequencies  $> 5$ . `fisher_exact` test is appropriate if more than 20% cells in expected frequencies  $< 5$  and 2X2 contingency table. For larger table, we should use advanced techniques like `chi2_contingency` along with `montecarlo simulation`.

### chi2\_contingency Hypothesis Formulation

Null Hypothesis H0: protocoltype does not influences whether a connection is classified as attack\_or\_normal.

Alternate Hypothesis Ha: protocoltype does influences whether a connection is classified as attack\_or\_normal.

Significance level: 0.05

```
In [ ]: alpha = 0.05
```

```
In [ ]: chi2, p_value, dof, expected = chi2_contingency(contingency_table)
print(f"Chi-square Statistic: {chi2}")
print(f"P-value: {p_value}")
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies:")
print(expected)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of chi2_contingency test")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of chi2_contingency test")
```

Chi-square Statistic: 10029.24862778463  
P-value: 0.0  
Degrees of Freedom: 2  
Expected Frequencies:  
[[ 4432.22605638 3858.77394362]  
[ 54895.77391187 47793.22608813]  
[ 8015.00003175 6977.99996825]]  
As p\_value <= 0.05, We reject the null hypothesis of chi2\_contingency test i.e., protocoltype does influences whether a connection is classified as attack\_or\_normal.

### Check the assumptions of chi2\_contingency test

The chi2\_contingency test assume the following characteristics about the data:

- Independence** : The observations in each sample must be independent of each other. This means that the value of one observation should not be related to the value of any other observation in the same sample.
- Expected counts** : No more than 20% of the expected counts should be less than 5, and all individual expected counts should be 1 or greater.

### Independence

- protocoltype and attack\_or\_normal are independent groups.

### Expected counts

```
In [ ]: expected_df = pd.DataFrame(expected,
                                 index=contingency_table.index.tolist(),
                                 columns=contingency_table.columns.tolist())
sns.heatmap(expected_df, annot=True, fmt=".2f", cmap="Blues", cbar=True)
```

Out[ ]: <Axes: >



All values in expected contingency table are greater than 5.

## Observation

As all the assumptions of chi2\_contingency test is satisfied. No need to apply chi2\_contingency with monte carlo simulation.(larger table)

chi2\_contingency test suggests that protocoltype does influences whether a connection is classified as attack\_or\_normal.

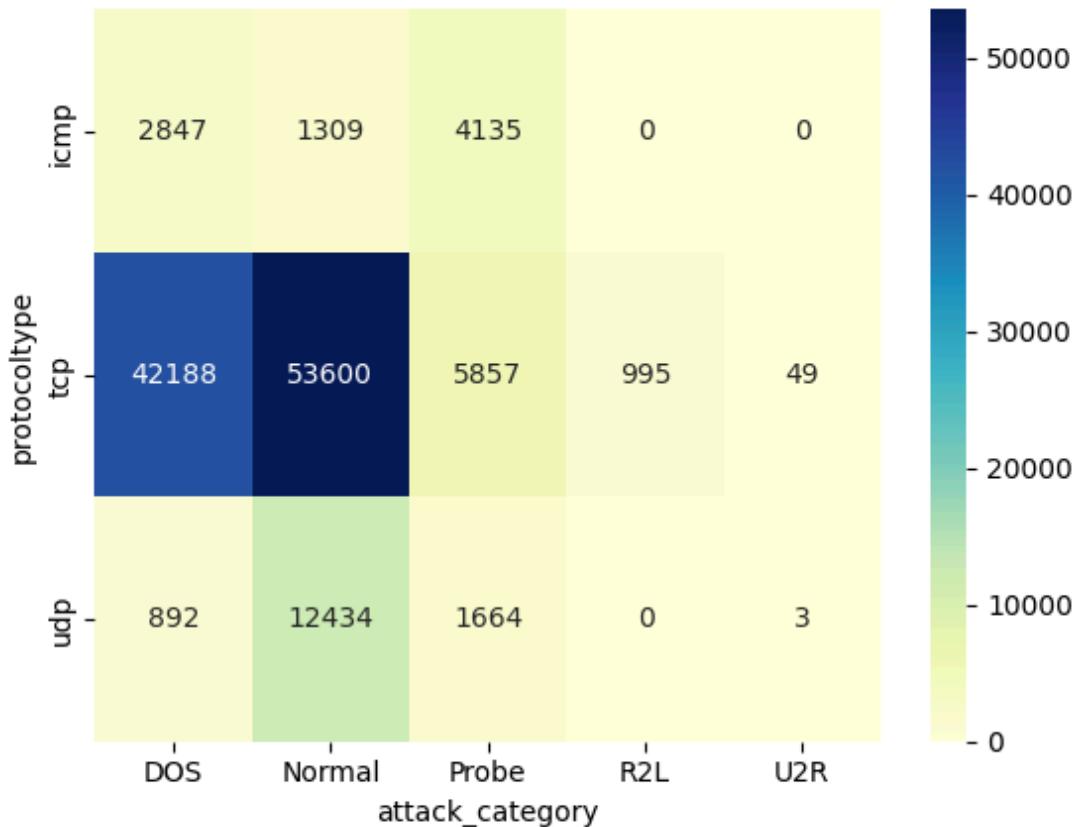
## protocoltype vs attack\_categories

```
In [ ]: # data groups
Protocoltype = nadp_add["Protocoltype"]
Attack_category = nadp_add["attack_category"]
# Create a contingency table
contingency_table = pd.crosstab(Protocoltype, Attack_category)
```

### Visual Analysis

```
In [ ]: sns.heatmap(contingency_table, annot=True, fmt="d", cmap="YlGnBu")
```

```
Out[ ]: <Axes: xlabel='attack_category', ylabel='Protocoltype'>
```



## Observation

It is difficult to judge the relation between two categorical features by contingency heat plot. Let's use hypothesis testing.

We can frame alternate hypothesis as protocoltype influences attack\_category categorization.

## Selection of Appropriate Test

attack\_category is categorical column with five categories. protocoltype column is providing type of network protocol used in a network connection. It is a categorical column.

So here the independence between two categorical features should be tested.

Comparing observed contingency table with expected contingency table is required. So `chi2_contingency` test is appropriate if atleast 20% cells in expected frequencies > 5. `fisher_exact` test is appropriate if more than 20% cells in expected frequencies < 5 and 2X2 contingency table. For larger table, we should use advanced techniques like `chi2_contingency` along with `montecarlo simulation`.

### chi2\_contingency Hypothesis Formulation

- Null Hypothesis H0: protocoltype does not influences attack\_category.
- Alternate Hypothesis Ha: protocoltype does influences attack\_category.
- Significance level: 0.05

```
In [ ]: chi2, p_value, dof, expected = chi2_contingency(contingency_table)
print(f"Chi-square Statistic: {chi2}")
print(f"P-value: {p_value}")
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies:")
print(expected)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of chi2_contingency test")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of chi2_contingency test")
```

Chi-square Statistic: 25578.381972955114  
P-value: 0.0  
Degrees of Freedom: 8  
Expected Frequencies:  
[[3.02271723e+03 4.43222606e+03 7.67147690e+02 6.54866122e+01  
 3.42241591e+00]  
 [3.74381630e+04 5.48957739e+04 9.50158355e+03 8.11090908e+02  
 4.23886706e+01]  
 [5.46611981e+03 8.01500003e+03 1.38726876e+03 1.18422479e+02  
 6.18891350e+00]]  
As p\_value <= 0.05, We reject the null hypothesis of chi2\_contingency test i.e., protocoltype does influences attack\_category.

Check the assumptions of chi2\_contingency test

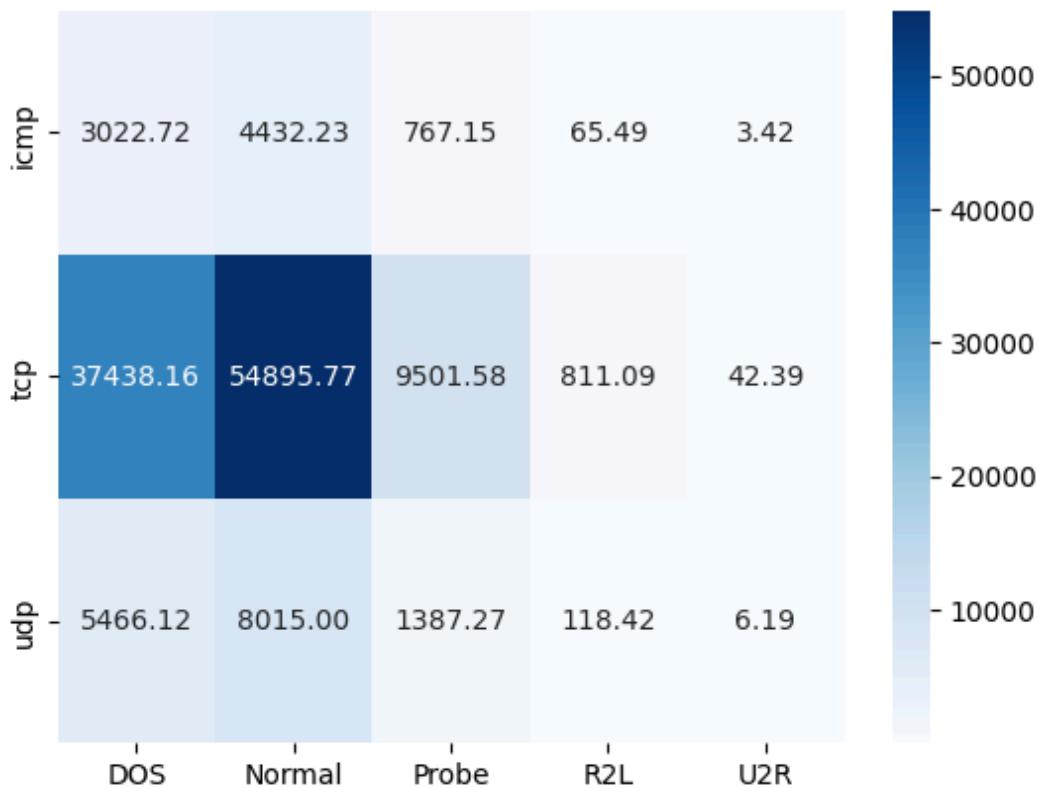
#### Independence

- protocoltype and attack\_or\_normal are independent groups.

#### Expected counts

```
In [ ]: expected_df = pd.DataFrame(expected,
                                    index=contingency_table.index.tolist(),
                                    columns=contingency_table.columns.tolist())
sns.heatmap(expected_df, annot=True, fmt=".2f", cmap="Blues", cbar=True)
```

Out[ ]: <Axes: >



### Observation

tcp-U2R value in expected contingency table is less than 5. But only one out of 15 cells, It is less than 20% of cells (3 cells max). So no need to apply chi2\_contingency with montecarlo simulation (larger table).

chi2\_contingency test suggests that protocol type does influences whether a connection is classified as attack\_or\_normal.

## Role of Service in Network Security

**Does Specific services are targets of network anomalies more often than others?**

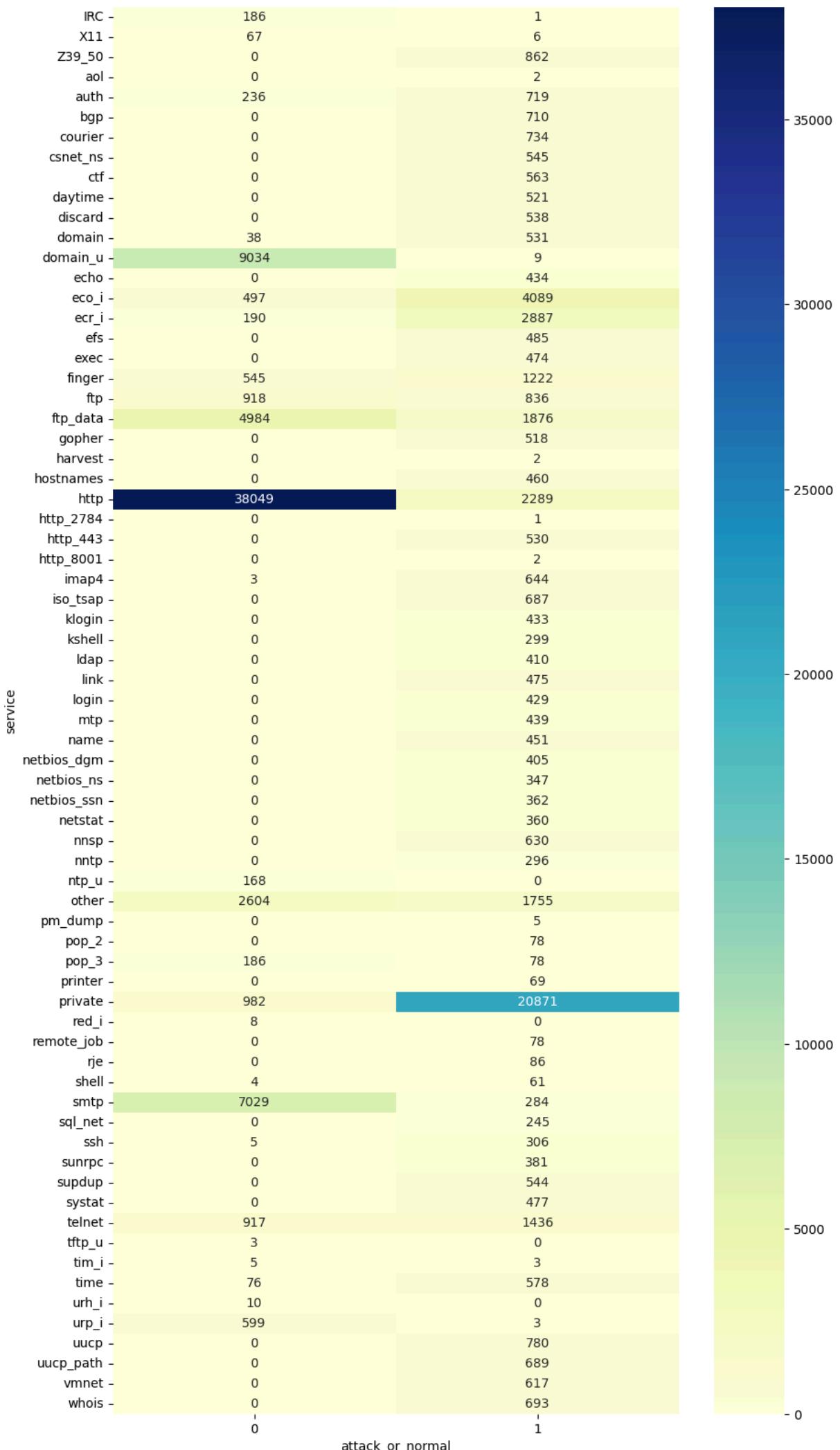
**service vs attack\_or\_normal**

```
In [ ]: # data groups
Service = nadp_add["service"]
Attack_or_normal = nadp_add["attack_or_normal"]
# Create a contingency table
contingency_table = pd.crosstab(Service, Attack_or_normal)
```

**Visual Analysis**

```
In [ ]: fig = plt.figure(figsize=(10,20))
sns.heatmap(contingency_table, annot=True, fmt="d", cmap="YlGnBu")
```

```
Out[ ]: <Axes: xlabel='attack_or_normal', ylabel='service'>
```



## Observation

It is difficult to judge the relation between two categorical features by contingency heat plot. Let's use hypothesis testing.

We can frame alternate hypothesis as service type influences attack\_or\_normal categorization.

## Selection of Appropriate Test

`attack_or_normal` is categorical column with two categories - 0 and 1 (normal and attack). `service` column is providing type of service used in a network connection. It is a categorical column.

So here the independence between two categorical features should be tested.

Comparing observed contingency table with expected contingency table is required. So `chi2_contingency` test is appropriate if atleast 20% cells in expected frequencies > 5. `fisher_exact` test is appropriate if more than 20% cells in expected frequencies < 5 and 2X2 contingency table. For larger table, we should use advanced techniques like `chi2_contingency` along with `montecarlo simulation`.

### chi2\_contingency Hypothesis Formulation

Null Hypothesis H0: service type does not influences whether a connection is classified as attack\_or\_normal.

Alternate Hypothesis Ha: service type does influences whether a connection is classified as attack\_or\_normal.

Significance level: 0.05

```
In [ ]: chi2, p_value, dof, expected = chi2_contingency(contingency_table)
print("Chi-square Statistic: {chi2}")
print("P-value: {p_value}")
print("Degrees of Freedom: {dof}")
print("Expected Frequencies:")
print(expected)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of chi2_contingency test")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of chi2_contingency test")
```

Chi-square Statistic: 93240.03213516614

P-value: 0.0

Degrees of Freedom: 69

Expected Frequencies:

```
[[9.99669850e+01 8.70330150e+01]
 [3.90245449e+01 3.39754551e+01]
 [4.60810380e+02 4.01189620e+02]
 [1.06916561e+00 9.30834385e-01]
 [5.10526581e+02 4.44473419e+02]
 [3.79553793e+02 3.30446207e+02]
 [3.92383781e+02 3.41616219e+02]
 [2.91347630e+02 2.53652370e+02]
 [3.00970121e+02 2.62029879e+02]
 [2.78517643e+02 2.42482357e+02]
 [2.87605550e+02 2.50394450e+02]
 [3.04177617e+02 2.64822383e+02]
 [4.83423233e+03 4.20876767e+03]
 [2.32008938e+02 2.01991062e+02]
 [2.45159675e+03 2.13440325e+03]
 [1.64491130e+03 1.43208870e+03]
 [2.59272662e+02 2.25727338e+02]
 [2.53392251e+02 2.20607749e+02]
 [9.44607821e+02 8.22392179e+02]
 [9.37658244e+02 8.16341756e+02]
 [3.66723806e+03 3.19276194e+03]
 [2.76913894e+02 2.41086106e+02]
 [1.06916561e+00 9.30834385e-01]
 [2.45908091e+02 2.14091909e+02]
 [2.15640013e+04 1.87739987e+04]
 [5.34582807e-01 4.65417193e-01]
 [2.83328888e+02 2.46671112e+02]
 [1.06916561e+00 9.30834385e-01]
 [3.45875076e+02 3.01124924e+02]
 [3.67258389e+02 3.19741611e+02]
 [2.31474356e+02 2.01525644e+02]
 [1.59840259e+02 1.39159741e+02]
 [2.19178951e+02 1.90821049e+02]
 [2.53926834e+02 2.21073166e+02]
 [2.29336024e+02 1.99663976e+02]
 [2.34681852e+02 2.04318148e+02]
 [2.41096846e+02 2.09903154e+02]
 [2.16506037e+02 1.88493963e+02]
 [1.85500234e+02 1.61499766e+02]
 [1.93518976e+02 1.68481024e+02]
 [1.92449811e+02 1.67550189e+02]
 [3.36787169e+02 2.93212831e+02]
 [1.58236511e+02 1.37763489e+02]
 [8.98099116e+01 7.81900884e+01]
 [2.33024646e+03 2.02875354e+03]
 [2.67291404e+00 2.32708596e+00]
 [4.16974590e+01 3.63025410e+01]
 [1.41129861e+02 1.22870139e+02]
 [3.68862137e+01 3.21137863e+01]
 [1.16822381e+04 1.01707619e+04]
 [4.27666246e+00 3.72333754e+00]
 [4.16974590e+01 3.63025410e+01]
 [4.59741214e+01 4.00258786e+01]
 [3.47478825e+01 3.02521175e+01]
 [3.90940407e+03 3.40359593e+03]
 [1.30972788e+02 1.14027212e+02]
 [1.66255253e+02 1.44744747e+02]
 [2.03676050e+02 1.77323950e+02]
 [2.90813047e+02 2.53186953e+02]
 [2.54995999e+02 2.22004001e+02]
 [1.25787335e+03 1.09512665e+03]
 [1.60374842e+00 1.39625158e+00]
 [4.27666246e+00 3.72333754e+00]
 [3.49617156e+02 3.04382844e+02]
 [5.34582807e+00 4.65417193e+00]
 [3.21818850e+02 2.80181150e+02]
 [4.16974590e+02 3.63025410e+02]
 [3.68327554e+02 3.20672446e+02]
 [3.29837592e+02 2.87162408e+02]
 [3.70465886e+02 3.22534114e+02]]
```

As p\_value <= 0.05, We reject the null hypothesis of chi2\_contingency test i.e., service type does influences whether a connection is classified as attack\_or\_normal.

Check the assumptions of chi2\_contingency test

### Independence

service and attack\_or\_normal are independent groups.

### Expected counts

```
In [ ]: fig = plt.figure(figsize=(10,20))
expected_df = pd.DataFrame(expected,
                           index=contingency_table.index.tolist(),
                           columns=contingency_table.columns.tolist())
sns.heatmap(expected_df, annot=True, fmt=".2f", cmap="Blues", cbar=True)
```

Out[ ]: <Axes: >



```
In [ ]: # Calculate the number of cells with counts Less than 5
count_less_than_5 = (expected_df < 5).sum().sum()
Total_cells = expected_df.shape[0]*expected_df.shape[1]
# Print the result
print("Number of cells with count less than 5:", count_less_than_5)
print("Total Number of cells:", Total_cells)
print("20% of Total Number of cells", 0.2*Total_cells)
```

```
Number of cells with count less than 5: 17
Total Number of cells: 140
20% of Total Number of cells 28.0
```

### Observation

17 values in expected contingency table are less than 5. 17 is less than 20% of Total Number of cells (28 max). So no need to apply chi2\_contingency with monte carlo simulation.(larger table)

chi2\_contingency test suggests that service type does influences whether a connection is classified as attack\_or\_normal.

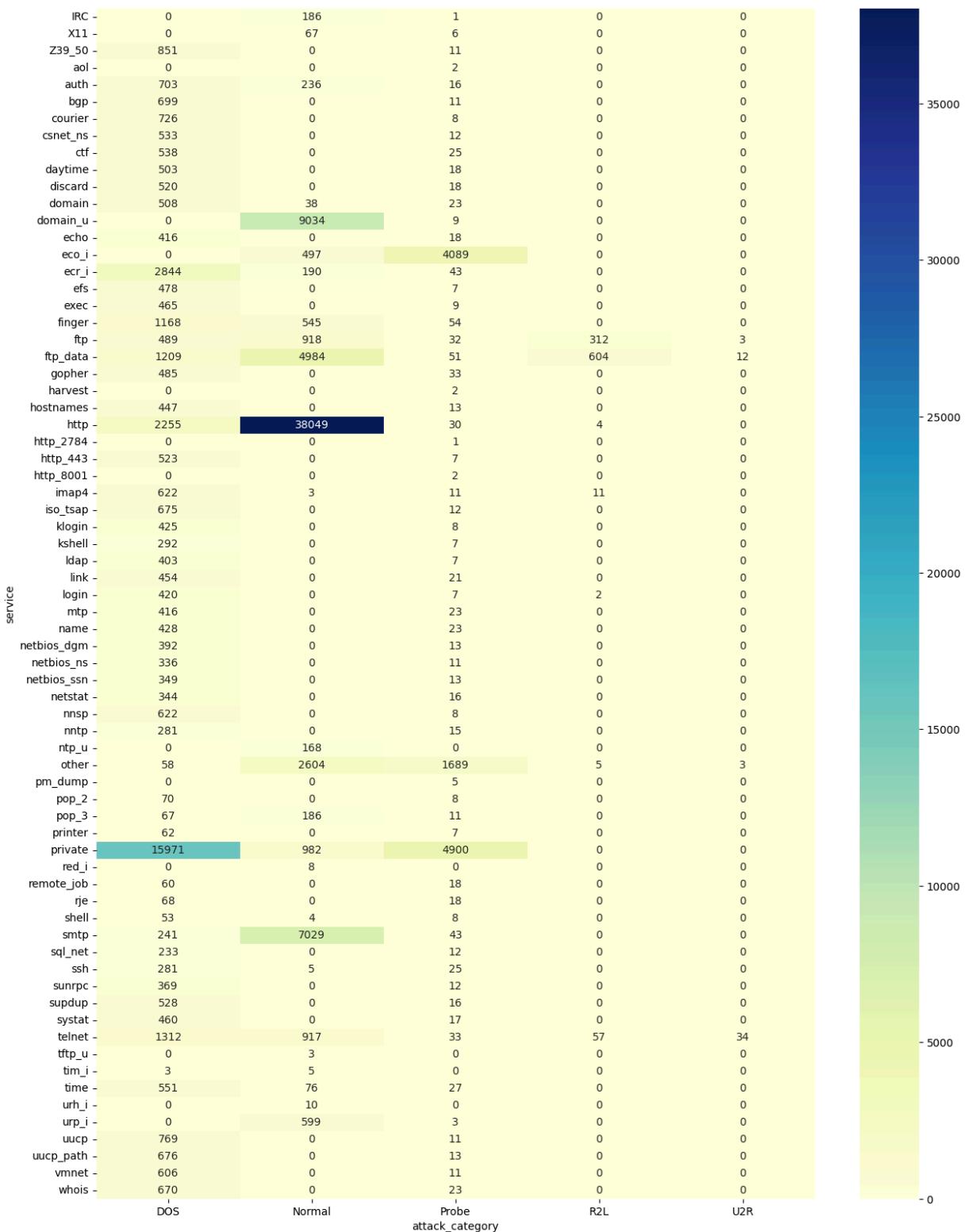
### service vs attack\_categories

```
In [ ]: # data groups
Service = nadp_add["service"]
Attack_category = nadp_add["attack_category"]
# Create a contingency table
contingency_table = pd.crosstab(Service, Attack_category)
```

### Visual Analysis

```
In [ ]: fig = plt.figure(figsize=(15,20))
sns.heatmap(contingency_table, annot=True, fmt="d", cmap="YlGnBu")
```

```
Out[ ]: <Axes: xlabel='attack_category', ylabel='service'>
```



## Observation

It is difficult to judge the relation between two categorical features by contingency heat plot. Let's use hypothesis testing.

We can frame alternate hypothesis as service type influences attack\_category categorization.

## Selection of Appropriate Test

attack\_category is categorical column with five categories. service column is providing type of service used in a network connection. It is a categorical column.

So here the independence between two categorical features should be tested.

Comparing observed contingency table with expected contingency table is required. So chi2\_contingency test is appropriate if atleast 20% cells in expected frequencies > 5. fisher\_exact test is appropriate if more than 20% cells in expected frequencies < 5 and 2X2 contingency table. For larger table,

we should use advanced techniques like `chi2_contingency` along with `montecarlo simulation`.

### chi2\_contingency Hypothesis Formulation

Null Hypothesis H0: service type does not influences attack\_category.

Alternate Hypothesis Ha: service type does influences attack\_category.

Significance level: 0.05

```
In [ ]: chi2, p_value, dof, expected = chi2_contingency(contingency_table)
print(f"Chi-square Statistic: {chi2}")
print(f"P-value: {p_value}")
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies:")
print(expected)
if p_val <= alpha:
    print("As p_value <= 0.05, We reject the null hypothesis of chi2_contingency test")
else:
    print("As p_value > 0.05, We cannot reject the null hypothesis of chi2_contingency test")
```

Chi-square Statistic: 156818.89348191937  
P-value: 0.0  
Degrees of Freedom: 276  
Expected Frequencies:  
[[6.81761092e+01 9.99669850e+01 1.73026918e+01 1.47702285e+00  
7.71911441e-02]  
[2.66142030e+01 3.90245449e+01 6.75452676e+00 5.76591809e-01  
3.01334413e-02]  
[3.14266343e+02 4.60810380e+02 7.97589325e+01 6.80852246e+00  
3.55822279e-01]  
[7.29156248e-01 1.06916561e+00 1.85055528e-01 1.57970359e-02  
8.25573734e-04]  
[3.48172108e+02 5.10526581e+02 8.83640145e+01 7.54308463e+00  
3.94211458e-01]  
[2.58850468e+02 3.79553793e+02 6.56947124e+01 5.60794773e+00  
2.93078676e-01]  
[2.67600343e+02 3.92383781e+02 6.79153787e+01 5.79751217e+00  
3.02985560e-01]  
[1.98695078e+02 2.91347630e+02 5.04276313e+01 4.30469228e+00  
2.24968843e-01]  
[2.05257484e+02 3.00970121e+02 5.20931311e+01 4.44686560e+00  
2.32399006e-01]  
[1.89945203e+02 2.78517643e+02 4.82069650e+01 4.11512784e+00  
2.15061958e-01]  
[1.96143031e+02 2.87605550e+02 4.97799370e+01 4.24940265e+00  
2.22079334e-01]  
[2.07444952e+02 3.04177617e+02 5.26482977e+01 4.49425671e+00  
2.34875727e-01]  
[3.29687997e+03 4.83423233e+03 8.36728569e+02 7.14262977e+01  
3.73283164e+00]  
[1.58226906e+02 2.32008938e+02 4.01570495e+01 3.42795678e+00  
1.79149500e-01]  
[1.67195528e+03 2.45159675e+03 4.24332325e+02 3.62226033e+01  
1.89304057e+00]  
[1.12180689e+03 1.64491130e+03 2.84707929e+02 2.43037397e+01  
1.27014519e+00]  
[1.76820390e+02 2.59272662e+02 4.48759655e+01 3.83078120e+00  
2.00201631e-01]  
[1.72810031e+02 2.53392251e+02 4.38581601e+01 3.74389750e+00  
1.95660975e-01]  
[6.44209545e+02 9.44607821e+02 1.63496559e+02 1.39566812e+01  
7.29394394e-01]  
[6.39470029e+02 9.37658244e+02 1.62293698e+02 1.38540005e+01  
7.24028165e-01]  
[2.50100593e+03 3.66723806e+03 6.34740460e+02 5.41838330e+01  
2.83171791e+00]  
[1.88851468e+02 2.76913894e+02 4.79293817e+01 4.09143229e+00  
2.13823597e-01]  
[7.29156248e-01 1.06916561e+00 1.85055528e-01 1.57970359e-02  
8.25573734e-04]  
[1.67705937e+02 2.45908091e+02 4.25627714e+01 3.63331825e+00  
1.89881959e-01]  
[1.47063524e+04 2.15640013e+04 3.73238494e+03 3.18610417e+02  
1.66509966e+01]  
[3.64578124e-01 5.34582807e-01 9.25277639e-02 7.89851794e-03  
4.12786867e-04]  
[1.93226406e+02 2.83328888e+02 4.90397149e+01 4.18621451e+00  
2.18777040e-01]  
[7.29156248e-01 1.06916561e+00 1.85055528e-01 1.57970359e-02  
8.25573734e-04]  
[2.35882046e+02 3.45875076e+02 5.98654632e+01 5.11034110e+00  
2.67073103e-01]  
[2.50465171e+02 3.67258389e+02 6.35665738e+01 5.42628182e+00  
2.83584578e-01]  
[1.57862328e+02 2.31474356e+02 4.00645218e+01 3.42005827e+00  
1.78736713e-01]  
[1.09008859e+02 1.59840259e+02 2.76658014e+01 2.36165686e+00  
1.23423273e-01]  
[1.49477031e+02 2.19178951e+02 3.79363832e+01 3.23839235e+00  
1.69242615e-01]  
[1.73174609e+02 2.53926834e+02 4.39506878e+01 3.75179602e+00  
1.96073762e-01]  
[1.56404015e+02 2.29336024e+02 3.96944107e+01 3.38846419e+00  
1.77085566e-01]]

```

[1.60049796e+02 2.34681852e+02 4.06196883e+01 3.46744937e+00
 1.81213435e-01]
[1.64424734e+02 2.41096846e+02 4.17300215e+01 3.56223159e+00
 1.86166877e-01]
[1.47654140e+02 2.16506037e+02 3.74737444e+01 3.19889976e+00
 1.67178681e-01]
[1.26508609e+02 1.85500234e+02 3.21071341e+01 2.74078572e+00
 1.43237043e-01]
[1.31977281e+02 1.93518976e+02 3.34950505e+01 2.85926349e+00
 1.49428846e-01]
[1.31248125e+02 1.92449811e+02 3.33099950e+01 2.84346646e+00
 1.48603272e-01]
[2.29684218e+02 3.36787169e+02 5.82924912e+01 4.97606630e+00
 2.60055726e-01]
[1.07915125e+02 1.58236511e+02 2.73882181e+01 2.33796131e+00
 1.22184913e-01]
[6.12491248e+01 8.98099116e+01 1.55446643e+01 1.32695101e+00
 6.93481937e-02]
[1.58919604e+03 2.33024646e+03 4.03328523e+02 3.44296397e+01
 1.79933795e+00]
[1.82289062e+00 2.67291404e+00 4.62638819e-01 3.94925897e-02
 2.06393434e-03]
[2.84370937e+01 4.16974590e+01 7.21716558e+00 6.16084399e-01
 3.21973756e-02]
[9.62486247e+01 1.41129861e+02 2.44273297e+01 2.08520874e+00
 1.08975733e-01]
[2.51558905e+01 3.68862137e+01 6.38441571e+00 5.44997738e-01
 2.84822938e-02]
[7.96712574e+03 1.16822381e+04 2.02200922e+03 1.72606312e+02
 9.02063141e+00]
[2.91662499e+00 4.27666246e+00 7.40222111e-01 6.31881435e-02
 3.30229494e-03]
[2.84370937e+01 4.16974590e+01 7.21716558e+00 6.16084399e-01
 3.21973756e-02]
[3.13537187e+01 4.59741214e+01 7.95738769e+00 6.79272543e-01
 3.54996706e-02]
[2.36975781e+01 3.47478825e+01 6.01430465e+00 5.13403666e-01
 2.68311464e-02]
[2.66615982e+03 3.90940407e+03 6.76655537e+02 5.77618617e+01
 3.01871036e+00]
[8.93216404e+01 1.30972788e+02 2.26693022e+01 1.93513689e+00
 1.01132782e-01]
[1.13383797e+02 1.66255253e+02 2.87761346e+01 2.45643908e+00
 1.28376716e-01]
[1.38904265e+02 2.03676050e+02 3.52530780e+01 3.00933533e+00
 1.57271796e-01]
[1.98330499e+02 2.90813047e+02 5.03351036e+01 4.29679376e+00
 2.24556056e-01]
[1.73903765e+02 2.54995999e+02 4.41357434e+01 3.76759306e+00
 1.96899336e-01]
[8.57852325e+02 1.25787335e+03 2.17717828e+02 1.85852127e+01
 9.71287498e-01]
[1.09373437e+00 1.60374842e+00 2.77583292e-01 2.36955538e-02
 1.23836060e-03]
[2.91662499e+00 4.27666246e+00 7.40222111e-01 6.31881435e-02
 3.30229494e-03]
[2.38434093e+02 3.49617156e+02 6.05131576e+01 5.16563073e+00
 2.69962611e-01]
[3.64578124e+00 5.34582807e+00 9.25277639e-01 7.89851794e-02
 4.12786867e-03]
[2.19476031e+02 3.21818850e+02 5.57017139e+01 4.75490780e+00
 2.48497694e-01]
[2.84370937e+02 4.16974590e+02 7.21716558e+01 6.16084399e+00
 3.21973756e-01]
[2.51194327e+02 3.68327554e+02 6.37516293e+01 5.44207886e+00
 2.84410151e-01]
[2.24944702e+02 3.29837592e+02 5.70896303e+01 4.87338557e+00
 2.54689497e-01]
[2.52652640e+02 3.70465886e+02 6.41217404e+01 5.47367293e+00
 2.86061299e-01]]

```

As p\_value <= 0.05, We reject the null hypothesis of chi2\_contingency test i.e., service type does influences attack\_category.

Check the assumptions of chi2\_contingency test

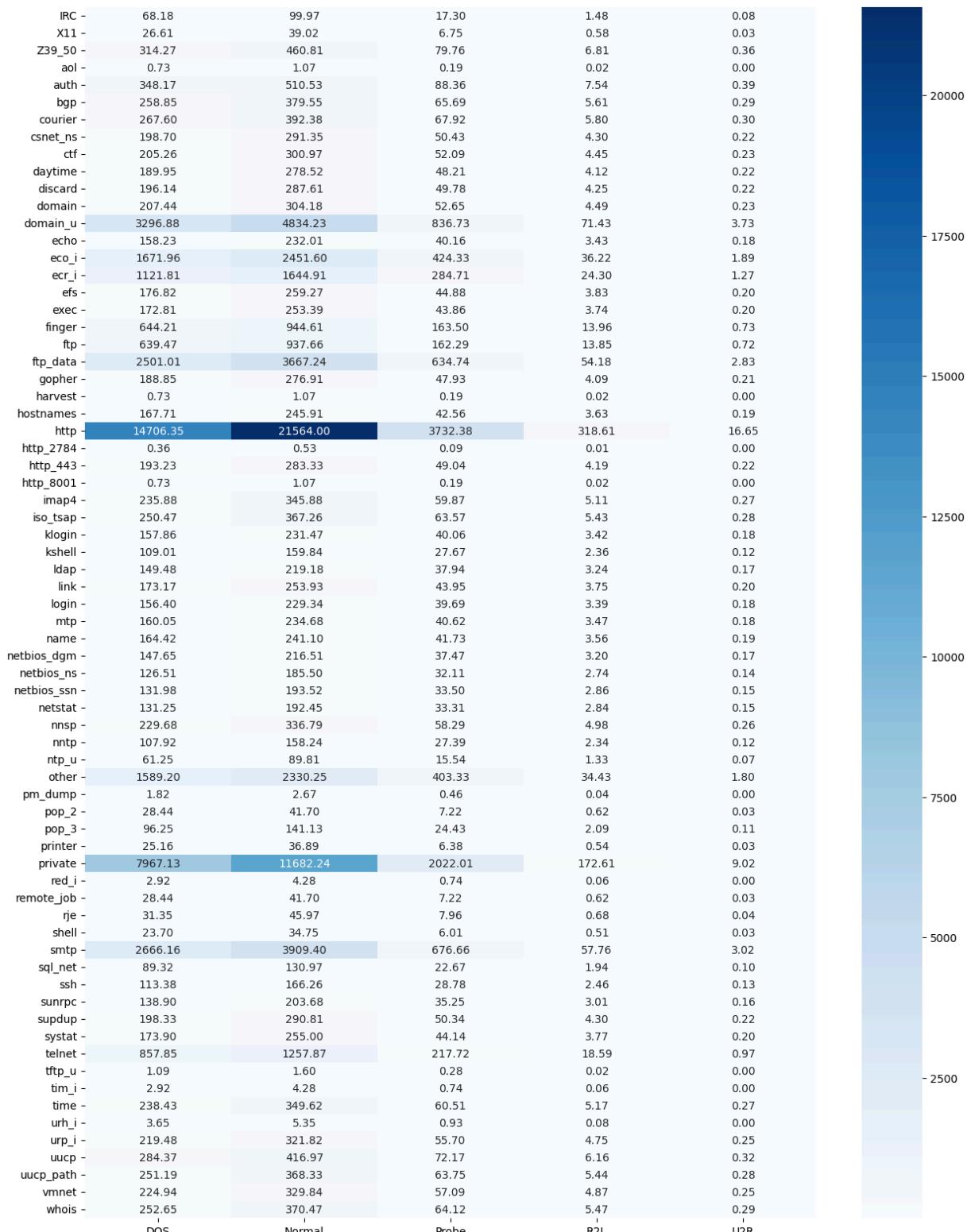
## Independence

service and attack\_category are independent groups.

### Expected counts

```
In [ ]: fig = plt.figure(figsize=(15,20))
expected_df = pd.DataFrame(expected,
                           index=contingency_table.index.tolist(),
                           columns=contingency_table.columns.tolist())
sns.heatmap(expected_df, annot=True, fmt=".2f", cmap="Blues", cbar=True)
```

Out[ ]: <Axes: >



```
In [ ]: # Calculate the number of cells with counts less than 5
count_less_than_5 = (expected_df < 5).sum().sum()
Total_cells = expected_df.shape[0]*expected_df.shape[1]
# Print the result
print("Number of cells with count less than 5:", count_less_than_5)
print("Total Number of cells:", Total_cells)
print("20% of Total Number of cells", 0.2*Total_cells)
```

```
Number of cells with count less than 5: 143
Total Number of cells: 350
20% of Total Number of cells 70.0
```

### Observation

143 values in expected contingency table are less than 5. 143 is greater than 20% of Total Number of cells (70 max). So we need to apply chi2\_contingency with monte carlo simulation.

### chi2\_contingency test with monte carlo simulation

```
In [ ]: def monte_carlo_chi_square(observed, n_simulations=10000):
    # Run chi2_contingency to get observed chi-square and expected frequencies
    chi_square_stat, p_val, dof, expected = chi2_contingency(observed)

    # Store simulated chi-square statistics
    simulated_stats = []

    # Run Monte Carlo simulations
    for _ in range(n_simulations):
        # Generate a random contingency table under the null hypothesis
        simulated_data = np.random.multinomial(observed.sum(), expected.flatten() /

            # Avoid zero cells in the expected array
            expected_nonzero = expected.flatten()
            expected_nonzero[expected_nonzero == 0] = 1e-10 # Small value to prevent ze

        # Calculate chi-square statistic for the simulated table
        simulated_stat = ((simulated_data - expected_nonzero) ** 2 / expected_nonzero).sum()
        simulated_stats.append(simulated_stat)

    # Calculate Monte Carlo p-value
    p_value_mc = np.sum(np.array(simulated_stats) >= chi_square_stat) / n_simulations

    return chi_square_stat, p_value_mc

observed = np.array(contingency_table)

# Perform Monte Carlo chi-square test
chi_square_stat, p_value_mc = monte_carlo_chi_square(observed)

# Print results
print("Chi-square Statistic:", chi_square_stat)
print("Monte Carlo P-value:", p_value_mc)

if p_value_mc <= alpha:
    print("As p_value <= 0.05, we reject the null hypothesis: service type influences attack_category.")
else:
    print("As p_value > 0.05, we cannot reject the null hypothesis: service type does not influence attack_category.)
```

### Observation

As chi2\_contingency assumptions are failed, We have performed chi square test with monte carlo simulation. It suggests that service type influences attack\_category.

```
In [ ]: nadp_add.shape
```

```
Out[ ]: (125973, 63)
```

## Error\_flag vs Anomalies & Urgent vs Anomalies

1. Does Error flags in the Flag feature are significantly associated with anomalies?

## 2. Does Connections that include urgent packets are more likely to be anomalous?

### Error\_flag vs Attack\_or\_normal & Urgent vs Attack\_or\_normal

#### Selection of Appropriate Test

for finding the association between "error\_flag\_or\_not" & "attack\_or\_normal" and "urgent\_or\_not" & "attack\_or\_normal" features, We are applying statsmodels logit method to get p\_value using Maximum Likelihood Estimation method.

```
In [ ]: nadp_logit = nadp_add.copy(deep = True)
# Should create a new flag feature where all flag except "SF" are treated as error if
nadp_logit["error_flag_or_not"] = nadp_logit["flag"].apply(lambda x: 0 if x == "SF"
# Should create a new urgent feature where atleast one urgent activated is 1 and not
nadp_logit["urgent_or_not"] = nadp_logit["urgent"].apply(lambda x: 0 if x == 0 else
```

#### Pre processing required for applying logistic regression

```
In [ ]: # Lets drop flag_category and flag as there is error_flag_or_not feature
# Lets drop urgent as there is urgent_or_not feature
# Lets drop attack and attack_category features as we are taking attack_or_normal feature
# Lets drop lastflag feature also as it is providing the information about success or failure
nadp_logit.drop(["flag_category","attack","attack_category","flag","lastflag","urgent"], axis=1)
```

```
In [ ]: # ENCODING THE CATEGORICAL FEATURES
# Calculate the mean of the target for each category
protocoltype_mean = nadp_logit.groupby('protocoltype')['attack_or_normal'].mean()
service_category_mean = nadp_logit.groupby('service_category')['attack_or_normal'].mean()

# Map the mean encoding back to the original nadp_Logit
nadp_logit['protocoltype'] = nadp_logit['protocoltype'].map(protocoltype_mean)
nadp_logit['service_category'] = nadp_logit['service_category'].map(service_category_mean)

# Get value counts and sort
service_value_counts = nadp_logit['service'].value_counts()

# Create a mapping from category to its rank based on frequency
encoding = {category: rank for rank, category in enumerate(service_value_counts.index)}

# Apply the encoding
nadp_logit['service'] = nadp_logit['service'].map(encoding)
```

```
In [ ]: # SCALING
# Assuming nadp_Logit is your original DataFrame and you want to scale all but the target variable
X = nadp_logit.drop(["attack_or_normal"], axis=1) # Features
y = nadp_logit["attack_or_normal"] # Target variable

# Create a StandardScaler object
scaler = StandardScaler()

# Fit the scaler to the features and transform them
X_scaled = scaler.fit_transform(X)

# Convert the scaled features back to a DataFrame
nadp_logit_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

#### Hypothesis Formulation

##### **error\_flag\_or\_not vs attack\_or\_normal**

Null Hypothesis H0: There is no significant association between error\_flag\_or\_not and attack\_or\_normal.

Alternate Hypothesis Ha: There is a significant association between error\_flag\_or\_not and attack\_or\_normal.

Significance level: 0.05

### urgent\_or\_not vs attack\_or\_normal

Null Hypothesis H0: There is no significant association between urgent\_or\_not and attack\_or\_normal.

Alternate Hypothesis Ha: There is a significant association between urgent\_or\_not and attack\_or\_normal.

Significance level: 0.05

### Applying Statsmodels logit function

```
In [ ]: # Independent variables
X = sm.add_constant(nadp_logit_scaled) # Adds a constant term to the predictor

# Fit the model
model = sm.Logit(y, X)
result = model.fit()

# Print the summary
print(result.summary())

# Hypothesis test decision
alpha = 0.05 # Significance Level

# Checking p_value
error_flag_p_value = result.pvalues['error_flag_or_not']
urgent_p_value = result.pvalues["urgent_or_not"]

if error_flag_p_value < alpha:
    print("Reject H0: There is a significant association between error_flag_or_not and attack_or_normal")
else:
    print("Accept H0: There is no significant association between error_flag_or_not and attack_or_normal")

if urgent_p_value < alpha:
    print("Reject H0: There is a significant association between urgent_or_not and attack_or_normal")
else:
    print("Accept H0: There is no significant association between urgent_or_not and attack_or_normal")
```

Warning: Maximum number of iterations has been exceeded.

Current function value: 0.079344

Iterations: 35

Logit Regression Results

Dep. Variable:	attack_or_normal	No. Observations:	125973		
Model:	Logit	Df Residuals:	125914		
Method:	MLE	Df Model:	58		
Date:	Fri, 16 May 2025	Pseudo R-squ.:	0.8851		
Time:	13:28:15	Log-Likelihood:	-9995.2		
converged:	False	LL-Null:	-87016.		
Covariance Type:	nonrobust	LLR p-value:	0.000		
0.975]					
	coef	std err	z	P> z	[0.025
const	3.7449	4.89e+04	7.66e-05	1.000	-9.58e+04
9.58e+04					
duration	-0.0607	0.020	-3.035	0.002	-0.100
-0.022					
protocoltype	0.5545	0.017	33.475	0.000	0.522
0.587					
service	-0.9373	0.041	-22.653	0.000	-1.018
-0.856					
srcbytes	0.4777	0.145	3.301	0.001	0.194
0.761					
dstbytes	0.5428	0.380	1.427	0.154	-0.203
1.288					
land	-0.0761	0.010	-7.685	0.000	-0.096
-0.057					
wrongfragment	9.1176	5.46e+05	1.67e-05	1.000	-1.07e+06
1.07e+06					
hot	1.1546	0.069	16.719	0.000	1.019
1.290					
numfailedlogins	0.0231	0.010	2.221	0.026	0.003
0.043					
loggedin	-0.1253	0.032	-3.978	0.000	-0.187
-0.064					
numcompromised	24.9253	1.365	18.258	0.000	22.250
27.601					
rootshell	0.0564	0.017	3.363	0.001	0.024
0.089					
suattempted	-0.3098	0.054	-5.736	0.000	-0.416
-0.204					
numroot	-26.5921	1.564	-17.007	0.000	-29.657
-23.527					
numfilecreations	-0.3597	0.051	-7.021	0.000	-0.460
-0.259					
numshells	0.0198	0.010	1.908	0.056	-0.001
0.040					
numaccessfiles	-0.0854	0.039	-2.178	0.029	-0.162
-0.009					
ishostlogin	-0.0678	1659.753	-4.09e-05	1.000	-3253.124
3252.988					
isguestlogin	-1.0299	0.088	-11.650	0.000	-1.203
-0.857					
count	-0.0460	0.178	-0.258	0.797	-0.396
0.304					
srvcount	-31.3377	2.403	-13.040	0.000	-36.048
-26.628					
serrorrate	-0.9588	0.191	-5.022	0.000	-1.333
-0.585					
srvserrorrate	1.4778	0.205	7.193	0.000	1.075
1.880					
rerrorrate	-1.2346	0.126	-9.823	0.000	-1.481
-0.988					
svrerrorrate	1.8047	0.120	15.017	0.000	1.569
2.040					
samesrvrate	-0.3399	0.066	-5.167	0.000	-0.469
-0.211					
diffsrvrate	-0.4556	0.028	-16.352	0.000	-0.510
-0.401					

srvidffhostrate	-0.3607	0.033	-10.797	0.000	-0.426
-0.295					
dsthostcount	1.3061	0.073	17.931	0.000	1.163
1.449					
dsthostsrvcount	-2.2591	0.099	-22.908	0.000	-2.452
-2.066					
dsthostsamesrvrate	1.9754	0.084	23.461	0.000	1.810
2.140					
dsthostdiffsrvrate	0.9846	0.047	21.027	0.000	0.893
1.076					
dsthostsamesrcportrate	0.7968	0.030	26.435	0.000	0.738
0.856					
dsthostsrvdiffhostrate	0.0171	0.021	0.830	0.406	-0.023
0.058					
dsthosterrorrate	0.7827	0.153	5.118	0.000	0.483
1.082					
dsthostsrvserrorrate	1.3422	0.126	10.683	0.000	1.096
1.588					
dsthostrrorrate	-0.7702	0.060	-12.787	0.000	-0.888
-0.652					
dsthostsrvrrorrate	0.6439	0.068	9.448	0.000	0.510
0.778					
service_category	0.7040	0.042	16.843	0.000	0.622
0.786					
serrors_count	0.2558	0.253	1.012	0.311	-0.239
0.751					
rerrors_count	-0.4141	0.281	-1.475	0.140	-0.964
0.136					
samesrv_count	30.5967	2.306	13.269	0.000	26.077
35.116					
diffsrv_count	18.9108	1.109	17.055	0.000	16.738
21.084					
serrors_srvcount	2.9783	0.231	12.897	0.000	2.526
3.431					
rerrors_srvcount	0.3280	0.070	4.690	0.000	0.191
0.465					
srvidffhost_srvcount	3.1862	0.193	16.519	0.000	2.808
3.564					
dsthost_serrors_count	-0.3592	0.166	-2.157	0.031	-0.685
-0.033					
dsthost_rerrors_count	2.2826	0.067	34.190	0.000	2.152
2.413					
dsthost_samesrv_count	0.0959	0.101	0.951	0.342	-0.102
0.294					
dsthost_diffsrv_count	-1.0297	0.052	-19.879	0.000	-1.131
-0.928					
dsthost_serrors_srvcount	-0.2259	0.039	-5.823	0.000	-0.302
-0.150					
dsthost_rerrors_srvcount	-1.4447	0.090	-16.025	0.000	-1.621
-1.268					
dsthost_samesrcport_srvcount	-0.0651	0.027	-2.418	0.016	-0.118
-0.012					
dsthost_srvidffhost_srvcount	0.5851	0.048	12.157	0.000	0.491
0.679					
srcbytes/sec	0.1136	0.011	10.813	0.000	0.093
0.134					
dstbytes/sec	0.0715	0.026	2.743	0.006	0.020
0.123					
error_flag_or_not	0.4461	0.095	4.684	0.000	0.259
0.633					
urgent_or_not	0.0093	0.011	0.852	0.394	-0.012
0.031					
=====					
=====					

Possibly complete quasi-separation: A fraction 0.37 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.  
 Reject H0: There is a significant association between error\_flag\_or\_not and attack\_or\_normal.  
 Accept H0: There is no significant association between urgent\_or\_not and attack\_or\_no rmal.

#### Observation Before applying VIF

There is a significant association between error\_flag\_or\_not and attack\_or\_normal.

There is no significant association between urgent\_or\_not and attack\_or\_normal.

### Check the assumptions of logistic regression

The two-samples independent t-test assume the following characteristics about the data:

**Independence** : The observations in each sample must be independent of each other. This means that the value of one observation should not be related to the value of any other observation in the same sample.

**Binary outcome** : By default, logistic regression assumes that the outcome variable is binary.

**Absence of Multicollinearity** : Multicollinearity corresponds to a situation where the data contain highly correlated independent variables. This is a problem because it reduces the precision of the estimated coefficients, which weakens the statistical power of the logistic regression model.

#### Independence

every sample in nadp\_logit dataset independent of each other.

#### Binary Outcome

`attack_or_normal` feature is Binary feature.

#### Absence of Multi-collinearity

```
In [ ]: def calculate_vif(X):
    vif = pd.DataFrame()
    vif["Features"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vif

def remove_worst_feature(X):
    vif = calculate_vif(X)
    vif["VIF"] = round(vif["VIF"], 2)
    vif = vif.sort_values(by="VIF", ascending=False)

    # Check if all VIF values are Less than 10
    if vif["VIF"].max() < 10:
        return X # Stop if all VIFs are acceptable

    # Identify the worst feature, skipping 'error_flag_or_not' and 'urgent_or_not' i
    for i in range(len(vif)):
        worst_feature = vif["Features"].iloc[i]
        if worst_feature not in ["error_flag_or_not", "urgent_or_not"]:
            print(f"Removing feature: {worst_feature} with VIF: {vif['VIF'].iloc[i]}")
            return remove_worst_feature(X.drop(columns=[worst_feature]))

    # Skip if the worst feature is 'error_flag_or_not' or 'urgent_or_not'
    print(f"Skipping removal of '{worst_feature}' with VIF: {vif['VIF'].iloc[i]}")

    # If only 'error_flag_or_not' and 'urgent_or_not' remain with high VIF, return w
    print("No more removable features with VIF >= 10 that are not 'error_flag_or_no
    return X

# Example usage
X_t = nadp_logit_scaled.copy(deep=True) # Assuming nadp_logit is your original Data
X_reduced = remove_worst_feature(X_t)
```

```
# The reduced dataset will have all VIFs < 10 except for 'error_flag_or_not' and 'ur
print("Final features after VIF removal:", X_reduced.columns)
```

```
Removing feature: numroot with VIF: 888.61
Removing feature: count with VIF: 333.2
Removing feature: srvserrorrate with VIF: 141.11
Removing feature: dsthosterrorrate with VIF: 74.89
Removing feature: serrorrate with VIF: 69.2
Removing feature: srverrorrate with VIF: 64.26
Removing feature: dsthostsrvserrorrate with VIF: 45.15
Removing feature: srvcount with VIF: 36.16
Removing feature: dsthostsamesrvrate with VIF: 29.06
Removing feature: rerrorrate with VIF: 23.4
Removing feature: dsthost_serrors_count with VIF: 19.37
Removing feature: dsthosterrorrate with VIF: 17.82
Removing feature: dsthost_diffsrv_count with VIF: 15.26
Removing feature: samesrvrate with VIF: 12.67
Removing feature: rerrors_count with VIF: 10.07
Final features after VIF removal: Index(['duration', 'protocoltype', 'service', 'srcbytes', 'dstbytes', 'land',
       'wrongfragment', 'hot', 'numfailedlogins', 'loggedin', 'numcompromised',
       'rootshell', 'suattempted', 'numfilecreations', 'numshells',
       'numaccessfiles', 'ishostlogin', 'isguestlogin', 'diffsrvrate',
       'srvdifffhostrate', 'dsthostcount', 'dsthostsrvcount',
       'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
       'dsthostsrvdiffhostrate', 'dsthostsrverrorrate', 'service_category',
       'serrors_count', 'samesrv_count', 'diffsrv_count', 'serrors_srvcount',
       'rerrors_srvcount', 'srvdifffhost_srvcount', 'dsthost_rerrors_count',
       'dsthost_samesrv_count', 'dsthost_serrors_srvcount',
       'dsthost_rerrors_srvcount', 'dsthost_samesrcport_srvcount',
       'dsthost_srvidffhost_srvcount', 'srcbytes/sec', 'dstbytes/sec',
       'error_flag_or_not', 'urgent_or_not'],
      dtype='object')
```

```
In [ ]: vif = calculate_vif(nadp_logit_scaled[X_reduced.columns])
vif["VIF"] = round(vif["VIF"], 2)
vif = vif.sort_values(by="VIF", ascending=False)
vif
```

Out[ ]:

	Features	VIF
21	dsthostsrvcount	8.79
25	dsthostsrverrorrate	8.61
41	error_flag_or_not	8.29
34	dsthost_samesrv_count	8.19
33	dsthost_rerrors_count	7.12
20	dsthostcount	6.11
9	loggedin	4.76
27	serrors_count	4.37
23	dsthostsamesrcportrate	4.20
17	isguestlogin	3.95
7	hot	3.87
37	dsthost_samesrcport_srvcount	3.33
30	serrors_srvcount	3.25
26	service_category	3.05
38	dsthost_svrdiffhost_srvcount	2.94
22	dsthostdiffsvrrate	2.83
31	rerrors_srvcount	2.71
24	dsthostsrvdiffhostrate	2.69
29	diffsrv_count	2.50
32	srvdifffhost_srvcount	2.24
36	dsthost_rerrors_srvcount	2.12
1	protocoltype	2.08
18	diffsrvrate	2.07
35	dsthost_serrors_srvcount	1.97
12	sattempted	1.92
28	samesrv_count	1.74
19	srvdifffhostrate	1.74
11	rootshell	1.58
2	service	1.49
15	numaccessfiles	1.44
0	duration	1.33
10	numcompromised	1.18
6	wrongfragment	1.11
40	dstbytes/sec	1.07
13	numfilecreations	1.04
42	urgent_or_not	1.03
14	numshells	1.02
8	numfailedlogins	1.02
5	land	1.02
39	srcbytes/sec	1.01
3	srcbytes	1.01

	Features	VIF
4	dstbytes	1.00
16	ishostlogin	1.00

Applying statsmodels logit after removing vif > 10 features

```
In [ ]: # Independent variables
X = sm.add_constant(nadp_logit_scaled[X_reduced.columns]) # Adds a constant term to

# Dependent variable
y = y

# Fit the model
model = sm.Logit(y, X)
result = model.fit()

# Print the summary
print(result.summary())

# Hypothesis test decision
alpha = 0.05 # Significance Level

# Checking p_value
error_flag_p_value = result.pvalues['error_flag_or_not']
urgent_p_value = result.pvalues["urgent_or_not"]

if error_flag_p_value < alpha:
    print("Reject H0: There is a significant association between error_flag_or_not and")
else:
    print("Accept H0: There is no significant association between error_flag_or_not and")

if urgent_p_value < alpha:
    print("Reject H0: There is a significant association between urgent_or_not and")
else:
    print("Accept H0: There is no significant association between urgent_or_not and")
```

Warning: Maximum number of iterations has been exceeded.

Current function value: 0.090744

Iterations: 35

Logit Regression Results

Dep. Variable:	attack_or_normal	No. Observations:	125973		
Model:	Logit	Df Residuals:	125929		
Method:	MLE	Df Model:	43		
Date:	Fri, 16 May 2025	Pseudo R-squ.:	0.8686		
Time:	13:48:15	Log-Likelihood:	-11431.		
converged:	False	LL-Null:	-87016.		
Covariance Type:	nonrobust	LLR p-value:	0.000		
0.975]					
	coef	std err	z	P> z	[0.025
const	4.4618	613.360	0.007	0.994	-1197.702
1206.626					
duration	-0.1176	0.017	-7.092	0.000	-0.150
-0.085					
protocoltype	0.5393	0.015	35.014	0.000	0.509
0.570					
service	-0.5214	0.030	-17.448	0.000	-0.580
-0.463					
srcbytes	0.1980	0.098	2.023	0.043	0.006
0.390					
dstbytes	1.0648	0.367	2.900	0.004	0.345
1.784					
land	-0.0378	0.007	-5.104	0.000	-0.052
-0.023					
wrongfragment	6.8601	6854.220	0.001	0.999	-1.34e+04
1.34e+04					
hot	2.2878	0.087	26.329	0.000	2.118
2.458					
numfailedlogins	0.0151	0.010	1.557	0.119	-0.004
0.034					
loggedin	-0.0317	0.027	-1.161	0.246	-0.085
0.022					
numcompromised	-0.1651	0.096	-1.714	0.086	-0.354
0.024					
rootshell	0.0290	0.016	1.864	0.062	-0.001
0.059					
suattempted	-0.1182	0.036	-3.282	0.001	-0.189
-0.048					
numfilecreations	-0.5357	0.097	-5.513	0.000	-0.726
-0.345					
numshells	0.0087	0.009	0.939	0.348	-0.009
0.027					
numaccessfiles	-0.0191	0.037	-0.517	0.605	-0.092
0.053					
ishostlogin	-0.0603	329.987	-0.000	1.000	-646.823
646.702					
isguestlogin	-2.5759	0.111	-23.147	0.000	-2.794
-2.358					
diffsrvrate	-0.3343	0.021	-16.096	0.000	-0.375
-0.294					
srvdiffhostrate	-0.1529	0.027	-5.566	0.000	-0.207
-0.099					
dsthostcount	-0.0730	0.030	-2.403	0.016	-0.133
-0.013					
dsthostsrvcount	-2.5688	0.100	-25.653	0.000	-2.765
-2.373					
dsthostdiffsrvrate	-0.0405	0.019	-2.105	0.035	-0.078
-0.003					
dsthostsamesrcportrate	0.8477	0.024	35.911	0.000	0.801
0.894					
dsthostsrvdiffhostrate	0.0467	0.017	2.787	0.005	0.014
0.079					
dsthostsrvrerrorrate	0.1856	0.037	4.982	0.000	0.113
0.259					
service_category	0.5874	0.036	16.265	0.000	0.517
0.658					

serrors_count	1.4082	0.180	7.841	0.000	1.056
1.760					
samesrv_count	0.4005	0.014	27.848	0.000	0.372
0.429					
diffsrv_count	19.9470	0.848	23.519	0.000	18.285
21.609					
serrors_srvcount	2.6019	0.135	19.305	0.000	2.338
2.866					
rerrors_srvcount	0.4045	0.066	6.161	0.000	0.276
0.533					
srvdifffhost_srvcount	0.7473	0.065	11.458	0.000	0.619
0.875					
dsthost_rerrors_count	1.5019	0.040	37.227	0.000	1.423
1.581					
dsthost_samesrv_count	1.8522	0.083	22.287	0.000	1.689
2.015					
dsthost_serrors_srvcount	0.3242	0.021	15.389	0.000	0.283
0.365					
dsthost_rerrors_srvcount	-1.4893	0.090	-16.461	0.000	-1.667
-1.312					
dsthost_samesrcport_srvcount	-0.0182	0.024	-0.755	0.450	-0.066
0.029					
dsthost_srwdifffhost_srvcount	0.8027	0.051	15.871	0.000	0.704
0.902					
srcbytes/sec	0.0951	0.009	10.374	0.000	0.077
0.113					
dstbytes/sec	0.0534	0.026	2.079	0.038	0.003
0.104					
error_flag_or_not	1.3757	0.039	35.513	0.000	1.300
1.452					
urgent_or_not	0.0205	0.008	2.630	0.009	0.005
0.036					
<hr/>					
<hr/>					

Possibly complete quasi-separation: A fraction 0.29 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

Reject H0: There is a significant association between error\_flag\_or\_not and attack\_or\_normal.

Reject H0: There is a significant association between urgent\_or\_not and attack\_or\_normal.

### Observation After applying VIF

There is a significant association between error\_flag\_or\_not and attack\_or\_normal.

There is significant association between urgent\_or\_not and attack\_or\_normal.

### Quasi-Separation

Quasi separation is observed in above summary. Quasi-separation in logistic regression, particularly in statsmodels' Logit, occurs when a predictor variable almost perfectly predicts the outcome but not in every case. This means there is an alignment where one category of the predictor variable almost exclusively predicts one outcome in the target, though there are some exceptions.

In practical terms, quasi-separation can cause issues because logistic regression coefficients may become extremely large, leading to instability and difficulties in estimating standard errors. This often results in warnings or errors in statistical software, indicating convergence issues.

wrongfragment & ishostlogin has very high coefficient value compared to other feature coefficients. In these cases, Regularization may helpful to mitigate the effect of quasi separation.

# CHAPTER 4: MACHINE LEARNING MODELING FOR BINARY CLASSIFICATION

## Feature Engineering using Unsupervised Algorithms

### Data preprocessing for ML modeling

#### Removing unwanted features and rows

```
In [ ]: # Feature Engineering
nadp_binary = nadp_add.copy(deep=True)

# Remove the attack_category, attack and last_flag features
# Remove hierarchical features like service_category, flag_category
nadp_binary = nadp_binary.drop(["attack_category", "attack", "lastflag", "service_categ

In [ ]: # Checking null values
sum(nadp_binary.isna().sum())

Out[ ]: 0

In [ ]: # Checking duplicates after removing unwanted features
nadp_binary.duplicated().sum()

Out[ ]: np.int64(9)

In [ ]: # Drop duplicates
nadp_binary.drop_duplicates(keep="first", inplace=True)

In [ ]: # shape of nadp_binary
nadp_binary.shape

Out[ ]: (125964, 58)
```

#### Train Test Split and handling duplicates

```
In [ ]: # Separate features and target
nadp_X_binary = nadp_binary.drop(["attack_or_normal"], axis=1) # Features
nadp_y_binary = nadp_binary["attack_or_normal"] # Target variable

# Split the data with stratification
nadp_X_train_binary, nadp_X_test_binary, nadp_y_train_binary, nadp_y_test_binary = train_test_split(nadp_X_binary, nadp_y_binary, test_size=0.2, random_state=42, stratify=nadp_y_binary)

# Verify the value counts in train and test sets
print("Training set value counts:\n", nadp_y_train_binary.value_counts())
print("Test set value counts:\n", nadp_y_test_binary.value_counts())

Training set value counts:
attack_or_normal
0    53874
1    46897
Name: count, dtype: int64
Test set value counts:
attack_or_normal
0    13469
1    11724
Name: count, dtype: int64

In [ ]: # three categorical features
nadp_X_train_binary.describe(include = "object")
```

	protoctype	service	flag
count	100771	100771	100771
unique	3	69	11
top	tcp	http	SF
freq	82113	32343	59966

```
In [ ]: # checking duplicates after train test split
nadp_X_train_binary[nadp_X_train_binary.duplicated(keep=False)]
```

	duration	protoctype	service	flag	srcbytes	dstbytes	land	wrongfragment	urg
37107	0	tcp	finger	S0	0	0	1		0
67588	0	tcp	finger	S0	0	0	1		0
6922	0	icmp	ecr_i	SF	20	0	0		0
121116	0	tcp	finger	S0	0	0	1		0
16515	0	icmp	ecr_i	SF	20	0	0		0
13210	0	tcp	finger	S0	0	0	1		0
72491	0	tcp	finger	S0	0	0	1		0
89324	0	tcp	finger	S0	0	0	1		0

```
In [ ]: # checking duplicates related nadp_y_train_binary
nadp_y_train_binary[nadp_X_train_binary.duplicated(keep=False)]
```

```
Out[ ]: 37107      0
       67588      1
       6922       1
      121116      1
      16515       0
      13210       0
      72491       0
      89324      1
Name: attack_or_normal, dtype: int64
```

```
In [ ]: # REMOVING DUPLICATES WHOSE nadp_y_train_binary == 0 (keeping attacked rows)
# Create a boolean mask for y_train where the value is 0
mask_y_equals_zero = nadp_y_train_binary == 0

# Identify duplicates in X_train where y_train is 0
duplicates_mask = nadp_X_train_binary.duplicated(keep=False)

# Combine both masks to identify the rows to keep
rows_to_keep = nadp_X_train_binary[~(duplicates_mask & mask_y_equals_zero)]

# Remove duplicates from X_train and corresponding values in y_train
nadp_X_train_binary = nadp_X_train_binary.loc[rows_to_keep.index]
nadp_y_train_binary = nadp_y_train_binary.loc[rows_to_keep.index]

# Optionally, reset the index
nadp_X_train_binary.reset_index(drop=True, inplace=True)
nadp_y_train_binary.reset_index(drop=True, inplace=True)
```

```
In [ ]: # Final check of duplicates
nadp_X_train_binary.duplicated().sum()
```

```
Out[ ]: np.int64(0)
```

## Encoding the categorical features

Encode all the category features except service feature with One hot encoding as their nunique is not large.

For service feature, use label encoding in the descending order of their value counts.

Encode Train dataset first, Then use those stats to encode test dataset to avoid target data leak.

## Train Dataset Encoding

```
In [ ]: nadp_X_train_binary_encoded = nadp_X_train_binary.copy(deep=True)

# Initialize OneHotEncoder
ohe_encoder = OneHotEncoder(drop="first", sparse_output=False) # Drop first to avoid

# Fit and transform the selected columns
encoded_data = ohe_encoder.fit_transform(nadp_X_train_binary_encoded[['protocoltpe']])

# Convert to DataFrame with proper column names
encoded_nadp_X_train_binary_encoded = pd.DataFrame(encoded_data, columns=ohe_encoder.get_feature_names_out())

# reset index
nadp_X_train_binary_encoded = nadp_X_train_binary_encoded.reset_index(drop=True)
encoded_nadp_X_train_binary_encoded = encoded_nadp_X_train_binary_encoded.reset_index(drop=True)

# Combine the original DataFrame with the encoded DataFrame
nadp_X_train_binary_encoded = pd.concat([nadp_X_train_binary_encoded.drop(columns=['service']), encoded_nadp_X_train_binary_encoded])

# Get value counts from the training set and create encoding for 'service'
service_value_counts = nadp_X_train_binary_encoded['service'].value_counts()
service_encoding = {category: rank for rank, category in enumerate(service_value_counts.index)}
nadp_X_train_binary_encoded['service'] = nadp_X_train_binary_encoded['service'].map(service_encoding)
```

```
In [ ]: sum(nadp_X_train_binary_encoded.isna().sum())
```

```
Out[ ]: 0
```

```
In [ ]: nadp_X_train_binary_encoded.shape
```

```
Out[ ]: (100767, 67)
```

Storing the ohe\_encoder & service\_encoding into pickle file for future use

```
In [ ]: # Save OneHotEncoder
with open('ohe_encoder.pkl', 'wb') as f:
    pickle.dump(ohe_encoder, f)

# Save service encoding mapping
with open('service_encoding.pkl', 'wb') as f:
    pickle.dump(service_encoding, f)
```

```
In [ ]: nadp_X_train_binary_encoded.columns
```

```
Out[ ]: Index(['duration', 'service', 'srcbytes', 'dstbytes', 'land', 'wrongfragment',  
       'urgent', 'hot', 'numfailedlogins', 'loggedin', 'numcompromised',  
       'rootshell', 'suattempted', 'numroot', 'numfilecreations', 'numshells',  
       'numaccessfiles', 'ishostlogin', 'isguestlogin', 'count', 'srvcount',  
       'serrorrate', 'srvserrorrate', 'rerrorrate', 'srvrerrorrate',  
       'samesrvrate', 'diffsrvrate', 'srvdifffhostrate', 'dsthostcount',  
       'dsthostsrvcount', 'dsthostsamesrvrate', 'dsthostsdifffhostrate',  
       'dsthostsamesrcportrate', 'dsthostsrvdifffhostrate', 'dsthostsserrorrate',  
       'dsthostsrvserrorrate', 'dsthostrerrorrate', 'dsthostsrvrerrorrate',  
       'serrors_count', 'rerrors_count', 'samesrv_count', 'diffsrv_count',  
       'serrors_srvcount', 'rerrors_srvcount', 'srvdifffhost_srvcount',  
       'dsthost_serrors_count', 'dsthost_rerrors_count',  
       'dsthost_samesrv_count', 'dsthost_diffsrv_count',  
       'dsthost_serrors_srvcount', 'dsthost_rerrors_srvcount',  
       'dsthost_samesrcport_srvcount', 'dsthost_srvdifffhost_srvcount',  
       'srcbytes/sec', 'dstbytes/sec', 'protocoltype_tcp', 'protocoltype_udp',  
       'flag_REJ', 'flag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0',  
       'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH'],  
      dtype='object')
```

### Test Dataset Encoding

```
In [ ]: # Assuming nadp_X_test_binary is your test dataset  
nadp_X_test_binary_encoded = nadp_X_test_binary.copy(deep=True)  
  
# Transform 'protocoltype' and 'flag' columns using the fitted OneHotEncoder  
encoded_test_data = ohe_encoder.transform(nadp_X_test_binary_encoded[['protocoltype']]  
encoded_nadp_X_test_binary_encoded = pd.DataFrame(encoded_test_data, columns=ohe_enc  
  
# Reset index for both DataFrames  
encoded_nadp_X_test_binary_encoded = encoded_nadp_X_test_binary_encoded.reset_index()  
nadp_X_test_binary_encoded = nadp_X_test_binary_encoded.reset_index(drop=True)  
  
# Combine the original DataFrame with the encoded DataFrame  
nadp_X_test_binary_encoded = pd.concat([nadp_X_test_binary_encoded.drop(columns=['pr  
  
# Apply frequency encoding for 'service' in the test dataset  
nadp_X_test_binary_encoded['service'] = nadp_X_test_binary_encoded['service'].map(se  
  
# For any new service types in the test dataset that weren't in the training set, as  
max_service_value = nadp_X_train_binary_encoded['service'].max()  
nadp_X_test_binary_encoded['service'].fillna(max_service_value + 1, inplace=True)
```

```
In [ ]: sum(nadp_X_test_binary_encoded.isna().sum())
```

```
Out[ ]: 0
```

```
In [ ]: nadp_X_test_binary_encoded.shape
```

```
Out[ ]: (25193, 67)
```

### Standard Scaling

#### Train Dataset Scaling

```
In [ ]: # SCALING  
# Create a StandardScaler object  
nadp_X_train_binary_scaler = StandardScaler()  
  
# Fit the scaler to the training features and transform them  
nadp_X_train_binary_scaled = nadp_X_train_binary_scaler.fit_transform(nadp_X_train_b  
  
# Convert the scaled training features back to a DataFrame  
nadp_X_train_binary_scaled = pd.DataFrame(nadp_X_train_binary_scaled, columns=nadp_X
```

#### Test Dataset Scaling

```
In [ ]: # Scale the test features using the same scaler  
nadp_X_test_binary_scaled = nadp_X_train_binary_scaler.transform(nadp_X_test_binary_
```

```
# Convert the scaled test features back to a DataFrame
nadp_X_test_binary_scaled = pd.DataFrame(nadp_X_test_binary_scaled, columns=nadp_X_t
```

Storing the nadp\_X\_train\_binary\_scaler into pickle file for future use

```
In [ ]: # Save the scaler to a file
with open('nadp_X_train_binary_scaler.pkl', 'wb') as file:
    pickle.dump(nadp_X_train_binary_scaler, file)
```

## Removing Multi-collinear features using VIF

```
In [ ]: nadp_X_train_binary_scaled.columns
```

```
Out[ ]: Index(['duration', 'service', 'srcbytes', 'dstbytes', 'land', 'wrongfragment',
       'urgent', 'hot', 'numfailedlogins', 'loggedin', 'numcompromised',
       'rootshell', 'suattempted', 'numroot', 'numfilecreations', 'numshells',
       'numaccessfiles', 'ishostlogin', 'isguestlogin', 'count', 'srvcount',
       'serrorrate', 'srvserrorrate', 'rerrorrate', 'svrerrorrate',
       'samesrvrate', 'diffsrvrate', 'srvdifffhostrate', 'dsthostcount',
       'dsthostsrvcount', 'dsthostsamesrvrate', 'dsthostdiffsrvrate',
       'dsthostsamesrcportrate', 'dsthostsrvdiffhostrate', 'dsthosterrorrate',
       'dsthostsrvserrorrate', 'dsthosterrorrate', 'dsthostsrverrorrate',
       'serrors_count', 'rerrors_count', 'samesrv_count', 'diffsrv_count',
       'serrors_srvcount', 'rerrors_srvcount', 'srvdifffhost_srvcount',
       'dsthost_serrors_count', 'dsthost_rerrors_count',
       'dsthost_samesrv_count', 'dsthost_diffsrv_count',
       'dsthost_serrors_srvcount', 'dsthost_rerrors_srvcount',
       'dsthost_samesrcport_srvcount', 'dsthost_srvidffhost_srvcount',
       'srcbytes/sec', 'dstbytes/sec', 'protocoltype_tcp', 'protocoltype_udp',
       'flag_REJ', 'flag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0',
       'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH'],
      dtype='object')
```

```
In [ ]: def calculate_vif(X):
    vif = pd.DataFrame()
    vif["Features"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vif

def remove_worst_feature(X):
    vif = calculate_vif(X)
    vif["VIF"] = round(vif["VIF"], 2)
    vif = vif.sort_values(by="VIF", ascending=False)

    # Check if all VIF values are less than 10
    if vif["VIF"].max() < 10:
        return X # Stop if all VIFs are acceptable

    # Remove the feature with the highest VIF
    worst_feature = vif["Features"].iloc[0]
    print(f"Removing feature: {worst_feature} with VIF: {vif['VIF'].iloc[0]}")

    # Recursively call the function with the reduced dataset
    return remove_worst_feature(X.drop(columns=[worst_feature]))

# VIF should be applied only among continuous features
X_t = nadp_X_train_binary_scaled[['duration', 'srcbytes', 'dstbytes', 'wrongfragment',
       'urgent', 'hot', 'numfailedlogins', 'numcompromised', 'numroot', 'numfilecrea
       'numaccessfiles', 'count', 'srvcount', 'serrorrate', 'srvserrorrate', 'rerror
       'samesrvrate', 'diffsrvrate', 'srvdifffhostrate', 'dsthostcount',
       'dsthostsrvcount', 'dsthostsamesrvrate', 'dsthostdiffsrvrate',
       'dsthostsamesrcportrate', 'dsthostsrvdiffhostrate', 'dsthosterrorrate',
       'dsthostsrvserrorrate', 'dsthosterrorrate', 'dsthostsrverrorrate',
       'serrors_count', 'rerrors_count', 'samesrv_count', 'diffsrv_count',
       'serrors_srvcount', 'rerrors_srvcount', 'srvdifffhost_srvcount',
       'dsthost_serrors_count', 'dsthost_rerrors_count',
       'dsthost_samesrv_count', 'dsthost_diffsrv_count',
       'dsthost_serrors_srvcount', 'dsthost_rerrors_srvcount',
       'dsthost_samesrcport_srvcount', 'dsthost_srvidffhost_srvcount',
       'srcbytes/sec', 'dstbytes/sec']]
```

```
VIF_reduced = remove_worst_feature(X_t)
```

```
# The reduced dataset will have all VIFs < 10
print("Final features after VIF removal:", VIF_reduced.columns)

Removing feature: numroot with VIF: 1114.22
Removing feature: count with VIF: 358.15
Removing feature: srvserrorrate with VIF: 128.21
Removing feature: dsthosterrorrate with VIF: 72.32
Removing feature: svrerrorrate with VIF: 63.41
Removing feature: dsthostsrverrorrate with VIF: 48.22
Removing feature: srvcount with VIF: 32.0
Removing feature: dsthostsamesrvrate with VIF: 28.68
Removing feature: dsthost_serrors_count with VIF: 23.15
Removing feature: dsthostrrorrate with VIF: 20.46
Removing feature: dsthostsrvrrorrate with VIF: 18.7
Removing feature: dsthost_diffsrv_count with VIF: 14.85
Removing feature: samesrvrate with VIF: 13.5
Final features after VIF removal: Index(['duration', 'srcbytes', 'dstbytes', 'wrongfragment', 'urgent', 'hot',
       'numfailedlogins', 'numcompromised', 'numfilecreations', 'numshells',
       'numaccessfiles', 'serrorrate', 'rerrorrate', 'diffsrvrate',
       'srvdifffhostrate', 'dsthostcount', 'dsthostsrvcount',
       'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
       'dsthostsrvdiffhostrate', 'serrors_count', 'rerrors_count',
       'samesrv_count', 'diffsrv_count', 'serrors_srvcount',
       'rerrors_srvcount', 'srvdifffhost_srvcount', 'dsthost_rerrors_count',
       'dsthost_samesrv_count', 'dsthost_serrors_srvcount',
       'dsthost_rerrors_srvcount', 'dsthost_samesrcport_srvcount',
       'dsthost_srvdiffhost_srvcount', 'srcbytes/sec', 'dstbytes/sec'],
      dtype='object')
```

In [ ]: nadp\_X\_test\_binary\_scaled.shape

Out[ ]: (25193, 67)

```
In [ ]: VIF_reduced_columns = ['duration', 'srcbytes', 'dstbytes', 'wrongfragment', 'urgent',
       'numfailedlogins', 'numcompromised', 'numfilecreations', 'numshells',
       'numaccessfiles', 'serrorrate', 'rerrorrate', 'diffsrvrate',
       'srvdifffhostrate', 'dsthostcount', 'dsthostsrvcount',
       'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
       'dsthostsrvdiffhostrate', 'serrors_count', 'rerrors_count',
       'samesrv_count', 'diffsrv_count', 'serrors_srvcount',
       'rerrors_srvcount', 'srvdifffhost_srvcount', 'dsthost_rerrors_count',
       'dsthost_samesrv_count', 'dsthost_serrors_srvcount',
       'dsthost_rerrors_srvcount', 'dsthost_samesrcport_srvcount',
       'dsthost_srvdiffhost_srvcount', 'srcbytes/sec', 'dstbytes/sec']
cat_features = ['flag_REJ', 'flag_RST0', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0',
                'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH', 'isguestlogin',
                'ishostlogin', 'land', 'loggedin', 'protocoltype_tcp', 'protocoltype_rootshell',
                'service', 'suattempted']
final_selected_features = VIF_reduced_columns + cat_features
print(f"Number of final_selected_features : {len(final_selected_features)}")
print(f"Number of features removed by VIF : {nadp_X_train_binary_scaled.shape[1] - 1}
```

Number of final\_selected\_features : 54

Number of features removed by VIF : 13

```
In [ ]: vif = calculate_vif(nadp_X_train_binary_scaled[VIF_reduced_columns])
vif["VIF"] = round(vif["VIF"], 2)
vif = vif.sort_values(by="VIF", ascending=False)
vif
```

Out[ ]:

	Features	VIF
21	errors_count	9.81
28	dsthost_samesrv_count	8.13
11	serrorrate	7.85
27	dsthost_errors_count	7.85
16	dsthostsrvcount	7.49
23	diffsrv_count	6.51
15	dsthostcount	5.94
12	rerrorrate	5.39
20	serrors_count	4.77
18	dsthostsamesrcportrate	4.06
24	serrors_srvcount	3.29
31	dsthost_samesrcport_srvcount	3.20
25	errors_srvcount	3.18
32	dsthost_svrdiffhost_srvcount	2.89
17	dsthostdiffsvrrate	2.87
19	dsthostsrvdiffhostrate	2.57
26	svrdiffhost_srvcount	2.09
29	dsthost_serrors_srvcount	2.03
13	diffsrvrate	2.03
30	dsthost_errors_srvcount	1.80
14	svrdiffhostrate	1.72
0	duration	1.39
22	samesrv_count	1.36
7	numcompromised	1.11
10	numaccessfiles	1.10
3	wrongfragment	1.06
5	hot	1.06
34	dstbytes/sec	1.05
8	numfilecreations	1.03
4	urgent	1.02
6	numfailedlogins	1.02
33	srcbytes/sec	1.01
1	srcbytes	1.01
2	dstbytes	1.00
9	numshells	1.00

In [ ]: `# Filter both the training and test datasets to keep only the selected features  
nadp_X_train_binary_final = nadp_X_train_binary_scaled[final_selected_features]  
nadp_X_test_binary_final = nadp_X_test_binary_scaled[final_selected_features]  
nadp_y_train_binary_final = nadp_y_train_binary.copy(deep = True)  
nadp_y_test_binary_final = nadp_y_test_binary.copy(deep = True)`

In [ ]: `# Saving final preprocessed dataframes to csv  
nadp_X_train_binary_final.to_csv("nadp_X_train_binary_final",index=False)`

```
nadp_X_test_binary_final.to_csv("nadp_X_test_binary_final",index=False)
nadp_y_train_binary_final.to_csv("nadp_y_train_binary_final",index=False)
nadp_y_test_binary_final.to_csv("nadp_y_test_binary_final",index=False)
```

```
In [ ]: # Loading the final preprocessed dataframes
nadp_X_train_binary_final = pd.read_csv("./nadp_X_train_binary_final")
nadp_X_test_binary_final = pd.read_csv("./nadp_X_test_binary_final")
nadp_y_train_binary_final = pd.read_csv("./nadp_y_train_binary_final")
nadp_y_test_binary_final = pd.read_csv("./nadp_y_test_binary_final")
```

```
In [ ]: nadp_X_train_binary_final.to_pickle('nadp_X_train_binary_final.pkl', compression='gz')
```

## Visualise the nadp\_X\_train\_binary\_final and nadp\_X\_test\_binary\_final ground truth groups using UMAP

### create\_binary\_umap function

```
In [ ]: binary_cmap = ListedColormap(sns.husl_palette(len(np.unique(nadp_y_train_binary_final)), 10))

In [ ]: binary_train_umap = UMAP(init='random',n_neighbors=200,min_dist=0.1, random_state=42)
binary_test_umap = UMAP(init='random',n_neighbors=200,min_dist=0.1, random_state=42,)

In [ ]: np.save("binary_train_umap.npy",binary_train_umap)
np.save("binary_test_umap.npy",binary_test_umap)

In [ ]: binary_train_umap = np.load("binary_train_umap.npy")
binary_test_umap = np.load("binary_test_umap.npy")

In [ ]: def create_binary_umap(binary_train_umap,nadp_y_train_binary_final,binary_test_umap,
    # Create a figure with 1 row and 2 columns
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    # Plot the training data
    im_train = axs[0].scatter(binary_train_umap[:, 0],
                            binary_train_umap[:, 1],
                            s=25,
                            c=np.array(nadp_y_train_binary_final),
                            cmap=binary_cmap,
                            edgecolor='none')
    axs[0].set_title('Train Data UMAP')
    axs[0].set_xlabel('UMAP Dimension 1')
    axs[0].set_ylabel('UMAP Dimension 2')

    # Plot the test data
    im_test = axs[1].scatter(binary_test_umap[:, 0],
                            binary_test_umap[:, 1],
                            s=25,
                            c=np.array(nadp_y_test_binary_final),
                            cmap=binary_cmap,
                            edgecolor='none')
    axs[1].set_title('Test Data UMAP')
    axs[1].set_xlabel('UMAP Dimension 1')
    axs[1].set_ylabel('UMAP Dimension 2')
    # Add colorbar for training data
    cbar_train = fig.colorbar(im_train, ax=axs[1], label='attack_or_normal')

    # Display the plots
    plt.tight_layout()
    plt.savefig(f"{run_name}_umap.png")
    mlflow.log_artifact(f"{run_name}_umap.png")
    plt.show()
```

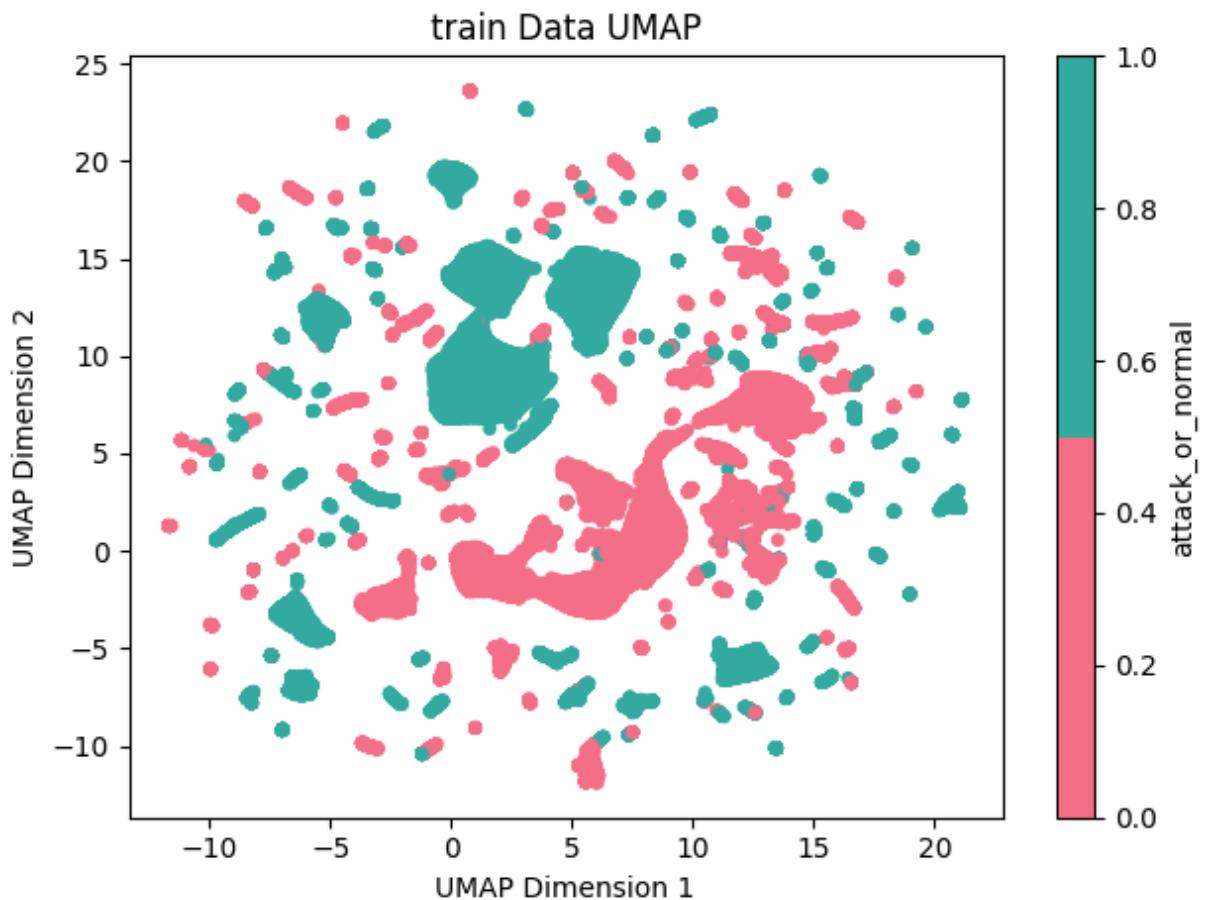
```
In [ ]: fig = plt.figure()
im_train = plt.scatter(binary_train_umap[:, 0],
                      binary_train_umap[:, 1],
                      s=25,
                      c=np.array(nadp_y_train_binary_final),
                      cmap=binary_cmap,
```

```

        edgecolor='none')
plt.title('train Data UMAP')
plt.xlabel('UMAP Dimension 1')
plt.ylabel('UMAP Dimension 2')
# Add colorbar for training data
cbar_train = fig.colorbar(im_train, label='attack_or_normal')

# Display the plots
plt.tight_layout()
plt.show()

```



In [ ]: `binary_test_umap[:,0].min()`

Out[ ]: `np.float32(5.2507334)`

### Original nadp\_binary\_ground\_truth\_umap

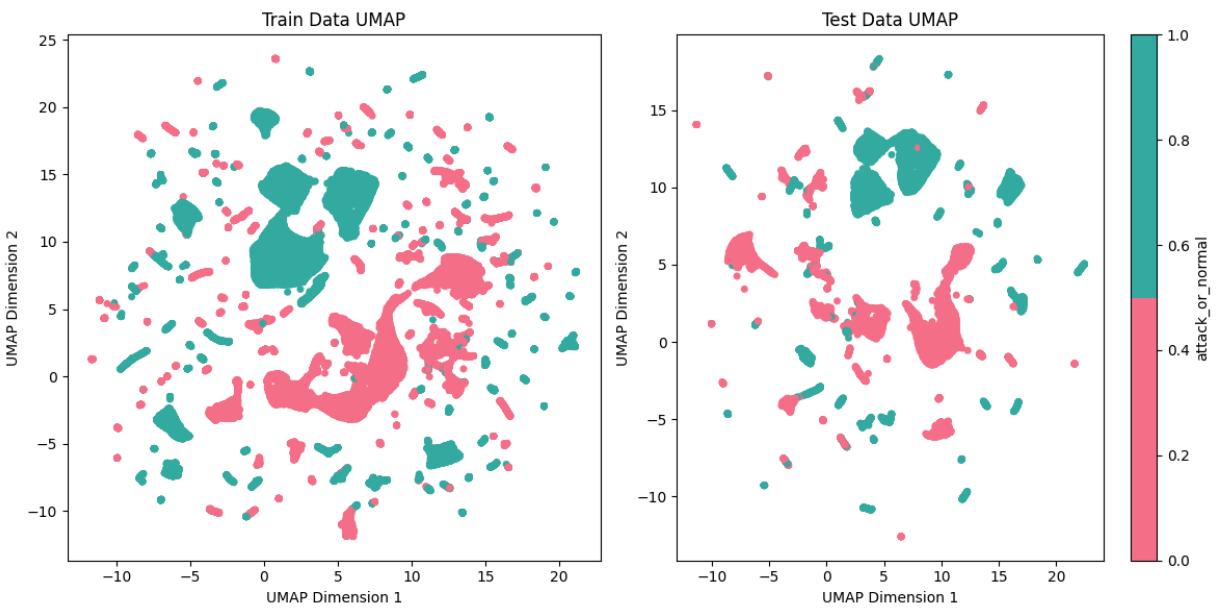
```

# Logging the binary_ground_truth_umap.png artifact into mlflow
experiment_name = "nadp_binary"
run_name = "binary_ground_truth_umap"
#mlflow.create_experiment("nadp_binary")
mlflow.set_experiment(experiment_name)

with mlflow.start_run(run_name=run_name):
    try:
        # Log umap
        create_binary_umap(binary_train_umap,nadp_y_train_binary_final,binary_test_u
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")

```

2025/05/16 14:13:34 INFO mlflow.tracking.fluent: Experiment with name 'nadp\_binary' does not exist. Creating a new experiment.



MLFLOW Logging is completed

Time taken to execute

binary\_ground\_truth\_umap: 10 s

## Creating additional features using scores from Anomaly Detection Algorithms (Unsupervised)

```
In [ ]: # Creating new dataframes to store the anomaly_scores
nadp_X_train_binary_anomaly = nadp_X_train_binary_final.copy(deep = True)
nadp_X_test_binary_anomaly = nadp_X_test_binary_final.copy(deep = True)
```

### Local Outlier Factor

```
In [ ]: # Logging the binary_lof_umap.png artifact into mlflow
experiment_name = "nadp_binary"
run_name = "binary_lof"
#mlflow.create_experiment("nadp_binary")
mlflow.set_experiment(experiment_name)

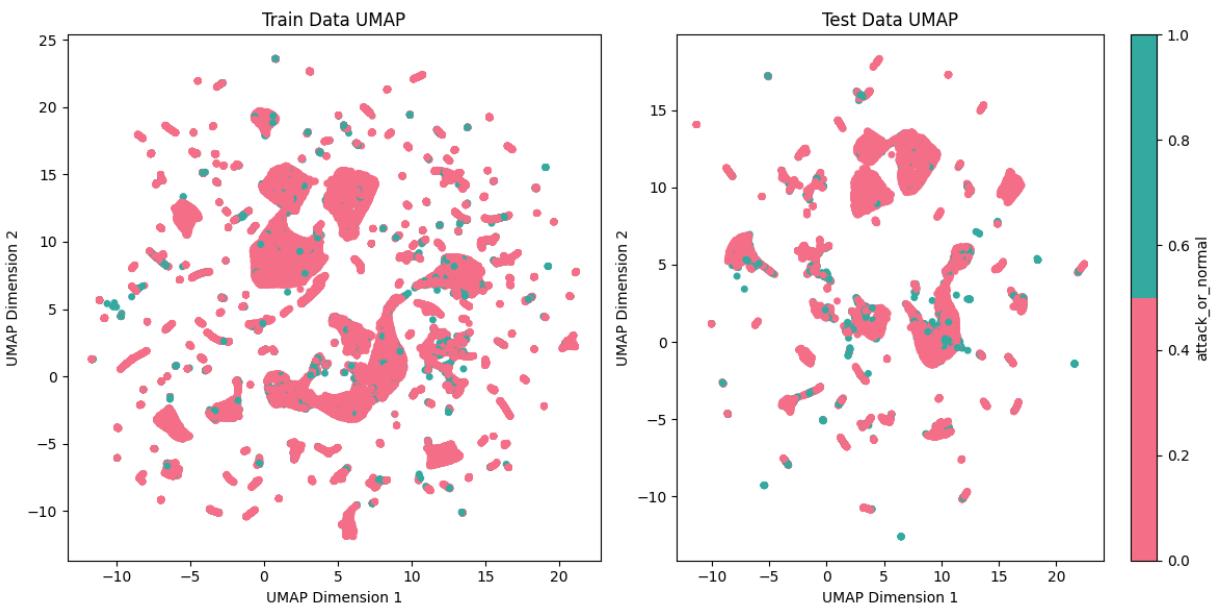
with mlflow.start_run(run_name=run_name):
    try:
        # Log params
        params = {"n_neighbors":20,"contamination":"auto","n_jobs": -1}
        mlflow.log_params(params)

        # Train dataset
        train_binary_lof = LocalOutlierFactor(**params)
        X_train_binary_lof_labels = train_binary_lof.fit_predict(nadp_X_train_binary)
        X_train_binary_lof_labels = np.where(X_train_binary_lof_labels == -1,1,0)
        nadp_X_train_binary_anomaly["binary_lof_nof"] = train_binary_lof.negative_outliers_
        train_metrics = {"actual_n_neighbors":train_binary_lof.n_neighbors_,"offset":train_binary_lof.offset_}
        mlflow.log_metrics(train_metrics)

        # Test dataset
        test_binary_lof = LocalOutlierFactor(**params)
        X_test_binary_lof_labels = test_binary_lof.fit_predict(nadp_X_test_binary)
        X_test_binary_lof_labels = np.where(X_test_binary_lof_labels == -1,1,0)
        nadp_X_test_binary_anomaly["binary_lof_nof"] = test_binary_lof.negative_outliers_
        test_metrics = {"actual_n_neighbors":test_binary_lof.n_neighbors_,"offset":test_binary_lof.offset_}
        mlflow.log_metrics(test_metrics)

        # Log umap
        create_binary_umap(binary_train_umap,X_train_binary_lof_labels,binary_test_u

        # Log the model
        mlflow.sklearn.log_model(train_binary_lof, f"{run_name}_train_model")
        mlflow.sklearn.log_model(test_binary_lof, f"{run_name}_test_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")
```



```
2025/05/16 16:22:59 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/16 16:23:16 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto in
fer the model signature.
2025/05/16 16:23:17 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/16 16:23:29 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto in
fer the model signature.
```

MLFLOW Logging is completed

Time taken to execute

binary\_lof: 3 min

## Isolation forest

```
In [ ]: # Logging the binary_iforest_umap.png artifact into mlflow
experiment_name = "nadp_binary"
run_name = "binary_iforest"
#mlflow.create_experiment("nadp_binary")
mlflow.set_experiment(experiment_name)

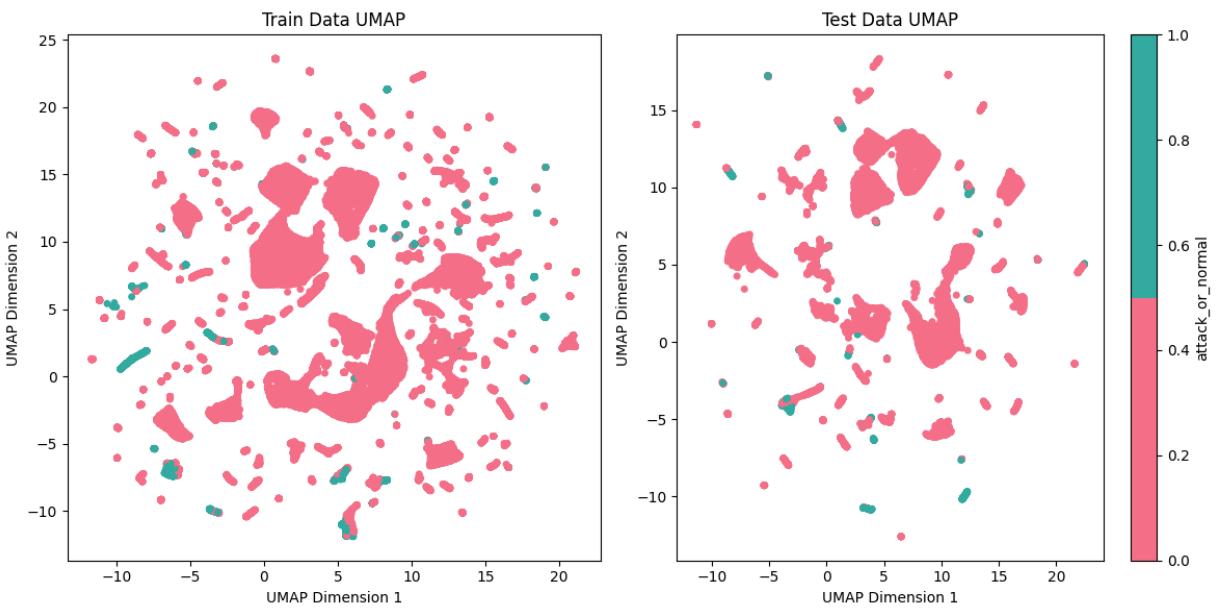
with mlflow.start_run(run_name=run_name):
    try:
        # Log params
        params = {"n_estimators":100,"contamination":"auto","n_jobs": -1,"random_st
        mlflow.log_params(params)

        # Train dataset
        train_binary_iforest = IsolationForest(**params)
        X_train_binary_iforest_labels = train_binary_iforest.fit_predict(nadp_X_trai
        X_train_binary_iforest_labels = np.where(X_train_binary_iforest_labels == -1
        nadp_X_train_binary_anomaly["binary_iforest_df"] = train_binary_iforest.decis
        train_metrics = {"n_estimators":len(train_binary_iforest.estimators_), "offse
        mlflow.log_metrics(train_metrics)

        # Test dataset
        X_test_binary_iforest_labels = train_binary_iforest.predict(nadp_X_test_bina
        X_test_binary_iforest_labels = np.where(X_test_binary_iforest_labels == -1, 1
        nadp_X_test_binary_anomaly["binary_iforest_df"] = train_binary_iforest.decisi

        # Log umap
        create_binary_umap(binary_train_umap,X_train_binary_iforest_labels,binary_te

        # Log the model
        mlflow.sklearn.log_model(train_binary_iforest, f"{run_name}_train_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")
```



```
2025/05/16 16:24:03 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

Time taken to execute

binary\_iforest: 26 s

```
In [ ]: params = {"n_estimators":100,"contamination":"auto","n_jobs": -1,"random_state":42,"train_binary_iforest = IsolationForest(**params)
train_binary_iforest.fit(nadp_X_train_binary_final)
```

```
Out[ ]: ▾ IsolationForest ⓘ ? IsolationForest(n_jobs=-1, random_state=42)
```

```
In [ ]: with open("train_binary_iforest.pkl","wb") as f:
pickle.dump(train_binary_iforest,f)
```

## Elliptic Envelope

```
# Logging the binary_robust_cov_umap.png artifact into mlflow
experiment_name = "nadp_binary"
run_name = "binary_robust_cov"
#mlflow.create_experiment("nadp_binary")
mlflow.set_experiment(experiment_name)

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

        # Log params
        params = {"contamination":0.1,"random_state":42}
        mlflow.log_params(params)

        # Train dataset
        train_binary_robust_cov = EllipticEnvelope(**params)
        X_train_binary_robust_cov_labels = train_binary_robust_cov.fit_predict(nadp_X_train_binary_anomaly["binary_robust_cov_df"])
        nadp_X_train_binary_anomaly["binary_robust_cov_df"] = train_binary_robust_cov
        train_metrics = {"offset":train_binary_robust_cov.offset_}
        mlflow.log_metrics(train_metrics)

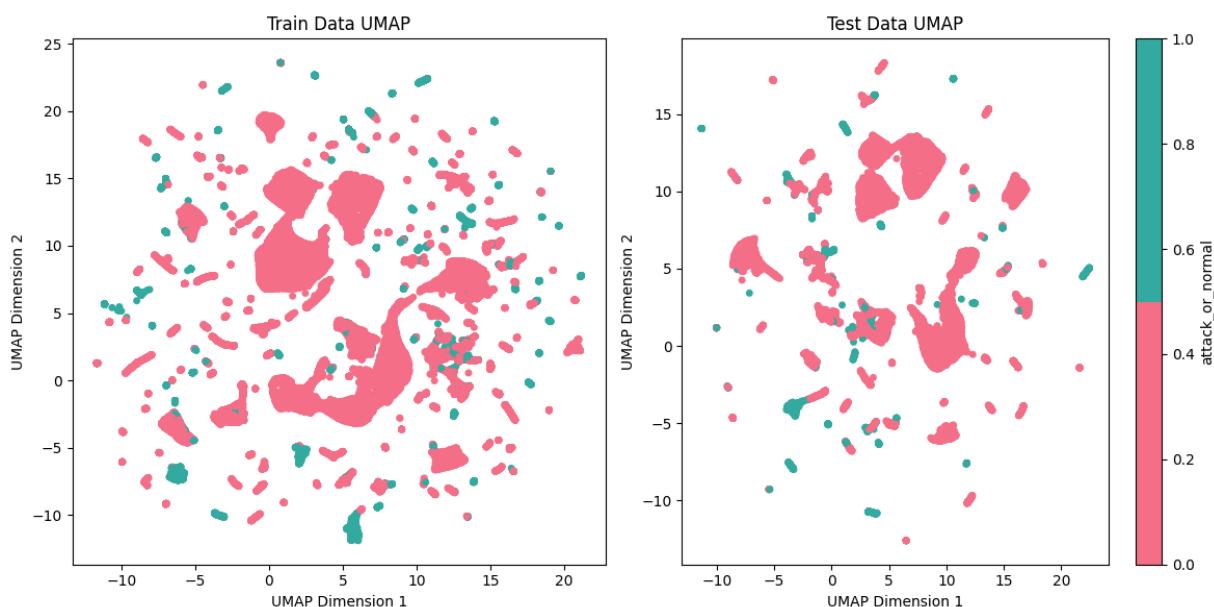
        # Test dataset
        X_test_binary_robust_cov_labels = train_binary_robust_cov.predict(nadp_X_test_binary_anomaly["binary_robust_cov_df"])
        nadp_X_test_binary_anomaly["binary_robust_cov_df"] = train_binary_robust_cov

        # Log umap
        create_binary_umap(binary_train_umap,X_train_binary_robust_cov_labels,binary
```

```

# Log the model
mlflow.sklearn.log_model(train_binary_robust_cov, f"{run_name}_train_model")
print("MLFLOW Logging is completed")
except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

```



2025/05/16 16:25:20 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging is completed

Time taken to execute

binary\_robust\_cov: 1 min

```
In [ ]: params = {"contamination":0.1,"random_state":42}
train_binary_robust_cov = EllipticEnvelope(**params)
train_binary_robust_cov = train_binary_robust_cov.fit(nadp_X_train_binary_final)
```

```
In [ ]: with open("train_binary_robust_cov.pkl","wb") as f:
pickle.dump(train_binary_robust_cov,f)
```

## One Class SVM

```
# Logging the binary_one_class_svm_umap.png artifact into mlflow
experiment_name = "nadp_binary"
run_name = "binary_one_class_svm"
#mlflow.create_experiment("nadp_binary")
mlflow.set_experiment(experiment_name)

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

        # log params
        params = {"nu":0.1, "verbose":0}
        mlflow.log_params(params)

        # Train dataset
        train_binary_one_class_svm = OneClassSVM(**params)
        X_train_binary_one_class_svm_labels = train_binary_one_class_svm.fit_predict(X_train_binary_one_class_svm)
        X_train_binary_one_class_svm_labels = np.where(X_train_binary_one_class_svm_labels < 0, -1, 1)
        nadp_X_train_binary_anomaly["binary_one_class_svm_df"] = train_binary_one_class_svm
        train_metrics = {"offset":train_binary_one_class_svm.offset_, "n_support_vectors":train_binary_one_class_svm.n_support_}
        mlflow.log_metrics(train_metrics)

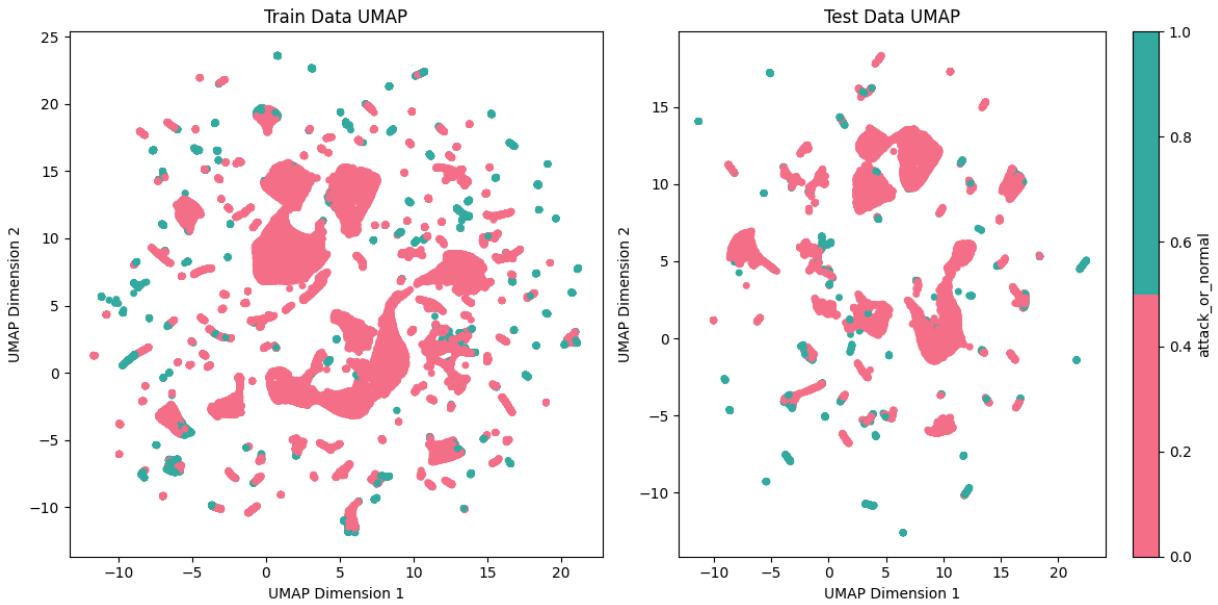
        # Test dataset
        X_test_binary_one_class_svm_labels = train_binary_one_class_svm.predict(nadp_X_test_binary_one_class_svm)
        X_test_binary_one_class_svm_labels = np.where(X_test_binary_one_class_svm_labels < 0, -1, 1)
        nadp_X_test_binary_anomaly["binary_one_class_svm_df"] = train_binary_one_class_svm
```

```

# Log umap
create_binary_umap(binary_train_umap,X_train_binary_one_class_svm_labels,bin

# Log the model
mlflow.sklearn.log_model(train_binary_one_class_svm, f"{run_name}_train_mode
print("MLFLOW Logging is completed")
except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

```



2025/05/16 16:44:23 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.  
MLFLOW Logging is completed

Time taken to execute

binary\_one\_class\_svm: 10 min

```

In [ ]: params = {"nu":0.1, "verbose":0}
train_binary_one_class_svm = OneClassSVM(**params)
train_binary_one_class_svm = train_binary_one_class_svm.fit(nadp_X_train_binary_fina

In [ ]: with open("train_binary_one_class_svm.pkl","wb") as f:
    pickle.dump(train_binary_one_class_svm,f)

```

## DBSCAN

```

In [ ]: # Logging the binary_dbSCAN_umap.png artifact into mlflow
experiment_name = "nadp_binary"
run_name = "binary_dbSCAN"
#mlflow.create_experiment("nadp_binary")
mlflow.set_experiment(experiment_name)

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

        # Log params
        params = {"eps":0.5, "min_samples":5, "n_jobs":-1}
        mlflow.log_params(params)

        # Train dataset
        train_binary_dbSCAN = DBSCAN(**params)
        X_train_binary_dbSCAN_labels = train_binary_dbSCAN.fit_predict(nadp_X_train_
        X_train_binary_dbSCAN_labels = np.where(X_train_binary_dbSCAN_labels == -1,1
        nadp_X_train_binary_anomaly["binary_dbSCAN_labels"] = train_binary_dbSCAN.lab
        train_metrics = {"n_unique_labels":len(np.unique(train_binary_dbSCAN.labels_)}
        mlflow.log_metrics(train_metrics)

        # Test dataset
        test_binary_dbSCAN = DBSCAN(**params)
        X_test_binary_dbSCAN_labels = test_binary_dbSCAN.fit_predict(nadp_X_test_bin

```

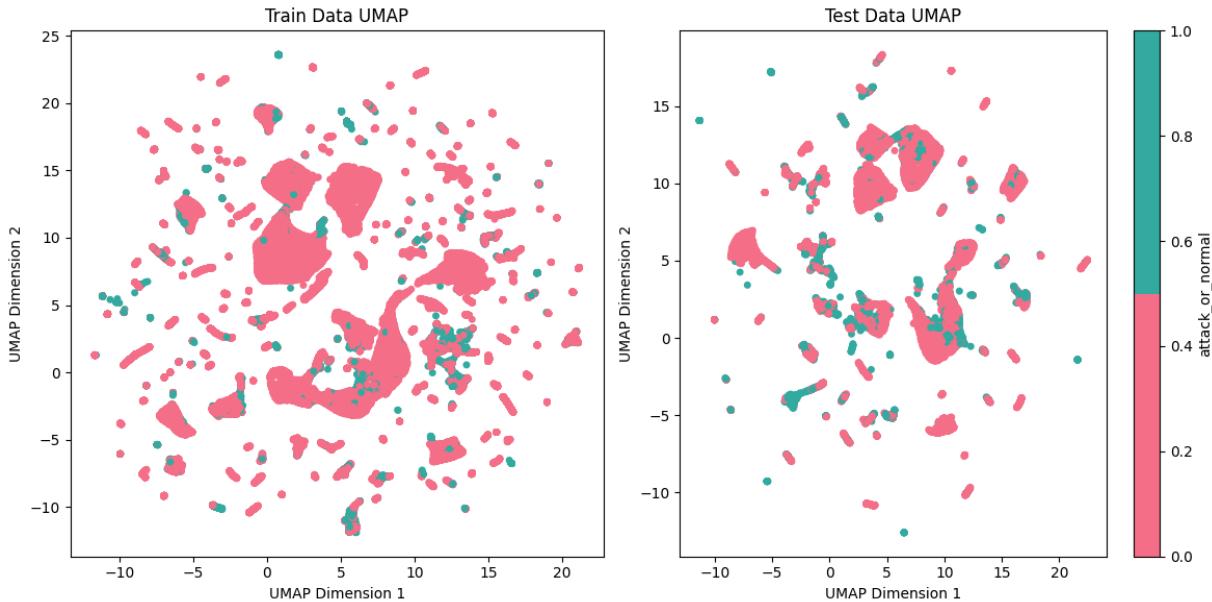
```

X_test_binary_dbSCAN_labels = np.where(X_test_binary_dbSCAN_labels == -1, 1, 0)
nadp_X_test_binary_anomaly["binary_dbSCAN_labels"] = test_binary_dbSCAN.labels_
test_metrics = {"n_unique_labels": len(np.unique(test_binary_dbSCAN.labels_))}
mlflow.log_metrics(test_metrics)

# Log umap
create_binary_umap(binary_train_umap, X_train_binary_dbSCAN_labels, binary_test_binary_dbSCAN_labels)

# Log the model
mlflow.sklearn.log_model(train_binary_dbSCAN, f"{run_name}_train_model")
mlflow.sklearn.log_model(test_binary_dbSCAN, f"{run_name}_test_model")
print("MLFLOW Logging is completed")
except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

```



```

2025/05/16 16:52:30 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/16 16:52:41 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto infer
the model signature.
2025/05/16 16:52:41 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/16 16:52:53 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto infer
the model signature.

```

MLFLOW Logging is completed

Time taken to execute

binary\_dbSCAN: 1 min

## kth Nearest Neighbor distance

```

In [ ]: # Logging the binary_knn_umap.png artifact into mlflow
experiment_name = "nadp_binary"
run_name = "binary_knn"
#mlflow.create_experiment("nadp_binary")
mlflow.set_experiment(experiment_name)

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

        # Log params
        params = {"n_neighbors": 5, "n_jobs": -1}
        mlflow.log_params(params)

        # Train dataset
        train_binary_knn = NearestNeighbors(**params)
        train_binary_knn.fit(nadp_X_train_binary_final)
        train_distances, train_indices = train_binary_knn.kneighbors(nadp_X_train_binary_final)
        nadp_X_train_binary_anomaly["binary_knn_kth_distance"] = train_distances[:, -1]
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")

```

```

# Test dataset
# test_binary_knn = NearestNeighbors(**params)
# X_train_binary_knn_labels = test_binary_knn.fit(nadp_X_train_binary_final)
test_distances, test_indices = train_binary_knn.kneighbors(nadp_X_test_binary)
nadp_X_test_binary_anomaly["binary_knn_kth_distance"] = test_distances[:, -1]

# Log umap (No umap in knn because we cannot calculate labels in unsupervised)
# create_binary_umap(binary_train_umap,X_train_binary_knn_labels,binary_test)

# Log the model
mlflow.sklearn.log_model(train_binary_knn, f"{run_name}_train_model")
print("MLFLOW Logging is completed")
except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

```

```

2025/05/16 16:54:04 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/16 16:54:15 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
MLFLOW Logging is completed

```

Time taken to execute

binary\_knn: 3 min

```
In [ ]: params = {"n_neighbors":5,"n_jobs":-1}
train_binary_knn = NearestNeighbors(**params)
train_binary_knn = train_binary_knn.fit(nadp_X_train_binary_final)
```

```
In [ ]: with open("train_binary_knn.pkl","wb") as f:
pickle.dump(train_binary_knn,f)
```

## GMM Score

```

In [ ]: # Logging the binary_gmm_umap.png artifact into mlflow
experiment_name = "nadp_binary"
run_name = "binary_gmm"
#mlflow.create_experiment("nadp_binary")
mlflow.set_experiment(experiment_name)

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

        # Log params
        params = {"n_components":2, "random_state":42,"verbose":0}
        mlflow.log_params(params)

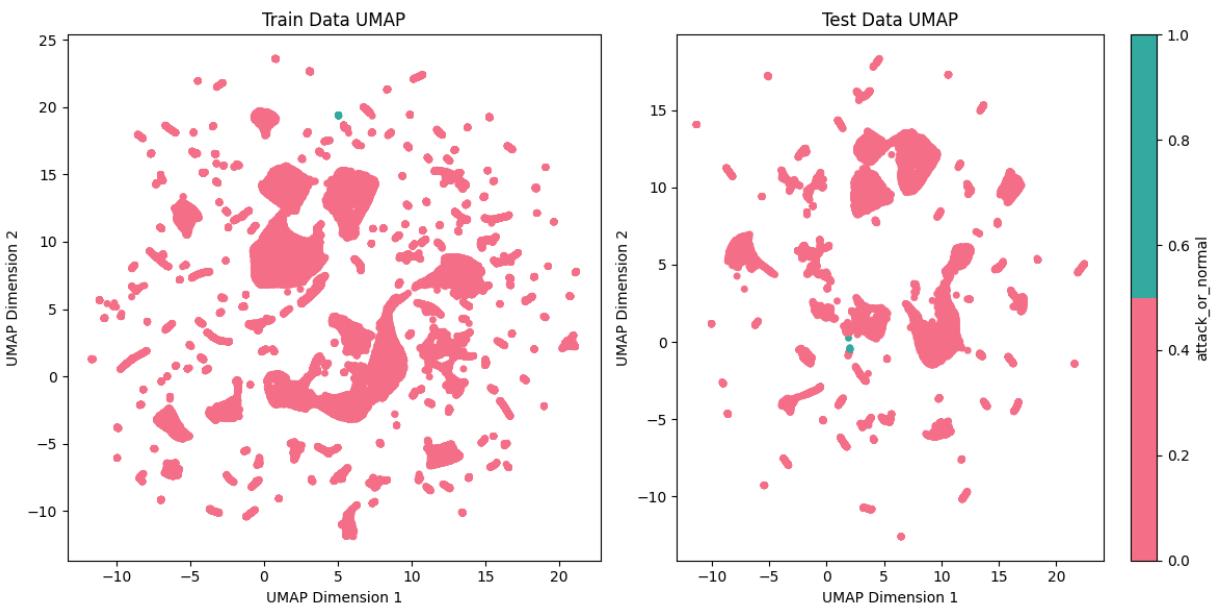
        # Train dataset
        train_binary_gmm = GaussianMixture(**params)
        X_train_binary_gmm_labels = train_binary_gmm.fit_predict(nadp_X_train_binary)
        # X_train_binary_gmm_labels = np.where(X_train_binary_gmm_labels == -1,1,0)
        nadp_X_train_binary_anomaly["binary_gmm_score"] = train_binary_gmm.score_samples
        train_metrics = {"AIC":train_binary_gmm.aic(nadp_X_train_binary_final),"BIC":train_binary_gmm.bic(nadp_X_train_binary_final)}
        mlflow.log_metrics(train_metrics)

        # Test dataset
        X_test_binary_gmm_labels = train_binary_gmm.predict(nadp_X_test_binary_final)
        # X_test_binary_gmm_labels = np.where(X_test_binary_gmm_labels == -1,1,0)
        nadp_X_test_binary_anomaly["binary_gmm_score"] = train_binary_gmm.score_samples

        # Log umap
        create_binary_umap(binary_train_umap,X_train_binary_gmm_labels,binary_test_u

        # Log the model
        mlflow.sklearn.log_model(train_binary_gmm, f"{run_name}_train_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")

```



```
2025/05/16 16:54:46 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

Time taken to execute

binary\_gmm: 28 s

```
In [ ]: params = {"n_components":2, "random_state":42, "verbose":0}
train_binary_gmm = GaussianMixture(**params)
train_binary_gmm = train_binary_gmm.fit(nadp_X_train_binary_final)
```

```
In [ ]: with open("train_binary_gmm.pkl", "wb") as f:
    pickle.dump(train_binary_gmm, f)
```

### Storing the preprocessed df

```
In [ ]: nadp_X_train_binary_anomaly.to_csv("nadp_X_train_binary_anomaly", index = False)
nadp_X_test_binary_anomaly.to_csv("nadp_X_test_binary_anomaly", index = False)
```

```
In [ ]: # Loading the anomaly preprocessed dataframes
nadp_X_train_binary_anomaly = pd.read_csv("./nadp_X_train_binary_anomaly")
nadp_X_test_binary_anomaly = pd.read_csv("./nadp_X_test_binary_anomaly")
```

```
In [ ]: nadp_X_test_binary_anomaly.shape
```

```
Out[ ]: (25193, 61)
```

```
In [ ]: nadp_X_train_binary_anomaly.shape
```

```
Out[ ]: (100767, 61)
```

### Creating an additional feature using kmeans and a new assumption

#### Assumptions and Additional Hyper parameter definitions

##### ASSUMPTION

Normal data belongs to large, dense clusters, while anomalies belong to small, sparse clusters.

##### ADDITIONAL HYPER PARAMETERS USED

**Alpha ( $\alpha$ ):** This parameter determines the threshold for defining a cluster as *small*. It controls the minimum size below which a cluster is considered anomalous due to its small size. Specifically, if the size of a cluster is less than  $\alpha$

$\beta * (N / k)$  (where  $N$  is the total number of data points and  $k$  is the number of clusters), that cluster is marked as anomalous.

**Beta ( $\beta$ ):** This parameter sets the threshold for *sparsity* of a cluster. It is used to determine if a cluster is sparse compared to the median within-cluster sum of squares (i.e., the average distance of points within the cluster to their centroid). If the within-cluster average distance  $\epsilon_i$  is greater than  $\beta * \text{median}(E)$  (where  $E$  is the set of within-cluster averages for all clusters), the cluster is marked as anomalous for being sparse.

**Gamma ( $\gamma$ ):** This parameter defines the threshold for *extreme density*. It identifies clusters that are much denser than usual. If the within-cluster average distance  $\epsilon_i$  is less than  $\gamma * \text{median}(E)$ , the cluster is considered anomalous for being extremely dense.

## Scaling the data with anomaly scores again

```
In [ ]: # Standardize the data
k_means_scaler = StandardScaler()
data = k_means_scaler.fit_transform(nadp_X_train_binary_anomaly)

# Save the scaler for future use
with open('k_means_scaler.pkl', 'wb') as f:
    pickle.dump(k_means_scaler, f)
```

## Hyper parameter tuning of alpha, beta, gamma

```
In [ ]: def label_clusters(labels, centroids, points, alpha, beta, gamma):
    """
    Labels clusters as normal or anomaly based on size, density, and extreme density
    """
    unique_labels = np.unique(labels[labels != -1])
    n_clusters = len(unique_labels)
    cluster_sizes = np.array([np.sum(labels == i) for i in unique_labels])
    N = len(points)
    anomaly_labels = np.full(labels.shape, 'normal')

    # Calculate within-cluster sum of squares
    within_cluster_sums = []
    for i in unique_labels:
        cluster_points = points[labels == i]
        centroid = centroids[i]
        sum_of_squares = np.sum(np.linalg.norm(cluster_points - centroid, axis=1)**2)
        within_cluster_sums.append(sum_of_squares / len(cluster_points) if len(cluster_points) > 0 else 0)

    median_within_sum = np.median(within_cluster_sums)

    # Label clusters based on the given conditions
    for i, label in enumerate(unique_labels):
        size = cluster_sizes[i]
        average_within_sum = within_cluster_sums[i]

        if size < alpha * (N / n_clusters):
            anomaly_labels[labels == label] = 'anomalous'
        elif average_within_sum > beta * median_within_sum:
            anomaly_labels[labels == label] = 'anomalous'
        elif average_within_sum < gamma * median_within_sum:
            anomaly_labels[labels == label] = 'anomalous'

    return anomaly_labels

def clustering_methods(data, k_values, alpha, beta, gamma):
    results = {}

    for k in k_values:
        kmeans = KMeans(n_clusters=k, random_state=42).fit(data)
        labels = kmeans.labels_
        centroids = kmeans.cluster_centers_
        results[k] = (labels, centroids)
```

```
        labeled_data = label_clusters(labels, centroids, data, alpha, beta, gamma)
results[f'KMeans_k={k}'] = labeled_data

return results
```

```
In [ ]: # Hyperparameter grid
alpha_values = [0.01, 0.05, 0.1, 0.25, 0.5, 0.75]
beta_values = [0.5, 1.0, 1.5, 2.0]
gamma_values = [0.025, 0.05, 0.1, 0.15, 0.25]
k_values = [2, 4, 8, 16, 32, 64, 128, 256]

# True Labels for accuracy calculation
true_labels = np.where(nadp_y_train_binary_final == 1, 'anomalous', 'normal')

# Search for the best hyperparameters
best_accuracy = 0
best_params = None

for alpha, beta, gamma in product(alpha_values, beta_values, gamma_values):
    results = clustering_methods(data, k_values, alpha, beta, gamma)
    for method, predicted_labels in results.items():
        accuracy = accuracy_score(true_labels, predicted_labels)
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_params = (method, alpha, beta, gamma)
print(f"method:{method}, accuracy: {accuracy}, alpha:{alpha}, beta:{beta}, g
```

method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.01, beta:0.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.01, beta:0.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.6155189695039051, alpha:0.01, beta:0.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5420921531850705, alpha:0.01, beta:0.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5600841545347187, alpha:0.01, beta:0.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5282384113846795, alpha:0.01, beta:0.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.01, beta:0.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.01, beta:0.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.6155189695039051, alpha:0.01, beta:0.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5420921531850705, alpha:0.01, beta:0.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5600841545347187, alpha:0.01, beta:0.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4762571079817797, alpha:0.01, beta:0.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.01, beta:0.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.01, beta:0.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.6155189695039051, alpha:0.01, beta:0.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4178153562178094, alpha:0.01, beta:0.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5600841545347187, alpha:0.01, beta:0.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4436670735459029, alpha:0.01, beta:0.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.01, beta:0.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.01, beta:0.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5474907459783461, alpha:0.01, beta:0.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2796947413339685, alpha:0.01, beta:0.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.38247640596623894, alpha:0.01, beta:0.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.39531791161789076, alpha:0.01, beta:0.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.01, beta:0.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.01, beta:0.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.01, beta:0.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.6066172457252871, alpha:0.01, beta:0.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4663332241706114, alpha:0.01, beta:0.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.40764337531136186, alpha:0.01, beta:0.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.42525826907618564, alpha:0.01, beta:0.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.01, beta:1.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.01, beta:1.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.01, beta:1.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.01, beta:1.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5679934899322199, alpha:0.01, beta:1.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5207061835720027, alpha:0.01, beta:1.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5424692607698949, alpha:0.01, beta:1.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5223039288656008, alpha:0.01, beta:1.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.01, beta:1.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.01, beta:1.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.01, beta:1.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.01, beta:1.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5679934899322199, alpha:0.01, beta:1.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5207061835720027, alpha:0.01, beta:1.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5424692607698949, alpha:0.01, beta:1.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4703226254627011, alpha:0.01, beta:1.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.01, beta:1.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.01, beta:1.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.01, beta:1.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.01, beta:1.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5679934899322199, alpha:0.01, beta:1.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.39642938660474164, alpha:0.01, beta:1.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5424692607698949, alpha:0.01, beta:1.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.43773259102682427, alpha:0.01, beta:1.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.01, beta:1.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.01, beta:1.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.01, beta:1.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.01, beta:1.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.4999652664066609, alpha:0.01, beta:1.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.25830877172090067, alpha:0.01, beta:1.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.36486151220141516, alpha:0.01, beta:1.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.38938342909881213, alpha:0.01, beta:1.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.01, beta:1.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.01, beta:1.0, gamma:0.25

method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.01, beta:1.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.01, beta:1.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5590917661536019, alpha:0.01, beta:1.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.44494725455754364, alpha:0.01, beta:1.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3900284815465381, alpha:0.01, beta:1.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.419323786557107, alpha:0.01, beta:1.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:1.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:1.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:1.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.01, beta:1.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.550725932100787, alpha:0.01, beta:1.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.530332350868836, alpha:0.01, beta:1.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5073188643107367, alpha:0.01, beta:1.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5087280558119226, alpha:0.01, beta:1.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:1.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:1.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:1.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.01, beta:1.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.550725932100787, alpha:0.01, beta:1.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.530332350868836, alpha:0.01, beta:1.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5073188643107367, alpha:0.01, beta:1.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4567467524090228, alpha:0.01, beta:1.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:1.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:1.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:1.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.01, beta:1.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.550725932100787, alpha:0.01, beta:1.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.40605555390157494, alpha:0.01, beta:1.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5073188643107367, alpha:0.01, beta:1.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.42415671797314597, alpha:0.01, beta:1.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:1.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:1.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:1.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.01, beta:1.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.482697708575228, alpha:0.01, beta:1.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.26793493901773396, alpha:0.01, beta:1.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.3297111574225687, alpha:0.01, beta:1.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.37580755604513383, alpha:0.01, beta:1.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:1.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:1.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:1.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.01, beta:1.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5418242083221689, alpha:0.01, beta:1.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.45457342185437694, alpha:0.01, beta:1.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3548780850873798, alpha:0.01, beta:1.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4057479135034287, alpha:0.01, beta:1.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:2.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:2.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:2.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.01, beta:2.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.550725932100787, alpha:0.01, beta:2.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5340637311818354, alpha:0.01, beta:2.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5167763255827801, alpha:0.01, beta:2.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5109510057856242, alpha:0.01, beta:2.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:2.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:2.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:2.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.01, beta:2.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.550725932100787, alpha:0.01, beta:2.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5340637311818354, alpha:0.01, beta:2.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5167763255827801, alpha:0.01, beta:2.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4589697023827245, alpha:0.01, beta:2.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:2.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:2.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:2.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.01, beta:2.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.550725932100787, alpha:0.01, beta:2.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.40978693421457424, alpha:0.01, beta:2.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5167763255827801, alpha:0.01, beta:2.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4263796679468477, alpha:0.01, beta:2.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:2.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:2.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:2.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.01, beta:2.0, gamma:0.15

method:KMeans\_k=32, accuracy: 0.482697708575228, alpha:0.01, beta:2.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.27166631933073326, alpha:0.01, beta:2.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.3391685770143003, alpha:0.01, beta:2.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.37803050601883553, alpha:0.01, beta:2.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.01, beta:2.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.01, beta:2.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.01, beta:2.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.01, beta:2.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5418242083221689, alpha:0.01, beta:2.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.45830480216737624, alpha:0.01, beta:2.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3643355463594232, alpha:0.01, beta:2.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4079708634771304, alpha:0.01, beta:2.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.05, beta:0.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.05, beta:0.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.6155189695039051, alpha:0.05, beta:0.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5420921531850705, alpha:0.05, beta:0.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.05, beta:0.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5282781069199242, alpha:0.05, beta:0.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.05, beta:0.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.05, beta:0.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.6155189695039051, alpha:0.05, beta:0.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5420921531850705, alpha:0.05, beta:0.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.05, beta:0.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4762968035170244, alpha:0.05, beta:0.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.05, beta:0.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.05, beta:0.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.6155189695039051, alpha:0.05, beta:0.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4178153562178094, alpha:0.05, beta:0.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.05, beta:0.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4437067690811476, alpha:0.05, beta:0.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.05, beta:0.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.05, beta:0.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5474907459783461, alpha:0.05, beta:0.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2796947413339685, alpha:0.05, beta:0.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.38282374189962987, alpha:0.05, beta:0.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.39535760715313545, alpha:0.05, beta:0.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.05, beta:0.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.05, beta:0.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.05, beta:0.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.6066172457252871, alpha:0.05, beta:0.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4663332241706114, alpha:0.05, beta:0.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.40764337531136186, alpha:0.05, beta:0.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.42525826907618564, alpha:0.05, beta:0.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.05, beta:1.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.05, beta:1.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.05, beta:1.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.05, beta:1.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5679934899322199, alpha:0.05, beta:1.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5207061835720027, alpha:0.05, beta:1.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.05, beta:1.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5223436244008455, alpha:0.05, beta:1.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.05, beta:1.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.05, beta:1.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.05, beta:1.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.05, beta:1.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5679934899322199, alpha:0.05, beta:1.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5207061835720027, alpha:0.05, beta:1.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.05, beta:1.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4703623209979458, alpha:0.05, beta:1.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.05, beta:1.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.05, beta:1.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.05, beta:1.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.05, beta:1.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5679934899322199, alpha:0.05, beta:1.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.39642938660474164, alpha:0.05, beta:1.0, gamma:0.1

method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.05, beta:1.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.43777228656206896, alpha:0.05, beta:1.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.05, beta:1.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.05, beta:1.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.05, beta:1.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.05, beta:1.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.4999652664066609, alpha:0.05, beta:1.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.25830877172090067, alpha:0.05, beta:1.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.36520884813480603, alpha:0.05, beta:1.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.38942312463405676, alpha:0.05, beta:1.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.05, beta:1.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.11522621493147558, alpha:0.05, beta:1.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.05, beta:1.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.05, beta:1.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5590917661536019, alpha:0.05, beta:1.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.44494725455754364, alpha:0.05, beta:1.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3900284815465381, alpha:0.05, beta:1.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.419323786557107, alpha:0.05, beta:1.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:1.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:1.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:1.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.05, beta:1.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5497930870225372, alpha:0.05, beta:1.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.530332350868836, alpha:0.05, beta:1.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5076662002441276, alpha:0.05, beta:1.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5087677513471672, alpha:0.05, beta:1.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:1.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:1.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:1.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.05, beta:1.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5497930870225372, alpha:0.05, beta:1.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.530332350868836, alpha:0.05, beta:1.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5076662002441276, alpha:0.05, beta:1.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4567864479442675, alpha:0.05, beta:1.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:1.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:1.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:1.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.05, beta:1.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5497930870225372, alpha:0.05, beta:1.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.40605555390157494, alpha:0.05, beta:1.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5076662002441276, alpha:0.05, beta:1.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.42419641350839066, alpha:0.05, beta:1.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:1.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:1.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:1.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.05, beta:1.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.48176486349697817, alpha:0.05, beta:1.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.26793493901773396, alpha:0.05, beta:1.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.3300584516756478, alpha:0.05, beta:1.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3758472515803785, alpha:0.05, beta:1.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:1.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:1.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:1.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.05, beta:1.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5408913632439192, alpha:0.05, beta:1.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.45457342185437694, alpha:0.05, beta:1.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3548780850873798, alpha:0.05, beta:1.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4057479135034287, alpha:0.05, beta:1.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:2.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:2.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:2.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.05, beta:2.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5497930870225372, alpha:0.05, beta:2.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5341530461361358, alpha:0.05, beta:2.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.517123661516171, alpha:0.05, beta:2.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.510990701320869, alpha:0.05, beta:2.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:2.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:2.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:2.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.05, beta:2.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5497930870225372, alpha:0.05, beta:2.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5341530461361358, alpha:0.05, beta:2.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.517123661516171, alpha:0.05, beta:2.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4590093979179692, alpha:0.05, beta:2.0, gamma:0.05

method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:2.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:2.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:2.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.05, beta:2.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5497930870225372, alpha:0.05, beta:2.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4098762491688747, alpha:0.05, beta:2.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.517123661516171, alpha:0.05, beta:2.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.42641936348209236, alpha:0.05, beta:2.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:2.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:2.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:2.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.05, beta:2.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.48176486349697817, alpha:0.05, beta:2.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2717556342850338, alpha:0.05, beta:2.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.33951591294769123, alpha:0.05, beta:2.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3780702015540802, alpha:0.05, beta:2.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.05, beta:2.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5499915646987605, alpha:0.05, beta:2.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.05, beta:2.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.05, beta:2.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5408913632439192, alpha:0.05, beta:2.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4583941171216767, alpha:0.05, beta:2.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3643355463594232, alpha:0.05, beta:2.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4079708634771304, alpha:0.05, beta:2.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.1, beta:0.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.1, beta:0.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.617523594033761, alpha:0.1, beta:0.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5420921531850705, alpha:0.1, beta:0.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.1, beta:0.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.528625442853315, alpha:0.1, beta:0.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.1, beta:0.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.1, beta:0.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.617523594033761, alpha:0.1, beta:0.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5420921531850705, alpha:0.1, beta:0.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.1, beta:0.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.47664413945041534, alpha:0.1, beta:0.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.1, beta:0.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.1, beta:0.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.617523594033761, alpha:0.1, beta:0.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4178153562178094, alpha:0.1, beta:0.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.1, beta:0.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4440541050145385, alpha:0.1, beta:0.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.1, beta:0.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.1, beta:0.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5494953705082021, alpha:0.1, beta:0.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2796947413339685, alpha:0.1, beta:0.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.38282374189962987, alpha:0.1, beta:0.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3957049430865263, alpha:0.1, beta:0.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.1, beta:0.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.1778161501285143, alpha:0.1, beta:0.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.17721079321603303, alpha:0.1, beta:0.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.608621870255143, alpha:0.1, beta:0.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4663332241706114, alpha:0.1, beta:0.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.40764337531136186, alpha:0.1, beta:0.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.42560560500957656, alpha:0.1, beta:0.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.1, beta:1.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.1, beta:1.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.1, beta:1.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.1, beta:1.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5699981144620759, alpha:0.1, beta:1.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5207061835720027, alpha:0.1, beta:1.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.1, beta:1.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5226909603342365, alpha:0.1, beta:1.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.1, beta:1.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.1, beta:1.0, gamma:0.05

method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.1, beta:1.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.1, beta:1.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5699981144620759, alpha:0.1, beta:1.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5207061835720027, alpha:0.1, beta:1.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.1, beta:1.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.47070965693133665, alpha:0.1, beta:1.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.1, beta:1.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.1, beta:1.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.1, beta:1.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.1, beta:1.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5699981144620759, alpha:0.1, beta:1.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.39642938660474164, alpha:0.1, beta:1.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.1, beta:1.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4381196224954598, alpha:0.1, beta:1.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.1, beta:1.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.1, beta:1.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.1, beta:1.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.1, beta:1.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5019698909365169, alpha:0.1, beta:1.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.25830877172090067, alpha:0.1, beta:1.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.36520884813480603, alpha:0.1, beta:1.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3897704605674477, alpha:0.1, beta:1.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.1, beta:1.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.1, beta:1.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.2419442873162841, alpha:0.1, beta:1.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.6051286631536118, alpha:0.1, beta:1.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5610963906834578, alpha:0.1, beta:1.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.44494725455754364, alpha:0.1, beta:1.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3900284815465381, alpha:0.1, beta:1.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.41967112249049787, alpha:0.1, beta:1.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:1.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:1.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:1.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.1, beta:1.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5517977115523931, alpha:0.1, beta:1.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5311361854575407, alpha:0.1, beta:1.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5084402631813987, alpha:0.1, beta:1.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5091150872805581, alpha:0.1, beta:1.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:1.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:1.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:1.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.1, beta:1.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5517977115523931, alpha:0.1, beta:1.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5311361854575407, alpha:0.1, beta:1.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5084402631813987, alpha:0.1, beta:1.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.45713378387765835, alpha:0.1, beta:1.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:1.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:1.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:1.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.1, beta:1.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5517977115523931, alpha:0.1, beta:1.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4068593884902796, alpha:0.1, beta:1.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5084402631813987, alpha:0.1, beta:1.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4245437494417815, alpha:0.1, beta:1.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:1.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:1.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:1.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.1, beta:1.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.48376948802683417, alpha:0.1, beta:1.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2687387736064386, alpha:0.1, beta:1.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.3308325146129189, alpha:0.1, beta:1.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3761945875137694, alpha:0.1, beta:1.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:1.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:1.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:1.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.5649766292536247, alpha:0.1, beta:1.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5428959877737751, alpha:0.1, beta:1.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4553772564430816, alpha:0.1, beta:1.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.35565214802465095, alpha:0.1, beta:1.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.40609524943681957, alpha:0.1, beta:1.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:2.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:2.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:2.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.1, beta:2.0, gamma:0.025

method:KMeans\_k=32, accuracy: 0.5517977115523931, alpha:0.1, beta:2.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5340240356465906, alpha:0.1, beta:2.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5178977244534421, alpha:0.1, beta:2.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5113380372542599, alpha:0.1, beta:2.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:2.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:2.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:2.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.1, beta:2.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5517977115523931, alpha:0.1, beta:2.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5340240356465906, alpha:0.1, beta:2.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5178977244534421, alpha:0.1, beta:2.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.45935673385136006, alpha:0.1, beta:2.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:2.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:2.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:2.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.1, beta:2.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5517977115523931, alpha:0.1, beta:2.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.40974723867932955, alpha:0.1, beta:2.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5178977244534421, alpha:0.1, beta:2.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.42676669941548323, alpha:0.1, beta:2.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:2.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:2.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:2.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.1, beta:2.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.48376948802683417, alpha:0.1, beta:2.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2716266237954886, alpha:0.1, beta:2.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.34028997588496235, alpha:0.1, beta:2.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3784175374874711, alpha:0.1, beta:2.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.1, beta:2.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.1, beta:2.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.6216320819315847, alpha:0.1, beta:2.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.5691545843381266, alpha:0.1, beta:2.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5428959877737751, alpha:0.1, beta:2.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.45826510663213155, alpha:0.1, beta:2.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.36510960929669434, alpha:0.1, beta:2.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4083181994105213, alpha:0.1, beta:2.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.25, beta:0.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.25, beta:0.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.25, beta:0.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5420921531850705, alpha:0.25, beta:0.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.25, beta:0.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.528625442853315, alpha:0.25, beta:0.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.25, beta:0.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.25, beta:0.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.25, beta:0.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5420921531850705, alpha:0.25, beta:0.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.25, beta:0.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.47664413945041534, alpha:0.25, beta:0.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.25, beta:0.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.25, beta:0.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.25, beta:0.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4178153562178094, alpha:0.25, beta:0.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5604314904681096, alpha:0.25, beta:0.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4440541050145385, alpha:0.25, beta:0.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.25, beta:0.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.25, beta:0.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5564817847112646, alpha:0.25, beta:0.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2796947413339685, alpha:0.25, beta:0.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.38282374189962987, alpha:0.25, beta:0.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3957049430865263, alpha:0.25, beta:0.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.25, beta:0.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.25, beta:0.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.25, beta:0.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.6156082844582056, alpha:0.25, beta:0.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4663332241706114, alpha:0.25, beta:0.5, gamma:0.25

method:KMeans\_k=128, accuracy: 0.40764337531136186, alpha:0.25, beta:0.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.42560560500957656, alpha:0.25, beta:0.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.25, beta:1.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.25, beta:1.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.25, beta:1.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.25, beta:1.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5769845286651384, alpha:0.25, beta:1.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.519892425099487, alpha:0.25, beta:1.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.25, beta:1.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5244871833040579, alpha:0.25, beta:1.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.25, beta:1.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.25, beta:1.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.25, beta:1.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.25, beta:1.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5769845286651384, alpha:0.25, beta:1.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.519892425099487, alpha:0.25, beta:1.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.25, beta:1.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4725058799011581, alpha:0.25, beta:1.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.25, beta:1.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.25, beta:1.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.25, beta:1.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.25, beta:1.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5769845286651384, alpha:0.25, beta:1.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.39561562813222584, alpha:0.25, beta:1.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5428165967032857, alpha:0.25, beta:1.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4399158454652813, alpha:0.25, beta:1.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.25, beta:1.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.25, beta:1.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.25, beta:1.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.25, beta:1.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5089563051395795, alpha:0.25, beta:1.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.25749501324838486, alpha:0.25, beta:1.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.36520884813480603, alpha:0.25, beta:1.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.39156668353726914, alpha:0.25, beta:1.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.25, beta:1.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.25, beta:1.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.25, beta:1.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.25, beta:1.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5680828048865204, alpha:0.25, beta:1.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.44413349608502783, alpha:0.25, beta:1.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3900284815465381, alpha:0.25, beta:1.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.42146734546031933, alpha:0.25, beta:1.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:1.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:1.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:1.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:1.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5587841257554557, alpha:0.25, beta:1.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5279803904055891, alpha:0.25, beta:1.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5097204441930394, alpha:0.25, beta:1.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5117250687228954, alpha:0.25, beta:1.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:1.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:1.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:1.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:1.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5587841257554557, alpha:0.25, beta:1.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5279803904055891, alpha:0.25, beta:1.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5097204441930394, alpha:0.25, beta:1.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4597437653199956, alpha:0.25, beta:1.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:1.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:1.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:1.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:1.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5587841257554557, alpha:0.25, beta:1.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.403703593438328, alpha:0.25, beta:1.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5097204441930394, alpha:0.25, beta:1.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4271537308841188, alpha:0.25, beta:1.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:1.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:1.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:1.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:1.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.4907559022298967, alpha:0.25, beta:1.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2655829785544871, alpha:0.25, beta:1.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.33211269562455964, alpha:0.25, beta:1.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.37880456895610665, alpha:0.25, beta:1.5, gamma:0.15

method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:1.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:1.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:1.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:1.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5498824019768377, alpha:0.25, beta:1.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.45222146139113, alpha:0.25, beta:1.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.35693232903629163, alpha:0.25, beta:1.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4087052308791569, alpha:0.25, beta:1.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:2.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:2.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:2.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:2.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5587841257554557, alpha:0.25, beta:2.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5308682405946391, alpha:0.25, beta:2.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5182152887353995, alpha:0.25, beta:2.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5140968769537646, alpha:0.25, beta:2.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:2.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:2.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:2.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:2.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5587841257554557, alpha:0.25, beta:2.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5308682405946391, alpha:0.25, beta:2.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5182152887353995, alpha:0.25, beta:2.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.46211557355086486, alpha:0.25, beta:2.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:2.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:2.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:2.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:2.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5587841257554557, alpha:0.25, beta:2.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.40659144362737804, alpha:0.25, beta:2.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5182152887353995, alpha:0.25, beta:2.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.42952553911498803, alpha:0.25, beta:2.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:2.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:2.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:2.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:2.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.4907559022298967, alpha:0.25, beta:2.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.26847082874353706, alpha:0.25, beta:2.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.34060754016691974, alpha:0.25, beta:2.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3811763771869759, alpha:0.25, beta:2.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.25, beta:2.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.25, beta:2.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.25, beta:2.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.5906298689054948, alpha:0.25, beta:2.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5498824019768377, alpha:0.25, beta:2.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.45510931158018003, alpha:0.25, beta:2.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.36542717357865173, alpha:0.25, beta:2.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4110770391100261, alpha:0.25, beta:2.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.5, beta:0.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.5, beta:0.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.5, beta:0.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5551321365129457, alpha:0.5, beta:0.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5628032986989788, alpha:0.5, beta:0.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5301735687278574, alpha:0.5, beta:0.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.5, beta:0.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.5, beta:0.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.5, beta:0.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5551321365129457, alpha:0.5, beta:0.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5628032986989788, alpha:0.5, beta:0.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.47819226532495757, alpha:0.5, beta:0.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.5, beta:0.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.5, beta:0.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.5, beta:0.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4308553395456846, alpha:0.5, beta:0.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5628032986989788, alpha:0.5, beta:0.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.44560223088908074, alpha:0.5, beta:0.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.15

method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.5, beta:0.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.5, beta:0.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5564817847112646, alpha:0.5, beta:0.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.29273472466184364, alpha:0.5, beta:0.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.38519555013049905, alpha:0.5, beta:0.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3972530689610686, alpha:0.5, beta:0.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.5, beta:0.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.5, beta:0.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.5, beta:0.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.6156082844582056, alpha:0.5, beta:0.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4793732074984866, alpha:0.5, beta:0.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.4100151835422311, alpha:0.5, beta:0.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4271537308841188, alpha:0.5, beta:0.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.5, beta:1.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.5, beta:1.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.5, beta:1.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.5, beta:1.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5912153780503537, alpha:0.5, beta:1.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5361477467821807, alpha:0.5, beta:1.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5466372919705856, alpha:0.5, beta:1.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5364950827155716, alpha:0.5, beta:1.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.5, beta:1.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.5, beta:1.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.5, beta:1.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.5, beta:1.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5912153780503537, alpha:0.5, beta:1.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5361477467821807, alpha:0.5, beta:1.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5466372919705856, alpha:0.5, beta:1.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4845137793126718, alpha:0.5, beta:1.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.5, beta:1.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.5, beta:1.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.5, beta:1.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.5, beta:1.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5912153780503537, alpha:0.5, beta:1.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.41187094981491956, alpha:0.5, beta:1.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5466372919705856, alpha:0.5, beta:1.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.451923744876795, alpha:0.5, beta:1.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.5, beta:1.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.5, beta:1.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.5, beta:1.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.5, beta:1.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5231871545247948, alpha:0.5, beta:1.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.27375033493107864, alpha:0.5, beta:1.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.36902954340210586, alpha:0.5, beta:1.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.40357458294878285, alpha:0.5, beta:1.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.5, beta:1.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.5, beta:1.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.5, beta:1.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.5, beta:1.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5823136542717358, alpha:0.5, beta:1.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.46038881776772156, alpha:0.5, beta:1.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.39384917681383785, alpha:0.5, beta:1.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.43347524487183303, alpha:0.5, beta:1.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:1.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:1.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:1.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:1.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5689164111266586, alpha:0.5, beta:1.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.558982603431679, alpha:0.5, beta:1.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5144243651195332, alpha:0.5, beta:1.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5258666031538103, alpha:0.5, beta:1.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:1.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:1.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:1.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:1.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5689164111266586, alpha:0.5, beta:1.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.558982603431679, alpha:0.5, beta:1.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5144243651195332, alpha:0.5, beta:1.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.47388529975091054, alpha:0.5, beta:1.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:1.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:1.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:1.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:1.5, gamma:0.1

method:KMeans\_k=32, accuracy: 0.5689164111266586, alpha:0.5, beta:1.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4347058064644179, alpha:0.5, beta:1.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5144243651195332, alpha:0.5, beta:1.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4412952653150337, alpha:0.5, beta:1.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:1.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:1.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:1.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:1.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5008881876010995, alpha:0.5, beta:1.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.296585191580577, alpha:0.5, beta:1.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.33681661655105344, alpha:0.5, beta:1.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3929461033870216, alpha:0.5, beta:1.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:1.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:1.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:1.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:1.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5600146873480405, alpha:0.5, beta:1.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4832236744172199, alpha:0.5, beta:1.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3616362499627854, alpha:0.5, beta:1.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.42284676531007176, alpha:0.5, beta:1.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:2.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:2.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:2.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:2.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5689164111266586, alpha:0.5, beta:2.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.561870453620729, alpha:0.5, beta:2.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5155755356416287, alpha:0.5, beta:2.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5264620361824803, alpha:0.5, beta:2.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:2.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:2.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:2.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:2.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5689164111266586, alpha:0.5, beta:2.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.561870453620729, alpha:0.5, beta:2.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5155755356416287, alpha:0.5, beta:2.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.47448073277958064, alpha:0.5, beta:2.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:2.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:2.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:2.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:2.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5689164111266586, alpha:0.5, beta:2.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.4375936566534679, alpha:0.5, beta:2.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5155755356416287, alpha:0.5, beta:2.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4418906983437038, alpha:0.5, beta:2.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:2.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:2.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:2.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:2.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5008881876010995, alpha:0.5, beta:2.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.29947304176962697, alpha:0.5, beta:2.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.33796778707314895, alpha:0.5, beta:2.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.3935415364156916, alpha:0.5, beta:2.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.5, beta:2.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.5, beta:2.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.5, beta:2.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.5, beta:2.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5600146873480405, alpha:0.5, beta:2.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4861115246062699, alpha:0.5, beta:2.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.36278742048488094, alpha:0.5, beta:2.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.42344219833874186, alpha:0.5, beta:2.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.75, beta:0.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.75, beta:0.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.75, beta:0.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5704347653497673, alpha:0.75, beta:0.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5681026526541427, alpha:0.75, beta:0.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5306598390346046, alpha:0.75, beta:0.5, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.75, beta:0.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.75, beta:0.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.75, beta:0.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5704347653497673, alpha:0.75, beta:0.5, gamma:0.05

method:KMeans\_k=128, accuracy: 0.5681026526541427, alpha:0.75, beta:0.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4786785356317048, alpha:0.75, beta:0.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.75, beta:0.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.75, beta:0.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.6245100082368236, alpha:0.75, beta:0.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.44615796838250615, alpha:0.75, beta:0.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5681026526541427, alpha:0.75, beta:0.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.446088501195828, alpha:0.75, beta:0.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.75, beta:0.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.75, beta:0.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5564817847112646, alpha:0.75, beta:0.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.30803735349866523, alpha:0.75, beta:0.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.39049490408566295, alpha:0.75, beta:0.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.39773933926781585, alpha:0.75, beta:0.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.4654003790923616, alpha:0.75, beta:0.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.19215616223565254, alpha:0.75, beta:0.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.1914515664850596, alpha:0.75, beta:0.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.6156082844582056, alpha:0.75, beta:0.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.4946758363353082, alpha:0.75, beta:0.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.415314537497395, alpha:0.75, beta:0.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.42764000119086604, alpha:0.75, beta:0.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.75, beta:1.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.75, beta:1.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.75, beta:1.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.75, beta:1.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:1.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5514503756190022, alpha:0.75, beta:1.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5609078368910457, alpha:0.75, beta:1.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5469449323687319, alpha:0.75, beta:1.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.75, beta:1.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.75, beta:1.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.75, beta:1.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.75, beta:1.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:1.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5514503756190022, alpha:0.75, beta:1.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5609078368910457, alpha:0.75, beta:1.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.49496362896583207, alpha:0.75, beta:1.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.75, beta:1.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.75, beta:1.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.75, beta:1.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.75, beta:1.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:1.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.42717357865174116, alpha:0.75, beta:1.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5609078368910457, alpha:0.75, beta:1.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.46237359452995525, alpha:0.75, beta:1.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.75, beta:1.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.75, beta:1.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.75, beta:1.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.75, beta:1.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5030317465043119, alpha:0.75, beta:1.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.2890529637679002, alpha:0.75, beta:1.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.3833000883225659, alpha:0.75, beta:1.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.4140244326019431, alpha:0.75, beta:1.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.9009298679131065, alpha:0.75, beta:1.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.1317395575932597, alpha:0.75, beta:1.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.25628429942342235, alpha:0.75, beta:1.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.630781902805482, alpha:0.75, beta:1.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5621582462512529, alpha:0.75, beta:1.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.47569144660454316, alpha:0.75, beta:1.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.40811972173429795, alpha:0.75, beta:1.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.4439250945249933, alpha:0.75, beta:1.0, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:1.5, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:1.5, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:1.5, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:1.5, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:1.5, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5716057836394851, alpha:0.75, beta:1.5, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5435013446862564, alpha:0.75, beta:1.5, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5354629987992101, alpha:0.75, beta:1.5, gamma:0.025

method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:1.5, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:1.5, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:1.5, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:1.5, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:1.5, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5716057836394851, alpha:0.75, beta:1.5, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5435013446862564, alpha:0.75, beta:1.5, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4834816953963103, alpha:0.75, beta:1.5, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:1.5, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:1.5, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:1.5, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:1.5, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:1.5, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.44732898667222404, alpha:0.75, beta:1.5, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5435013446862564, alpha:0.75, beta:1.5, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4508916609604335, alpha:0.75, beta:1.5, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:1.5, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:1.5, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:1.5, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:1.5, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5030317465043119, alpha:0.75, beta:1.5, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.3092083717883831, alpha:0.75, beta:1.5, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.36589359611777666, alpha:0.75, beta:1.5, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.40254249903242134, alpha:0.75, beta:1.5, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:1.5, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:1.5, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:1.5, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:1.5, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5621582462512529, alpha:0.75, beta:1.5, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.49584685462502603, alpha:0.75, beta:1.5, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.39071322952950865, alpha:0.75, beta:1.5, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.43244316095547153, alpha:0.75, beta:1.5, gamma:0.25  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:2.0, gamma:0.025  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:2.0, gamma:0.025  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:2.0, gamma:0.025  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:2.0, gamma:0.025  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:2.0, gamma:0.025  
method:KMeans\_k=64, accuracy: 0.5744936338285351, alpha:0.75, beta:2.0, gamma:0.025  
method:KMeans\_k=128, accuracy: 0.5503389006321514, alpha:0.75, beta:2.0, gamma:0.025  
method:KMeans\_k=256, accuracy: 0.5352049778201197, alpha:0.75, beta:2.0, gamma:0.025  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:2.0, gamma:0.05  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:2.0, gamma:0.05  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:2.0, gamma:0.05  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:2.0, gamma:0.05  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:2.0, gamma:0.05  
method:KMeans\_k=64, accuracy: 0.5744936338285351, alpha:0.75, beta:2.0, gamma:0.05  
method:KMeans\_k=128, accuracy: 0.5503389006321514, alpha:0.75, beta:2.0, gamma:0.05  
method:KMeans\_k=256, accuracy: 0.4832236744172199, alpha:0.75, beta:2.0, gamma:0.05  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:2.0, gamma:0.1  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:2.0, gamma:0.1  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:2.0, gamma:0.1  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:2.0, gamma:0.1  
method:KMeans\_k=32, accuracy: 0.5710599700298709, alpha:0.75, beta:2.0, gamma:0.1  
method:KMeans\_k=64, accuracy: 0.450216836861274, alpha:0.75, beta:2.0, gamma:0.1  
method:KMeans\_k=128, accuracy: 0.5503389006321514, alpha:0.75, beta:2.0, gamma:0.1  
method:KMeans\_k=256, accuracy: 0.4506336399813431, alpha:0.75, beta:2.0, gamma:0.1  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:2.0, gamma:0.15  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:2.0, gamma:0.15  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:2.0, gamma:0.15  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:2.0, gamma:0.15  
method:KMeans\_k=32, accuracy: 0.5030317465043119, alpha:0.75, beta:2.0, gamma:0.15  
method:KMeans\_k=64, accuracy: 0.3120962219774331, alpha:0.75, beta:2.0, gamma:0.15  
method:KMeans\_k=128, accuracy: 0.37273115206367163, alpha:0.75, beta:2.0, gamma:0.15  
method:KMeans\_k=256, accuracy: 0.40228447805333095, alpha:0.75, beta:2.0, gamma:0.15  
method:KMeans\_k=2, accuracy: 0.5345996209076385, alpha:0.75, beta:2.0, gamma:0.25  
method:KMeans\_k=4, accuracy: 0.5665049073605446, alpha:0.75, beta:2.0, gamma:0.25  
method:KMeans\_k=8, accuracy: 0.635972094038723, alpha:0.75, beta:2.0, gamma:0.25  
method:KMeans\_k=16, accuracy: 0.569650778528685, alpha:0.75, beta:2.0, gamma:0.25  
method:KMeans\_k=32, accuracy: 0.5621582462512529, alpha:0.75, beta:2.0, gamma:0.25  
method:KMeans\_k=64, accuracy: 0.49873470481407606, alpha:0.75, beta:2.0, gamma:0.25  
method:KMeans\_k=128, accuracy: 0.3975507854754037, alpha:0.75, beta:2.0, gamma:0.25  
method:KMeans\_k=256, accuracy: 0.43218513997638114, alpha:0.75, beta:2.0, gamma:0.25

```
In [ ]: print(f"best_accuracy:{best_accuracy},best_params:{best_params}")

best_accuracy:0.9009298679131065,best_params:('KMeans_k=2', 0.01, 1.0, 0.025)
```

```
best_accuracy:0.9108239800728413,best_params:('KMeans_k=2', 0.01, 1.0, 0.25)
```

## Best kmeans model training and testing and creating additional feature binary\_kmeans\_adv

```
In [ ]: # Applying the best k_means_scaler and parameters on both train and test data
best_method, best_alpha, best_beta, best_gamma = best_params #(2,0.01,1.0,0.25)
best_method = int(best_method.split('=')[-1])
data_train = k_means_scaler.transform(nadp_X_train_binary_anomaly)
data_test = k_means_scaler.transform(nadp_X_test_binary_anomaly)

# Re-run the best model using KMeans with the best parameters
kmeans_best = KMeans(n_clusters=best_method, random_state=42)
kmeans_best.fit(data_train)
train_labels = label_clusters(kmeans_best.labels_, kmeans_best.cluster_centers_, data_train)
test_labels = label_clusters(kmeans_best.predict(data_test), kmeans_best.cluster_centers_)

# Add the labels as a new feature
nadp_X_train_binary_anomaly["binary_kmeans_adv"] = np.where(train_labels == "anomal", 1, 0)
nadp_X_test_binary_anomaly["binary_kmeans_adv"] = np.where(test_labels == "anomal", 1, 0)
```

```
In [ ]: with open("kmeans_best.pkl","wb") as f:
    pickle.dump(kmeans_best,f)
```

## mlflow logging of kmeans\_adv

```
In [ ]: # Log metrics and model using MLflow
experiment_name = "nadp_binary"
run_name = "binary_kmeans_adv"
mlflow.set_experiment(experiment_name)

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

        # Log params
        params = {"alpha": best_alpha, "beta": best_beta, "gamma": best_gamma, "k": best_method}
        mlflow.log_params(params)

        # Define true labels for accuracy calculation
        train_true_labels = np.where(nadp_y_train_binary_final == 1, 'anomal', 'normal')
        test_true_labels = np.where(nadp_y_test_binary_final == 1, 'anomal', 'normal')

        # Flatten labels for comparison
        train_labels_flat = np.where(train_labels == "anomal", 1, 0).flatten()
        test_labels_flat = np.where(test_labels == "anomal", 1, 0).flatten()
        train_true_labels_flat = train_true_labels.flatten()
        test_true_labels_flat = test_true_labels.flatten()

        # Check for consistent lengths
        if len(train_true_labels_flat) != len(train_labels_flat):
            print(f"Mismatch between train_true_labels and train_labels: {len(train_true_labels)} vs {len(train_labels)}")
            raise ValueError("Labels have inconsistent lengths.")

        if len(test_true_labels_flat) != len(test_labels_flat):
            print(f"Mismatch between test_true_labels and test_labels: {len(test_true_labels)} vs {len(test_labels)}")
            raise ValueError("Labels have inconsistent lengths.")

        # Calculate metrics
        train_accuracy = accuracy_score(train_true_labels_flat, train_labels_flat)
        mlflow.log_metric("train_accuracy", train_accuracy)

        test_accuracy = accuracy_score(test_true_labels_flat, test_labels_flat)
        mlflow.log_metric("test_accuracy", test_accuracy)
```

```

mlflow.log_metric("silhouette_score_train", silhouette_score(data_train, kme
mlflow.log_metric("davies_bouldin_index_train", davies_bouldin_score(data_tr

    # Supervised metrics comparing predictions with true labels
    mlflow.log_metric("fowlkes_mallows_index", fowlkes_mallows_score(train_true_
    mlflow.log_metric("adjusted_mutual_info", adjusted_mutual_info_score(train_t
    mlflow.log_metric("adjusted_rand_score", adjusted_rand_score(train_true_labe
    mlflow.log_metric("normalized_mutual_info", normalized_mutual_info_score(trai
    mlflow.log_metric("homogeneity", homogeneity_score(train_true_labels_flat, t
    mlflow.log_metric("completeness", completeness_score(train_true_labels_flat,
    mlflow.log_metric("v_measure", v_measure_score(train_true_labels_flat, train

    # Create Pairwise Confusion Matrix
    pairwise_cm = pair_confusion_matrix(train_true_labels_flat, train_labels_fla
    pairwise_cm_disp = ConfusionMatrixDisplay(pairwise_cm)
    pairwise_cm_path = f"{run_name}_pairwise_cm.png"
    pairwise_cm_disp.plot(cmap=plt.cm.Blues)
    plt.savefig(pairwise_cm_path)
    plt.show()
    plt.close()
    mlflow.log_artifact(pairwise_cm_path)

    # Create Contingency Matrix
    cont_matrix = contingency_matrix(train_true_labels_flat, train_labels_flat)
    cont_matrix_disp = ConfusionMatrixDisplay(cont_matrix)
    contingency_cm_path = f"{run_name}_contingency_cm.png"
    cont_matrix_disp.plot(cmap=plt.cm.Blues)
    plt.savefig(contingency_cm_path)
    plt.show()
    plt.close()
    mlflow.log_artifact(contingency_cm_path)

    # Compute Learning curve data
    train_sizes, train_scores, test_scores = learning_curve(kmeans_best, data_tr
    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)

    # Plot the Learning curve
    plt.figure()
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training Sc
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validatio
    plt.title(f"Learning Curve - {run_name}")
    plt.xlabel("Training Examples")
    plt.ylabel("Accuracy")
    plt.legend(loc="best")
    plt.grid()

    # Save the plot
    learning_curve_path = f"{run_name}_learning_curve.png"
    plt.savefig(learning_curve_path)
    plt.show()
    plt.close()

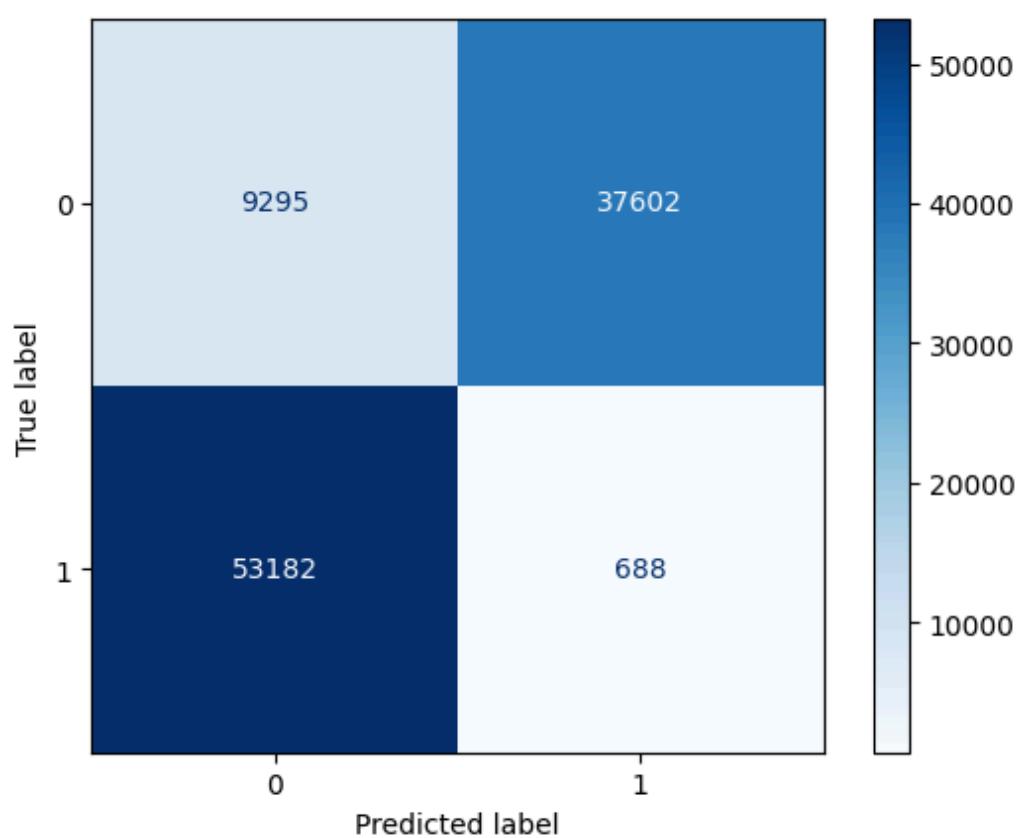
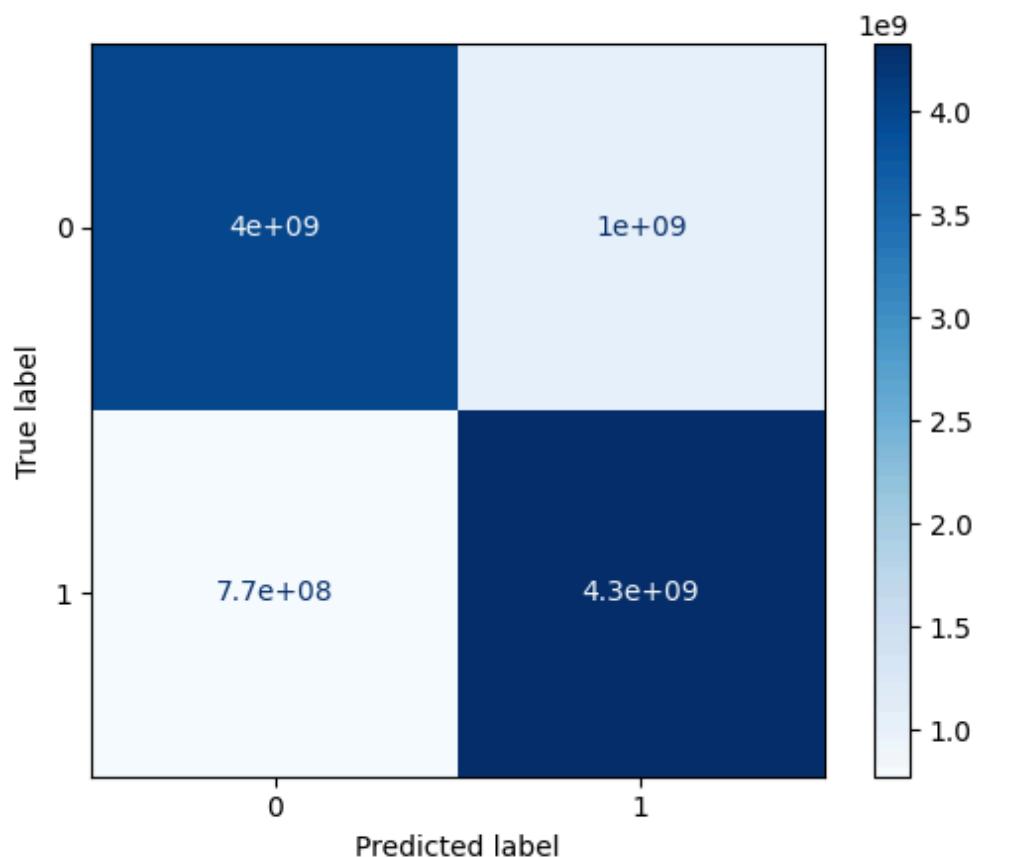
    # Log the plot as an artifact in MLflow
    mlflow.log_artifact(learning_curve_path)

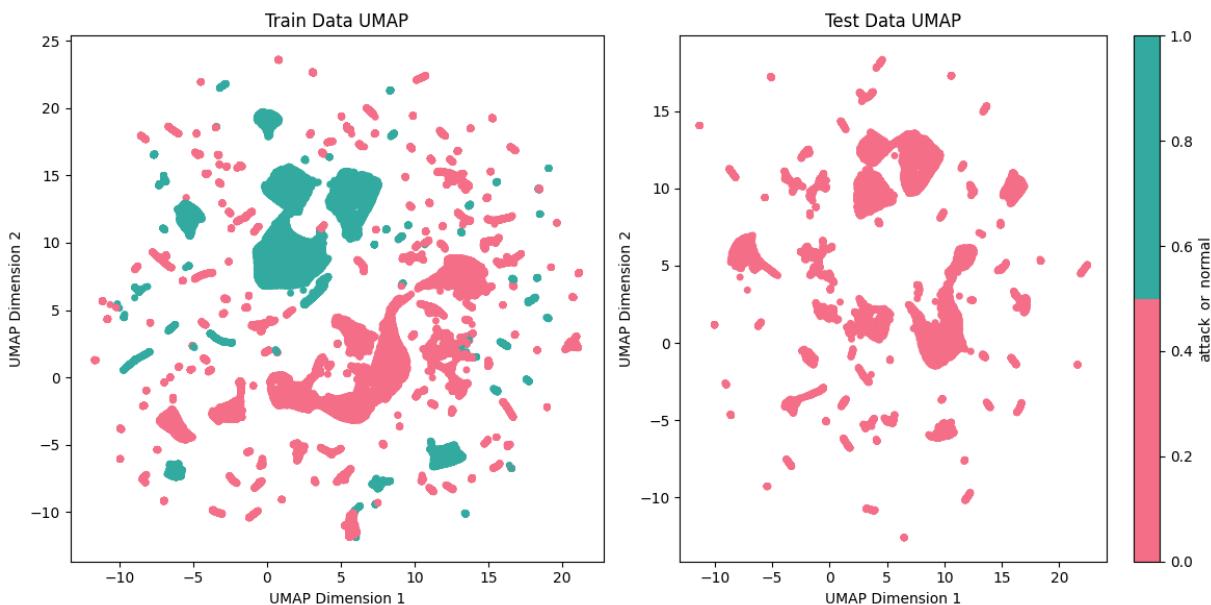
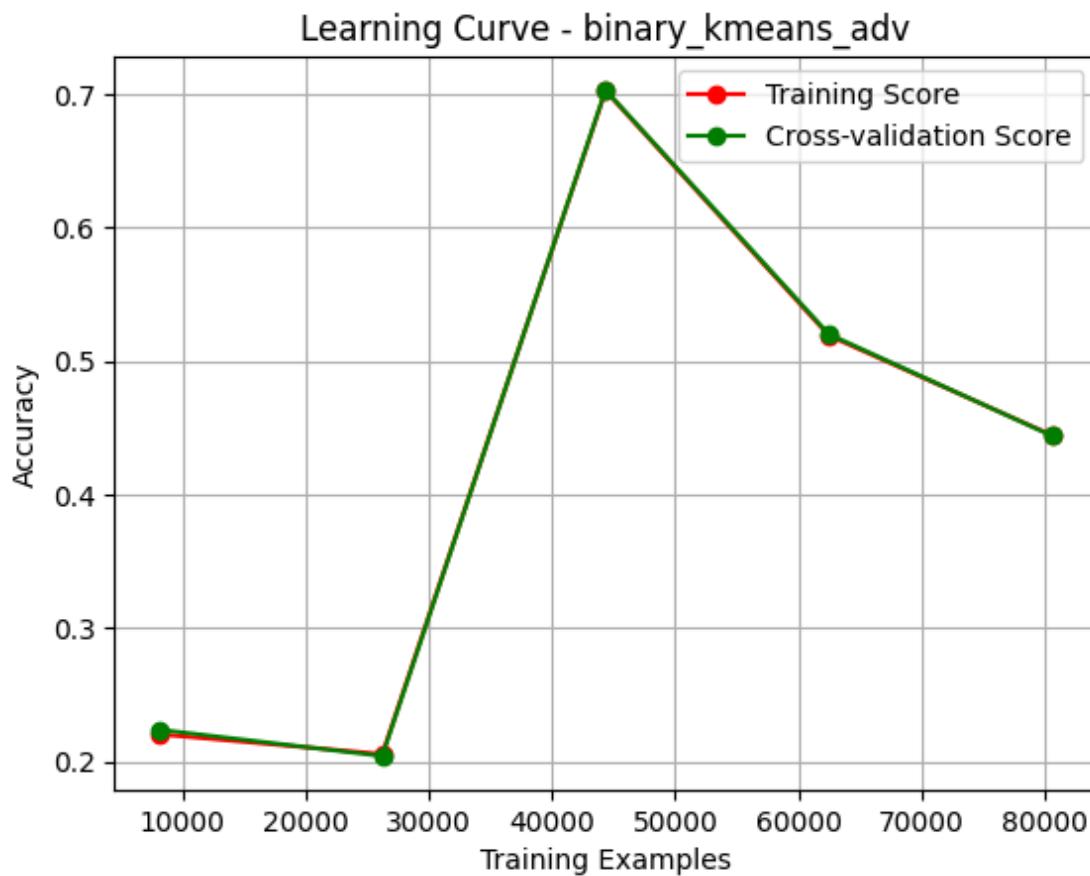
    # Log umap
    create_binary_umap(binary_train_umap, train_labels_flat, binary_test_umap, t

    # Log the trained model
    mlflow.sklearn.log_model(kmeans_best, f"{run_name}_train_model")
    print("MLFLOW Logging is completed")

except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

```





```
2025/05/16 17:56:51 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

In [ ]: `display_all(nadp_X_train_binary_anomaly.head())`

	duration	srcbytes	dstbytes	wrongfragment	urgent	hot	numfailedlogins	dur
0	-0.110353	-0.008212	-0.002061	-0.089155	-0.008191	-0.094254	-0.027121	
1	-0.110353	-0.008235	-0.005309	-0.089155	-0.008191	-0.094254	-0.027121	
2	-0.110353	-0.008201	-0.005003	-0.089155	-0.008191	-0.094254	-0.027121	
3	-0.110353	-0.008235	-0.005309	-0.089155	-0.008191	-0.094254	-0.027121	
4	-0.110353	-0.008235	-0.005309	-0.089155	-0.008191	-0.094254	-0.027121	

### Final scaling before classification algorithms

In [ ]: `kmeans_adv_scaler = StandardScaler()  
nadp_X_train_binary_anomaly_final = kmeans_adv_scaler.fit_transform(nadp_X_train_binary_anomaly)  
nadp_X_test_binary_anomaly_final = kmeans_adv_scaler.transform(nadp_X_test_binary_anomaly)`

# Save the scaler for future use

```

with open('kmeans_adv_scaler.pkl', 'wb') as f:
    pickle.dump(kmeans_adv_scaler, f)

In [ ]: nadp_X_train_binary_anomaly_final = pd.DataFrame(nadp_X_train_binary_anomaly_final,columns=nadp_X_train_binary_anomaly_final.columns)
nadp_X_test_binary_anomaly_final = pd.DataFrame(nadp_X_test_binary_anomaly_final,columns=nadp_X_test_binary_anomaly_final.columns)

In [ ]: nadp_X_train_binary_anomaly_final.to_csv("nadp_X_train_binary_anomaly_final",index=False)
nadp_X_test_binary_anomaly_final.to_csv("nadp_X_test_binary_anomaly_final",index=False)

In [ ]: X_train_imb = pd.read_csv("./nadp_X_train_binary_anomaly_final")
X_test_imb = pd.read_csv("./nadp_X_test_binary_anomaly_final")

In [ ]: y_train_imb = pd.read_csv("./nadp_y_train_binary_final").squeeze()
y_test_imb = pd.read_csv("./nadp_y_test_binary_final").squeeze()

```

## Creating Balanced datasets also to compare

```

In [ ]: # SMOTE balancing
smt = SMOTE(random_state=42)
X_train_bal, y_train_bal = smt.fit_resample(X_train_imb,y_train_imb)
X_test_bal = X_test_imb.copy(deep = True)
y_test_bal = y_test_imb.copy(deep = True)

print("X_train_bal shape",X_train_bal.shape)
print("X_test_bal shape",X_test_bal.shape)
print("y_train_bal shape",y_train_bal.shape)
print("y_test_bal shape",y_test_bal.shape)
print("y_train_bal value_counts", y_train_bal.value_counts())
print("y_test_bal value_counts", y_test_bal.value_counts())

X_train_bal shape (107740, 62)
X_test_bal shape (25193, 62)
y_train_bal shape (107740,)
y_test_bal shape (25193,)
y_train_bal value_counts attack_or_normal
0    53870
1    53870
Name: count, dtype: int64
y_test_bal value_counts attack_or_normal
0    13469
1    11724
Name: count, dtype: int64

In [ ]: # Save the scaler for future use
with open('binary_smote.pkl', 'wb') as f:
    pickle.dump(smt, f)

```

## Creating all the required functions to log metrics, params, artifacts, plots into MLflow and also print them.

### auc\_plots function

```

In [ ]: def auc_plots(model, X, y):
    try:
        y_pred_prob = model.predict_proba(X)[:, 1] # Consider only positive class
        fpr, tpr, _ = roc_curve(y, y_pred_prob)
        pr, re, _ = precision_recall_curve(y, y_pred_prob)
        roc_auc = auc(fpr, tpr)
        pr_auc = auc(re, pr)
        return fpr, tpr, pr, re, roc_auc, pr_auc
    except Exception as e:
        print(f"Error in auc_plots: {e}")
        return None, None, None, None, None, None

```

### plot\_learning\_curve function

```
In [ ]: def plot_learning_curve(model, X, y, run_name):
    try:
        train_sizes, train_scores, validation_scores = learning_curve(model, X, y, c
        train_mean = train_scores.mean(axis=1)
        train_std = train_scores.std(axis=1)
        validation_mean = validation_scores.mean(axis=1)
        validation_std = validation_scores.std(axis=1)

        plt.figure(figsize=(10, 6))
        plt.plot(train_sizes, train_mean, 'o-', color='blue', label='Training score')
        plt.plot(train_sizes, validation_mean, 'o-', color='red', label='Validation
        plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std
        plt.fill_between(train_sizes, validation_mean - validation_std, validation_m

        plt.xlabel('Training examples')
        plt.ylabel('Score')
        plt.title('Learning Curve')
        plt.legend(loc='best')
        plt.grid(True)

        # Save the Learning curve plot
        plt.savefig(f"{run_name}_learning_curve.png")
        plt.show()
        plt.close()
        return f"{run_name}_learning_curve.png"

    except Exception as e:
        print(f"Error in plot_learning_curve: {e}")
        return None
```

## mlflow\_logging\_and\_metric\_printing function

```
In [ ]: def mlflow_logging_and_metric_printing(model, run_name, bal_type, X_train, y_train,
                                              mlflow.set_experiment("nadp_binary"))

    with mlflow.start_run(run_name=run_name):
        try:
            # Log parameters
            if params:
                mlflow.log_params(params)
            mlflow.log_param("bal_type", bal_type)

            # Calculate metrics
            train_metrics = {
                "Accuracy_train": accuracy_score(y_train, y_pred_train),
                "Precision_train": precision_score(y_train, y_pred_train),
                "Recall_train": recall_score(y_train, y_pred_train),
                "F1_score_train": f1_score(y_train, y_pred_train),
                "F2_score_train": fbeta_score(y_train, y_pred_train, beta=2) # Emph
            }

            test_metrics = {
                "Accuracy_test": accuracy_score(y_test, y_pred_test),
                "Precision_test": precision_score(y_test, y_pred_test),
                "Recall_test": recall_score(y_test, y_pred_test),
                "F1_score_test": f1_score(y_test, y_pred_test),
                "F2_score_test": fbeta_score(y_test, y_pred_test, beta=2) # Emphasi
            }

            tuning_metrics = {"hyper_parameter_tuning_best_est_score":hyper_tuning_s

            # Compute AUC metrics
            train_fpr, train_tpr, train_pr, train_re, train_roc_auc, train_pr_auc =
            test_fpr, test_tpr, test_pr, test_re, test_roc_auc, test_pr_auc = auc_pl

            # Log AUC metrics
            if train_roc_auc is not None:
                train_metrics["Roc_auc_train"] = train_roc_auc
                mlflow.log_metric("Roc_auc_train", train_roc_auc)
            if train_pr_auc is not None:
```

```

        train_metrics["Pr_auc_train"] = train_pr_auc
        mlflow.log_metric("Pr_auc_train", train_pr_auc)
    if test_roc_auc is not None:
        test_metrics["Roc_auc_test"] = test_roc_auc
        mlflow.log_metric("Roc_auc_test", test_roc_auc)
    if test_pr_auc is not None:
        test_metrics["Pr_auc_test"] = test_pr_auc
        mlflow.log_metric("Pr_auc_test", test_pr_auc)

    # Print metrics
    print("Train Metrics:")
    for key, value in train_metrics.items():
        print(f"{key}: {value:.4f}")
    print("\nTest Metrics:")
    for key, value in test_metrics.items():
        print(f"{key}: {value:.4f}")
    print("\nTuning Metrics:")
    for key, value in tuning_metrics.items():
        print(f"{key}: {value:.4f}")

    # Classification Reports
    train_clf_report = classification_report(y_train, y_pred_train)
    test_clf_report = classification_report(y_test, y_pred_test)

    # Print classification reports
    print("\nTrain Classification Report:")
    print(train_clf_report)
    print("\nTest Classification Report:")
    print(test_clf_report)

    # Log metrics
    mlflow.log_metrics(train_metrics)
    mlflow.log_metrics(test_metrics)
    mlflow.log_metrics(tuning_metrics)

    # Convert classification reports to DataFrames
    train_clf_report_dict = classification_report(y_train, y_pred_train, output_dict=True)
    train_clf_report_df = pd.DataFrame(train_clf_report_dict).transpose()
    test_clf_report_dict = classification_report(y_test, y_pred_test, output_dict=True)
    test_clf_report_df = pd.DataFrame(test_clf_report_dict).transpose()

    # Save classification reports and log as artifacts
    train_clf_report_df.to_csv(f"{run_name}_train_classification_report.csv")
    mlflow.log_artifact(f"{run_name}_train_classification_report.csv")

    test_clf_report_df.to_csv(f"{run_name}_test_classification_report.csv")
    mlflow.log_artifact(f"{run_name}_test_classification_report.csv")

    # Plot confusion matrices
    fig, axes = plt.subplots(1, 2, figsize=(12, 6))

    ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_train, y_pred_train))
    axes[0].set_title('Train Confusion Matrix')
    # By mistake I have used "Not_Churned_off" & "Churned_off" --> "Change the labels"
    ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_test))
    axes[1].set_title('Test Confusion Matrix')

    plt.tight_layout()
    plt.savefig(f"{run_name}_confusion_matrix.png")
    mlflow.log_artifact(f"{run_name}_confusion_matrix.png")

    # ROC and Precision-Recall curves
    fig, axes = plt.subplots(2, 2, figsize=(12, 12))
    datasets = [("Train", X_train, y_train), ("Test", X_test, y_test)]

    for i, (name, X, y) in enumerate(datasets):
        fpr, tpr, pr, re, roc_auc, pr_auc = auc_plots(model, X, y)
        if fpr is None:
            return

        # ROC AUC Curve
        axes[i, 0].plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
        axes[i, 0].plot([0, 1], [0, 1], color='grey', lw=2, linestyle='--')

```

```

        axes[i, 0].set_xlim([0.0, 1.0])
        axes[i, 0].set_ylim([0.0, 1.0])
        axes[i, 0].set_xlabel('False Positive Rate')
        axes[i, 0].set_ylabel('True Positive Rate')
        axes[i, 0].set_title(f'Receiver Operating Characteristic ({name} data)')
        axes[i, 0].legend(loc='lower right')

        # Precision-Recall Curve
        axes[i, 1].plot(re, pr, color='green', lw=2, label=f'Precision-Recall Curve ({name} data)')
        axes[i, 1].set_xlabel('Recall')
        axes[i, 1].set_ylabel('Precision')
        axes[i, 1].set_title(f'Precision-Recall Curve ({name} data)')
        axes[i, 1].legend(loc='lower left')

    plt.tight_layout()
    plt.savefig(f'{run_name}_auc_plots.png')
    mlflow.log_artifact(f'{run_name}_auc_plots.png')

    # Plot Learning curves
    learning_curve_file = plot_learning_curve(model, X_train, y_train, run_name)
    if learning_curve_file:
        mlflow.log_artifact(learning_curve_file)

    # Log the model
    mlflow.sklearn.log_model(model, f'{run_name}_model')
    print("MLFLOW Logging is completed")

except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

# To view the Logged data, run the following command in the terminal:
# mlflow ui

```

## Initialise time and feature\_importance df

```

In [ ]: # Initialize DataFrames
time_df = pd.DataFrame(columns=["Model", "bal_type", "Training_Time", "Testing_Time"],
feature_importance_df = pd.DataFrame()

In [ ]: nadp_features = X_train_imb.columns

```

## Simple\_Logistic\_Regesssion\_on\_Imbalanced Dataset

```

In [ ]: # Model details
name = "Simple_Logistic_Regesssion_on_Imbalanced_Dataset"
model = LogisticRegression(random_state=42, n_jobs = -1)

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")

```

```
Model: Simple_Logistic_Regression_on_Imbalanced_Dataset
Training Time: 3.5734 seconds
Testing Time: 0.0425 seconds
```

## log\_time\_and\_feature\_importance\_df function

```
In [ ]: def log_time_and_feature_importances_df(time_df, feature_importance_df, name, traini
        # Store the times in the DataFrame using pd.concat
        time_df = pd.concat([time_df, pd.DataFrame({
            "Model": [name],
            "bal_type": [bal_type],
            "Training_Time": [training_time],
            "Testing_Time": [testing_time],
            "Tuning_Time": [tuning_time]
        })], ignore_index=True)

        # Feature Importances
        if hasattr(model, "coef_"):
            # For Linear models Like Logistic Regression or Linear SVC
            feature_importances = model.coef_.flatten()

        elif hasattr(model, "feature_importances_"):
            # For tree-based models Like Decision Trees, RandomForest, GradientBoosting,
            feature_importances = model.feature_importances_

        else:
            # If the model does not have feature importances, we skip Logging for this m
            feature_importances = None

        if feature_importances is not None:
            # Create a DataFrame with feature importances
            importance_df = pd.DataFrame(feature_importances, index=nadp_features, column
            feature_importance_df = pd.concat([feature_importance_df, importance_df], axis=1)

    return time_df, feature_importance_df
```

```
In [ ]: def feature_importances_df_new(feature_importance_df, name, model, nadp_features, ba
        # Feature Importances
        if hasattr(model, "coef_"):
            # For Linear models Like Logistic Regression or Linear SVC
            feature_importances = model.coef_.flatten()

        elif hasattr(model, "feature_importances_"):
            # For tree-based models Like Decision Trees, RandomForest, GradientBoosting,
            feature_importances = model.feature_importances_

        else:
            # If the model does not have feature importances, we skip Logging for this m
            feature_importances = None

        if feature_importances is not None:
            # Create a DataFrame with feature importances
            importance_df = pd.DataFrame(feature_importances, index=nadp_features, column
            feature_importance_df = pd.concat([feature_importance_df, importance_df], axis=1)

    return feature_importance_df
```

```
In [ ]: time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
```

```
In [ ]: params = {"random_state":42,"n_jobs":-1}
print(name)
mlflow_logging_and_metric_printing(model,name,"Imbalanced",X_train_imb,y_train_imb,X
```

### Simple\_Logistic\_Regression\_on\_Imbalanced\_Dataset

#### Train Metrics:

Accuracy\_train: 0.9742  
Precision\_train: 0.9752  
Recall\_train: 0.9691  
F1\_score\_train: 0.9722  
F2\_score\_train: 0.9703  
Roc\_auc\_train: 0.9965  
Pr\_auc\_train: 0.9965

#### Test Metrics:

Accuracy\_test: 0.8606  
Precision\_test: 0.7711  
Recall\_test: 0.9962  
F1\_score\_test: 0.8693  
F2\_score\_test: 0.9413  
Roc\_auc\_test: 0.9924  
Pr\_auc\_test: 0.9923

#### Tuning Metrics:

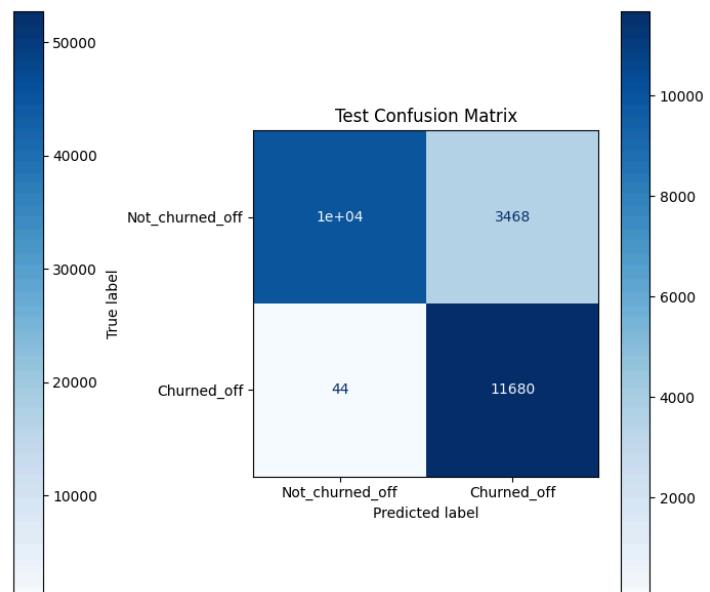
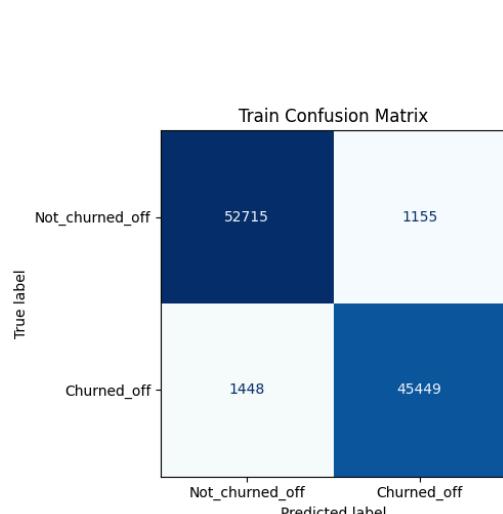
hyper\_parameter\_tuning\_best\_est\_score: 0.0000

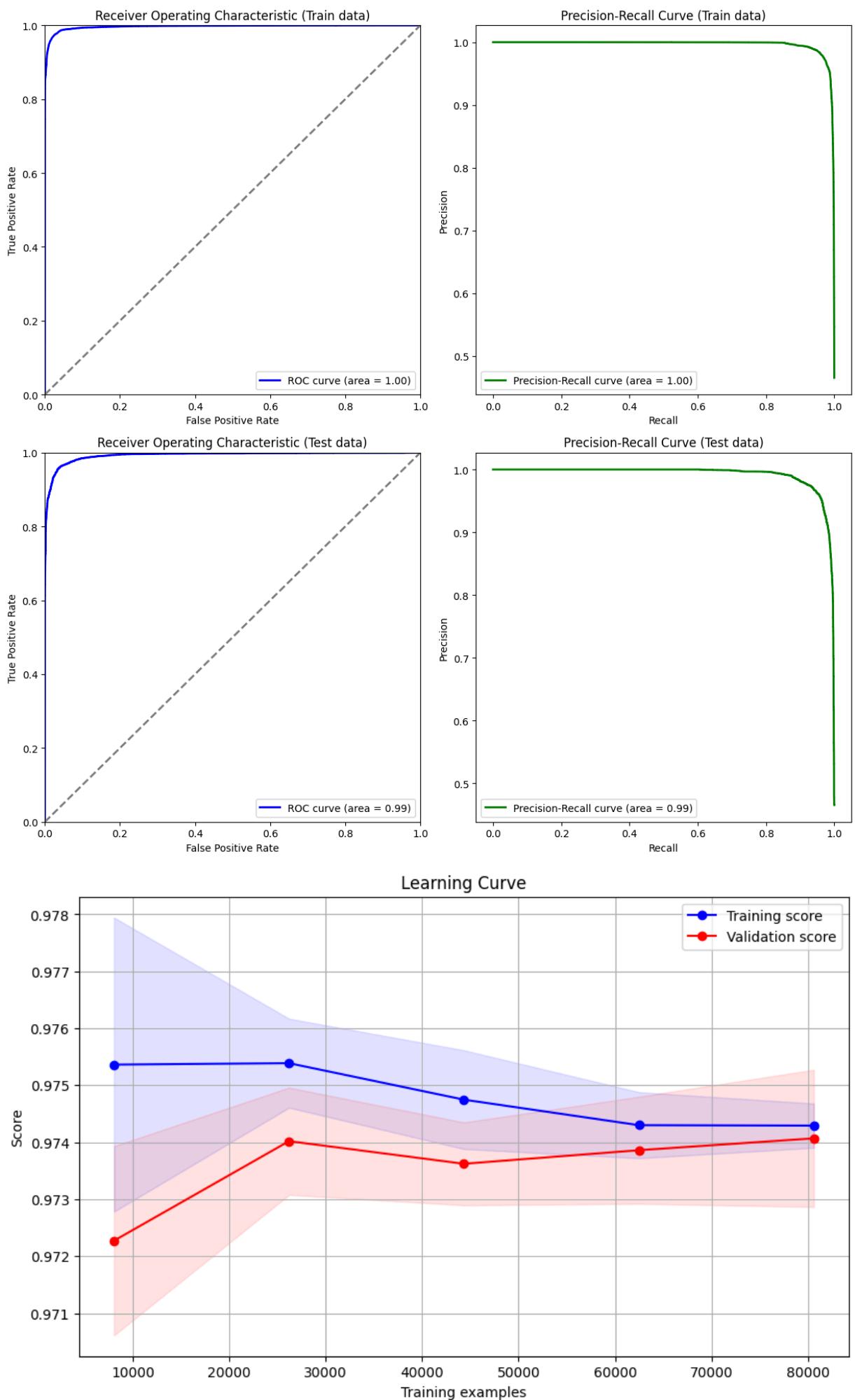
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.97	0.98	0.98	53870
1	0.98	0.97	0.97	46897
accuracy			0.97	100767
macro avg	0.97	0.97	0.97	100767
weighted avg	0.97	0.97	0.97	100767

#### Test Classification Report:

	precision	recall	f1-score	support
0	1.00	0.74	0.85	13469
1	0.77	1.00	0.87	11724
accuracy			0.86	25193
macro avg	0.88	0.87	0.86	25193
weighted avg	0.89	0.86	0.86	25193





```
2025/05/16 17:58:29 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## Simple\_Logistic\_Regresion\_on\_Balanced Dataset

```
In [ ]: # Model details
name = "Simple_Logistic_Regresion_on_Balanced_Dataset"
model = LogisticRegression(random_state=42,n_jobs = -1)
bal_type = "Balanced"

# Record training time
```

```
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")

# Log time and feature importances into df
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
```

```
Model: Simple_Logistic_Regesssion_on_balanced_Dataset
Training Time: 3.8236 seconds
Testing Time: 0.0769 seconds
```

```
In [ ]: params = {"random_state":42,"n_jobs":-1}
print(name)
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_tes
```

### Simple\_Logistic\_Regresssion\_on\_balanced\_Dataset

#### Train Metrics:

Accuracy\_train: 0.9745  
Precision\_train: 0.9768  
Recall\_train: 0.9722  
F1\_score\_train: 0.9745  
F2\_score\_train: 0.9731  
Roc\_auc\_train: 0.9966  
Pr\_auc\_train: 0.9969

#### Test Metrics:

Accuracy\_test: 0.8519  
Precision\_test: 0.7599  
Recall\_test: 0.9967  
F1\_score\_test: 0.8623  
F2\_score\_test: 0.9382  
Roc\_auc\_test: 0.9922  
Pr\_auc\_test: 0.9921

#### Tuning Metrics:

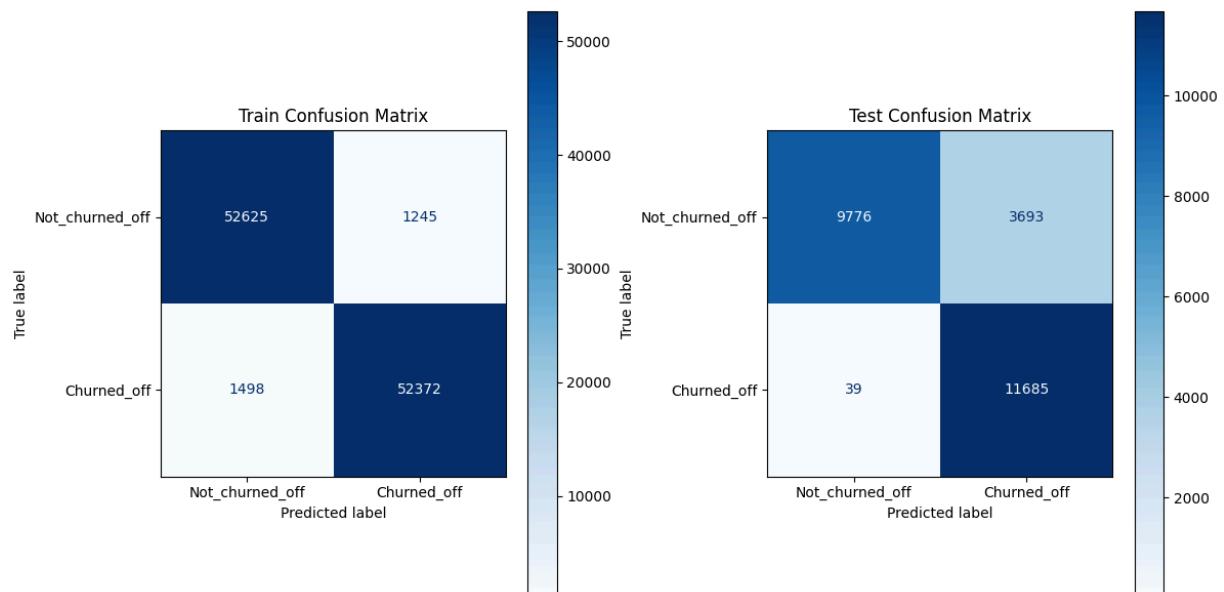
hyper\_parameter\_tuning\_best\_est\_score: 0.0000

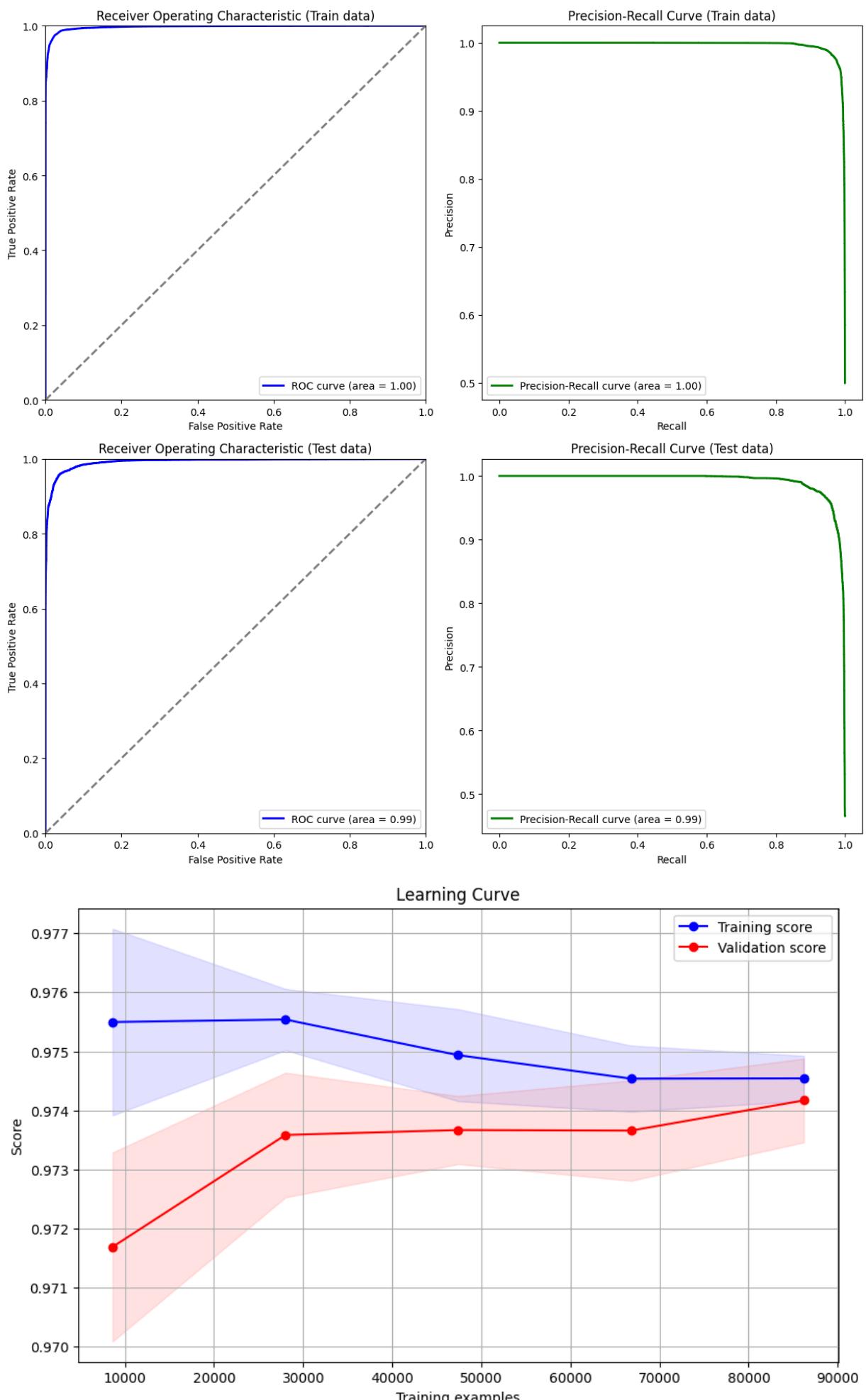
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.97	0.98	0.97	53870
1	0.98	0.97	0.97	53870
accuracy			0.97	107740
macro avg	0.97	0.97	0.97	107740
weighted avg	0.97	0.97	0.97	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	1.00	0.73	0.84	13469
1	0.76	1.00	0.86	11724
accuracy			0.85	25193
macro avg	0.88	0.86	0.85	25193
weighted avg	0.89	0.85	0.85	25193





```
2025/05/16 17:59:09 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## all\_logged\_metrics function

```
In [ ]: def all_logged_metrics():
    # Set the experiment name
    experiment_name = "nadb_binary"

    # Get the experiment details
    experiment = mlflow.get_experiment_by_name(experiment_name)
```

```

experiment_id = experiment.experiment_id

# Retrieve all runs from the experiment
runs_df = mlflow.search_runs(experiment_ids=[experiment_id])

# Extract metrics columns
metrics_columns = [col for col in runs_df.columns if col.startswith("metrics.")]
metrics_df = runs_df[metrics_columns]

# Add run_name as a column
metrics_df['run_name'] = runs_df['tags.mlflow.runName']
metrics_df["bal_type"] = runs_df["params.bal_type"]

# Combine all params into a dictionary
params_columns = [col for col in runs_df.columns if col.startswith("params.")]
metrics_df["params_dict"] = runs_df[params_columns].apply(lambda row: row.dropna())

# Sort remaining columns alphabetically
sorted_columns = sorted(metrics_columns)

# Rearrange columns: first column is 'run_name', followed by 'bal_type', 'params_dict', then the sorted metrics
ordered_columns = ['run_name', "bal_type", "params_dict"] + sorted_columns
metrics_df = metrics_df[ordered_columns]

# If you want to view it in a more readable format
return metrics_df

```

## Visualising all\_logged\_metrics, time\_df and feature\_imporatance df

### All\_logged\_metrics\_plots

In [ ]: # Example usage  
`pd.set_option('display.max_colwidth', None) # Show full content in cells  
all_logged_metrics_df = all_logged_metrics()  
all_logged_metrics_df.head()`

Out[ ]:

		run_name	bal_type	params_dict	metri
0	Simple_Logistic_Regession_on_balanced_Dataset		Balanced	{'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.n_jobs': '-1'}	
1	Simple_Logistic_Regression_on_Imbalanced_Dataset		Imbalanced	{'params.bal_type': 'Imbalanced', 'params.random_state': '42', 'params.n_jobs': '-1'}	
2		binary_kmeans_adv	None	{'params.k': '2', 'params.alpha': '0.01', 'params.beta': '1.0', 'params.gamma': '0.025'}	
3		binary_gmm	None	{'params.random_state': '42', 'params.n_components': '2', 'params.verbose': '0'}	9.8780
4		binary_knn	None	{'params.n_jobs': '-1', 'params.n_neighbors': '5'}	

In [ ]: `pd.reset_option('display.max_colwidth')`

```
In [ ]: def all_logged_metrics_df_plots(df):
    # Melt the DataFrame to make it easier to plot
    metrics_columns = df.columns[3:] # Select only metric columns
    df_melted = df.melt(id_vars=['run_name', 'bal_type'],
                         value_vars=metrics_columns,
                         var_name='Metric',
                         value_name='Value')

    # Create subplots with 8 rows and 2 columns
    n_rows = 8
    n_cols = 2
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 30))
    axes = axes.flatten() # Flatten the axes array for easy iteration

    # Plot each metric in a separate subplot
    for i, metric in enumerate(metrics_columns):
        sns.barplot(x='bal_type', y='Value', hue='run_name', data=df_melted)
        axes[i].set_title(metric)
        axes[i].set_xlabel('')
        axes[i].set_ylabel('Value')
        axes[i].legend_.remove() # Remove Legend from individual plots

    # Remove any unused subplots
    for j in range(len(metrics_columns), len(axes)):
        fig.delaxes(axes[j])

    # Add a single legend for all subplots
    handles, labels = axes[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc='upper center', ncol=3, bbox_to_anchor=(0.5, 1.0))

    # Adjust Layout
    plt.tight_layout()
    plt.subplots_adjust(top=0.95) # Adjust top to make space for the global Legend
    plt.show()

    # Example usage with your DataFrame
    # all_logged_metrics_df_plots(all_logged_metrics_df)
```

## time\_df\_plots

```
In [ ]: time_df
```

	Model	bal_type	Training_Time	Testing_Time
0	Simple_Logistic_Regression_on_Imbalanced_Dataset	Imbalanced	3.573384	0.042536
1	Simple_Logistic_Regession_on_balanced_Dataset	Balanced	3.823588	0.076877

```
In [ ]: def time_df_plots(time_df):
    # Create subplots for Training Time and Testing Time
    fig, axes = plt.subplots(1, 3, figsize=(25, 10))

    # Plot Training Time
    sns.barplot(x='Model', y='Training_Time', hue='bal_type', data=time_df, ax=axes[0])
    axes[0].set_title('Training Time by Model and Dataset Balance')
    axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45, ha='right')
    axes[0].set_xlabel('Model')
    axes[0].set_ylabel('Training Time (seconds)')

    # Plot Testing Time
    sns.barplot(x='Model', y='Testing_Time', hue='bal_type', data=time_df, ax=axes[1])
    axes[1].set_title('Testing Time by Model and Dataset Balance')
    axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45, ha='right')
    axes[1].set_xlabel('Model')
    axes[1].set_ylabel('Testing Time (seconds)')

    # Plot Tuning Time
    sns.barplot(x='Model', y='Tuning_Time', hue='bal_type', data=time_df, ax=axes[2])
    axes[2].set_title('Tuning Time by Model and Dataset Balance')
    axes[2].set_xticklabels(axes[2].get_xticklabels(), rotation=45, ha='right')
    axes[2].set_xlabel('Model')
```

```

axes[2].set_ylabel('Tuning Time (seconds)')

# Adjust Layout
plt.tight_layout()

# Show the plot
plt.show()
# time_df_plots(time_df)

```

## feature\_importance\_df\_plots

In [ ]: feature\_importance\_df

	Simple_Logistic_Regression_on_Imbalanced_Dataset	Simple_Logistic_R
<b>duration</b>	-0.355736	
<b>srcbytes</b>	0.480815	
<b>dstbytes</b>	0.753895	
<b>wrongfragment</b>	2.362582	
<b>urgent</b>	0.504681	
...	...	
<b>binary_one_class_svm_df</b>	0.427894	
<b>binary_dbSCAN_labels</b>	-0.032274	
<b>binary_knn_kth_distance</b>	-0.803444	
<b>binary_gmm_score</b>	1.027387	
<b>binary_kmeans_adv</b>	2.282959	

62 rows × 2 columns

```

In [ ]: def feature_importance_plots(feature_importance_df):
    # Reset the index to get the features as a column
    feature_importance_df_reset = feature_importance_df.reset_index()

    # Melt the DataFrame to have a Long-form DataFrame suitable for seaborn
    feature_importance_melted = feature_importance_df_reset.melt(id_vars='index',
                                                                var_name='Model',
                                                                value_name='Importa

    # Create a seaborn horizontal barplot
    plt.figure(figsize=(10, 30)) # Adjust the figure size to be taller
    sns.barplot(x='Importance', y='index', hue='Model', data=feature_importance_melt

    # Set title and Labels
    plt.title('Feature Importance Comparison')
    plt.xlabel('Importance Value')
    plt.ylabel('Features')
    plt.legend(loc = "lower right")
    # Display the plot
    plt.tight_layout()
    plt.show()

    # Example usage with your feature_importance_df
    # feature_importance_plots(feature_importance_df)

```

In [ ]: time\_df.to\_csv("time\_df", index = False)
feature\_importance\_df.to\_csv("feature\_importance\_df")

In [ ]: time\_df = pd.read\_csv("time\_df")
feature\_importance\_df = pd.read\_csv("feature\_importance\_df").set\_index("Unnamed: 0")
feature\_importance\_df.index.name = None

## Observation

We have created predefined functions for our easeness to log and print the metrics of the models

Basically we have to do Grid or Random search CV, Then find the best estimator. On that best estimator, we can apply

`mlflow_logging_and_metric_printing` function,  
`log_time_and_feature_importances_df` function

if we want to visualize all metrics, time and feature importances for all the models, then use `all_logged_metrics` function, get the df from that function and use it in `All_logged_metrics_plots` function. We can use 'time\_df\_plots' and 'feature\_importance\_df\_plots' functions also.

plots are modified at the end for better visualization

## Binary Classification using "attack\_or\_normal" as Target

### LOGISTIC REGRESSION MODEL

#### Imbalanced Dataset

#### Hyper parameter Tuning

```
In [ ]: # Define the parameter grid for RandomizedSearchCV
start_tune_time = time.time()
param_dist = {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'C': stats.uniform(loc=0.01, scale=5000-0.01), # Uniform distribution from 0.01
    'solver': ['saga'], # saga solver supports all penalties
    'class_weight': ['balanced']
}

# Initialize the Logistic Regression model
logReg = LogisticRegression(max_iter=100,random_state=42)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator= logReg,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters found: {'C': np.float64(1872.706848835624), 'class_weight': 'balance
d', 'penalty': 'l1', 'solver': 'saga'}
Best score: 0.9633511038589916
Tuning_time 101.18594193458557
```

#### Logging Best Logistic Regression Model into MLFLOW

```
In [ ]: # Model details
name = "Tuned_Logistic_Regression_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes
```

```

Model: Tuned_Logistic_Regression_on_Imbalanced_Dataset
params: {'C': np.float64(1872.706848835624), 'class_weight': 'balanced', 'penalty': 'l1', 'solver': 'saga'}
Training Time: 29.1182 seconds
Testing Time: 0.1041 seconds
Tuning Time: 101.1859 seconds
Train Metrics:
Accuracy_train: 0.9643
Precision_train: 0.9691
Recall_train: 0.9536
F1_score_train: 0.9613
F2_score_train: 0.9567
Roc_auc_train: 0.9955
Pr_auc_train: 0.9953

Test Metrics:
Accuracy_test: 0.9427
Precision_test: 0.8954
Recall_test: 0.9929
F1_score_test: 0.9416
F2_score_test: 0.9718
Roc_auc_test: 0.9951
Pr_auc_test: 0.9950

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9634

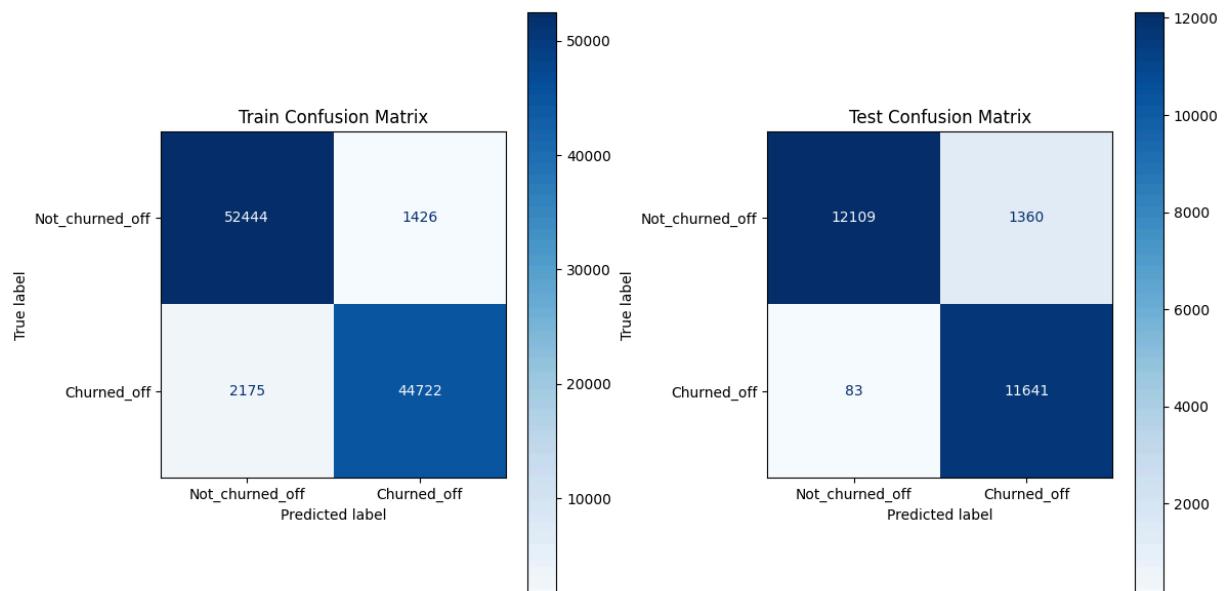
```

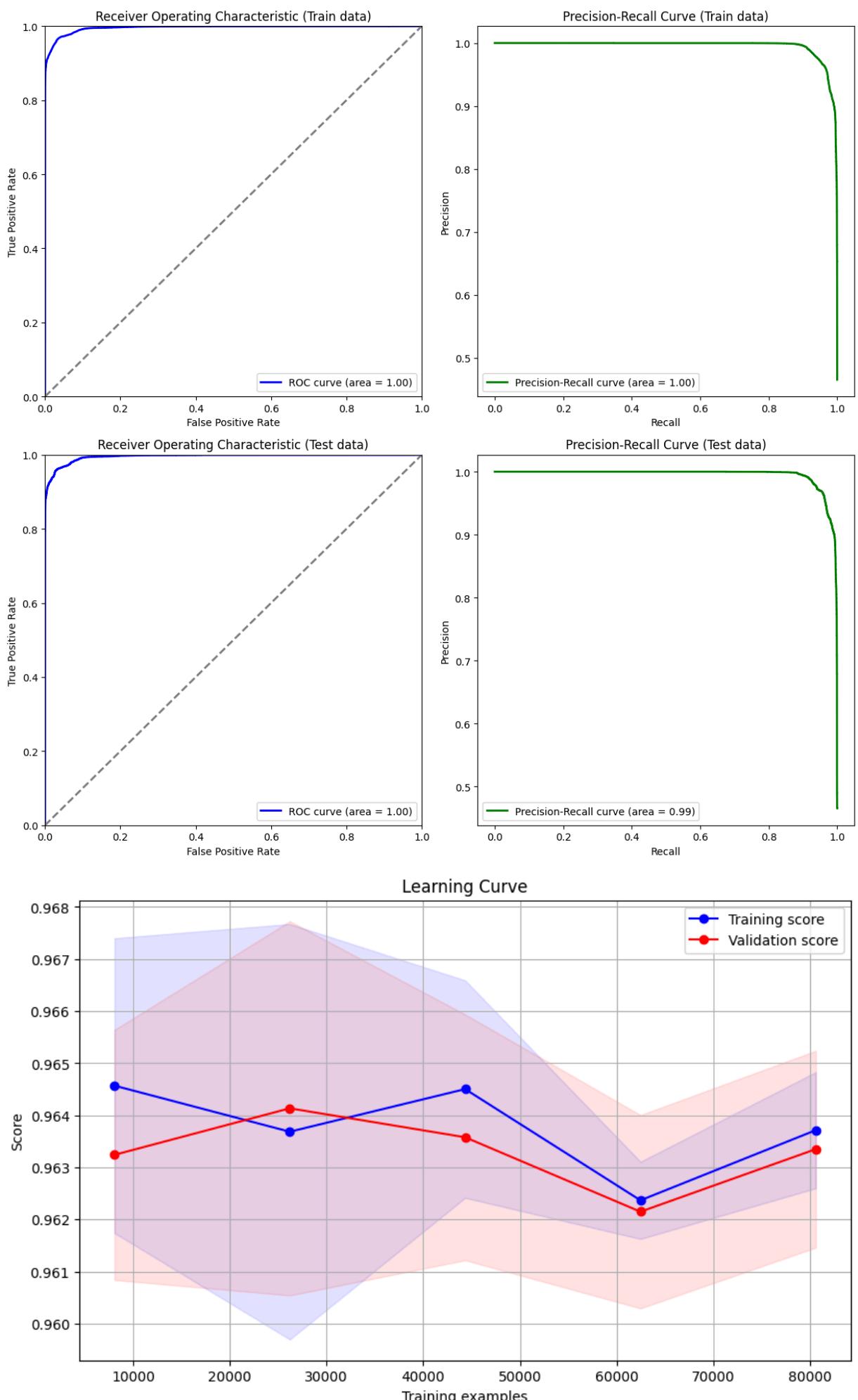
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.96	0.97	0.97	53870
1	0.97	0.95	0.96	46897
accuracy			0.96	100767
macro avg	0.96	0.96	0.96	100767
weighted avg	0.96	0.96	0.96	100767

#### Test Classification Report:

	precision	recall	f1-score	support
0	0.99	0.90	0.94	13469
1	0.90	0.99	0.94	11724
accuracy			0.94	25193
macro avg	0.94	0.95	0.94	25193
weighted avg	0.95	0.94	0.94	25193





```
2025/05/16 18:03:21 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Define the parameter grid for RandomizedSearchCV
start_tune_time = time.time()
param_dist = {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'C': np.logspace(-4, 4, 3),
    'max_iter': [100, 200, 500],
    'tol': [1e-2, 1e-3, 1e-4]
}
```

```

'C': stats.uniform(loc=0.01, scale=5000-0.01), # Uniform distribution from 0.01
'solver': ['saga'], # saga solver supports all penalties
'class_weight': ['balanced']
}

# Initialize the Logistic Regression model
logReg = LogisticRegression(max_iter=100,random_state = 42)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator= logReg,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time",tuning_time)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits  
Best parameters found: {'C': np.float64(1872.706848835624), 'class\_weight': 'balance d', 'penalty': 'l1', 'solver': 'saga'}  
Best score: 0.9631613142751068  
Tuning\_time 113.47426700592041

## Logging Best Logistic Regression Model into MLFLOW

```

In [ ]: # Model details
name = "Tuned_Logistic_Regression_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_te

```

```

Model: Tuned_Logistic_Regression_on_Balanced_Dataset
params: {'C': np.float64(1872.706848835624), 'class_weight': 'balanced', 'penalty': 'l1', 'solver': 'saga'}
Training Time: 25.0553 seconds
Testing Time: 0.0421 seconds
Tuning Time: 113.4743 seconds
Train Metrics:
Accuracy_train: 0.9641
Precision_train: 0.9734
Recall_train: 0.9544
F1_score_train: 0.9638
F2_score_train: 0.9581
Roc_auc_train: 0.9956
Pr_auc_train: 0.9959

Test Metrics:
Accuracy_test: 0.9424
Precision_test: 0.8952
Recall_test: 0.9926
F1_score_test: 0.9414
F2_score_test: 0.9714
Roc_auc_test: 0.9951
Pr_auc_test: 0.9950

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9632

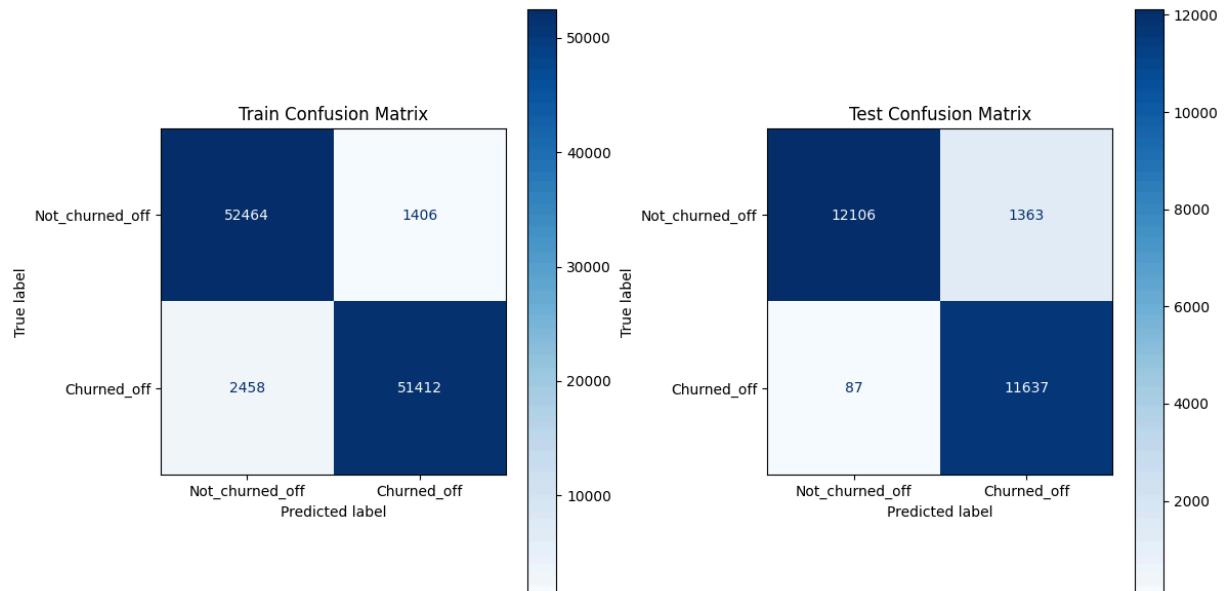
```

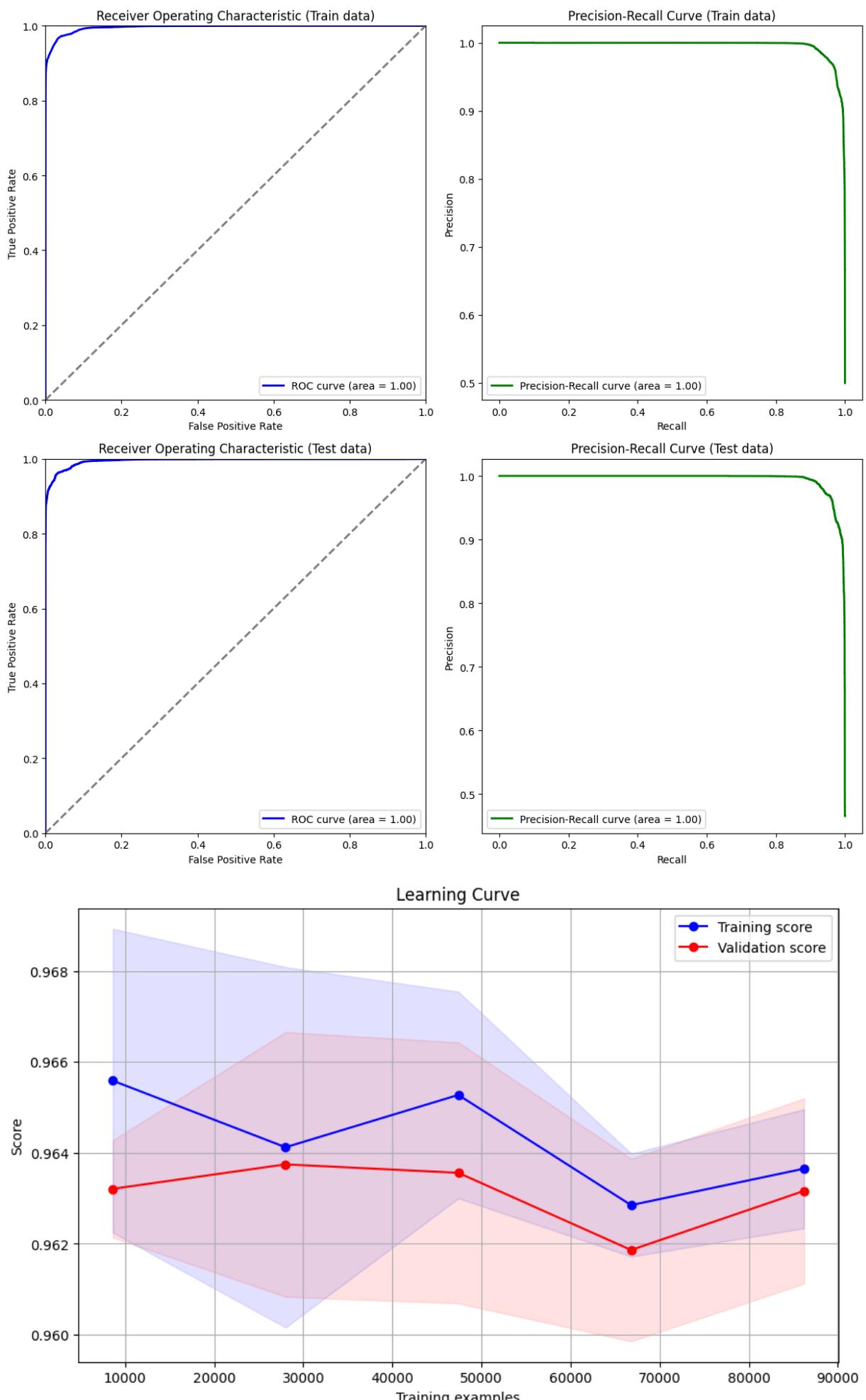
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.96	0.97	0.96	53870
1	0.97	0.95	0.96	53870
accuracy			0.96	107740
macro avg	0.96	0.96	0.96	107740
weighted avg	0.96	0.96	0.96	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	0.99	0.90	0.94	13469
1	0.90	0.99	0.94	11724
accuracy			0.94	25193
macro avg	0.94	0.95	0.94	25193
weighted avg	0.95	0.94	0.94	25193





```
2025/05/16 18:07:24 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## MULTI\_LAYER\_PERCEPTRON NEURAL NETWORK MODEL

Imbalanced Dataset

Hyper parameter Tuning

```
In [ ]: # Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'hidden_layer_sizes': [(50,), (100,), (100, 50), (50, 50, 50)],
    'activation': ['identity', 'logistic', 'tanh', 'relu'],
    'solver': ['adam'], # 'adam' is suitable for all activation functions
    'alpha': stats.uniform(0.0001, 5000), # Uniform distribution from 0.0001 to 500
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
    'learning_rate_init': stats.uniform(0.0001, 0.1), # Small Learning rates for better convergence
    'max_iter': stats.randint(200, 1000), # Increase max_iter for better convergence
    'early_stopping': [True, False], # Use early stopping to prevent overfitting
    'tol': [1e-4, 1e-5, 1e-6], # Lower tolerance for more precise convergence
}

# Initialize the MLPClassifier
mlp = MLPClassifier(max_iter=100, random_state=42)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=mlp,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits  
 Best parameters found: {'activation': 'logistic', 'alpha': np.float64(3.893929205071642), 'early\_stopping': False, 'hidden\_layer\_sizes': (50,), 'learning\_rate': 'constant', 'learning\_rate\_init': np.float64(0.030524224295953774), 'max\_iter': 221, 'solver': 'adam', 'tol': 0.0001}  
 Best score: 0.9417267795184443  
 Tuning\_time 207.44663906097412

## Logging Best MLPClassifier Model into MLFLOW

```
In [ ]: # Model details
name ="Tuned_MLPClassifier_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
```

```

print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes

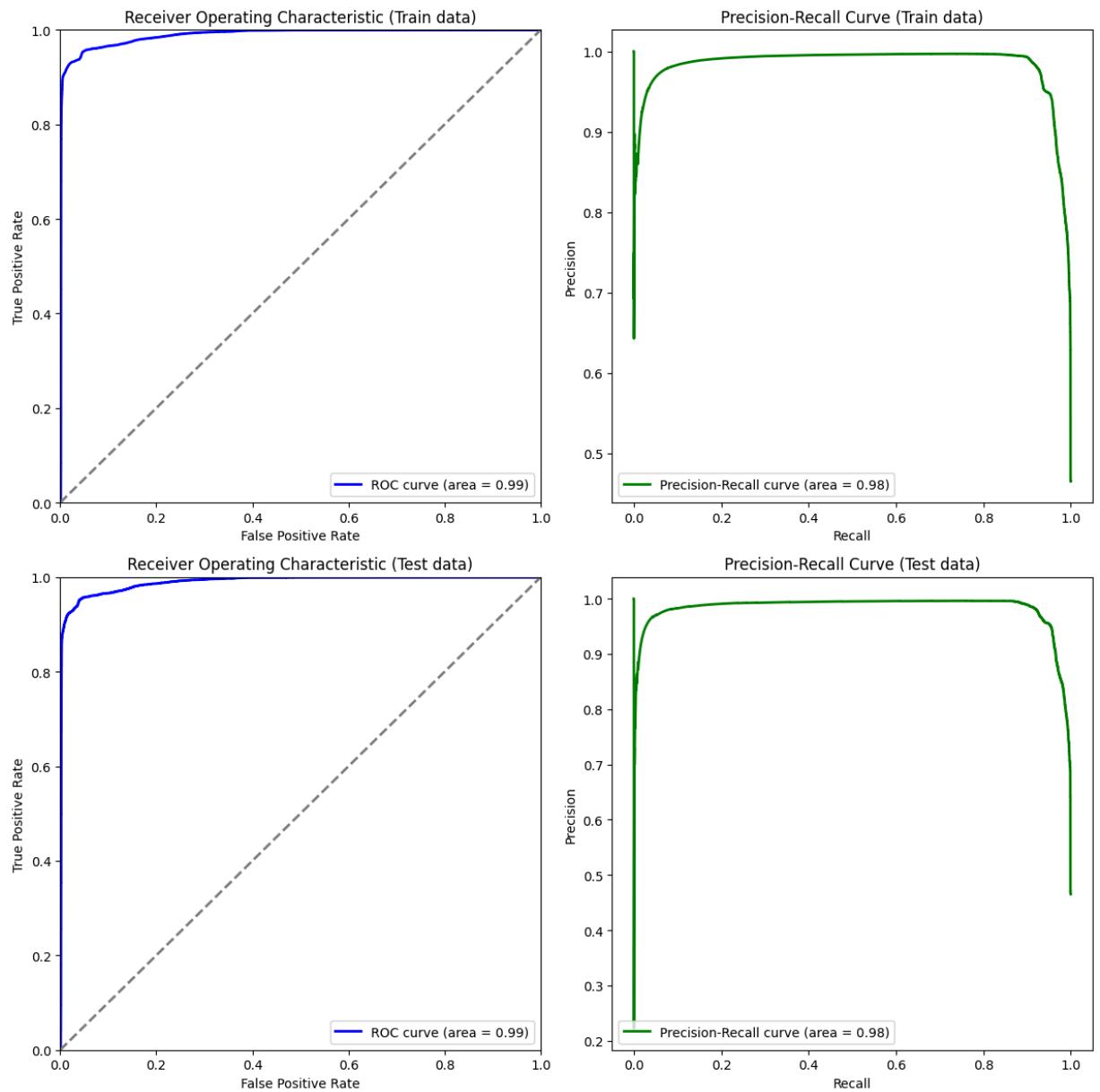
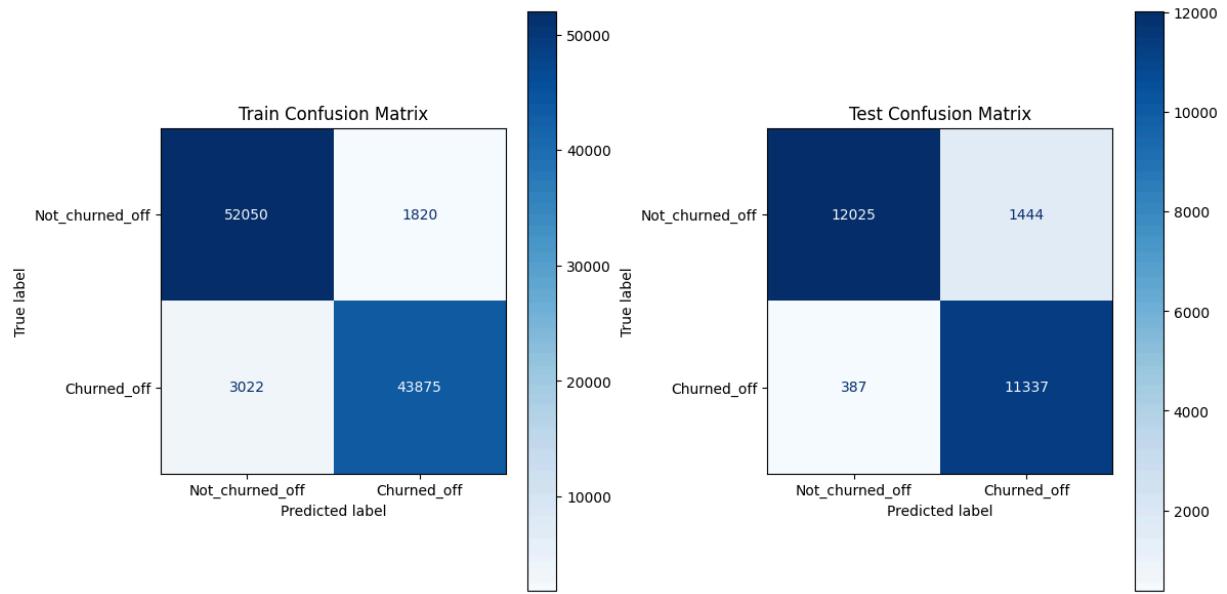
```

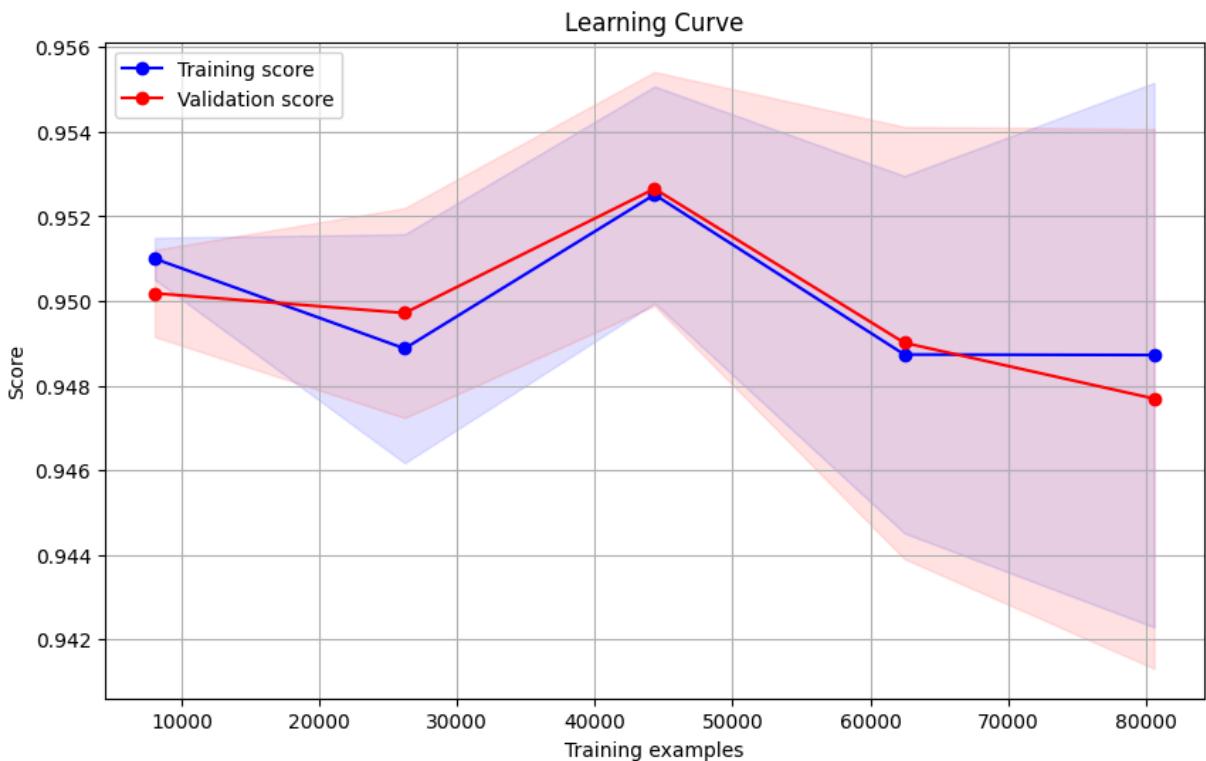
Model: Tuned\_MLPClassifier\_on\_Imbalanced\_Dataset  
 params: {'activation': 'logistic', 'alpha': np.float64(3.893929205071642), 'early\_stopping': False, 'hidden\_layer\_sizes': (50,), 'learning\_rate': 'constant', 'learning\_rate\_init': np.float64(0.030524224295953774), 'max\_iter': 221, 'solver': 'adam', 'tol': 0.0001}  
 Training Time: 18.3045 seconds  
 Testing Time: 0.8525 seconds  
 Tuning Time: 207.4466 seconds  
 Train Metrics:  
 Accuracy\_train: 0.9519  
 Precision\_train: 0.9602  
 Recall\_train: 0.9356  
 F1\_score\_train: 0.9477  
 F2\_score\_train: 0.9404  
 Roc\_auc\_train: 0.9886  
 Pr\_auc\_train: 0.9815  
 Test Metrics:  
 Accuracy\_test: 0.9273  
 Precision\_test: 0.8870  
 Recall\_test: 0.9670  
 F1\_score\_test: 0.9253  
 F2\_score\_test: 0.9499  
 Roc\_auc\_test: 0.9887  
 Pr\_auc\_test: 0.9807  
 Tuning Metrics:  
 hyper\_parameter\_tuning\_best\_est\_score: 0.9417  
 Train Classification Report:  

	precision	recall	f1-score	support
0	0.95	0.97	0.96	53870
1	0.96	0.94	0.95	46897
accuracy			0.95	100767
macro avg	0.95	0.95	0.95	100767
weighted avg	0.95	0.95	0.95	100767

 Test Classification Report:  

	precision	recall	f1-score	support
0	0.97	0.89	0.93	13469
1	0.89	0.97	0.93	11724
accuracy			0.93	25193
macro avg	0.93	0.93	0.93	25193
weighted avg	0.93	0.93	0.93	25193





2025/05/16 18:12:45 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'hidden_layer_sizes': [(50,), (100,), (100, 50), (50, 50, 50)],
    'activation': ['identity', 'logistic', 'tanh', 'relu'],
    'solver': ['adam'], # 'adam' is suitable for all activation functions
    'alpha': stats.uniform(0.0001, 5000), # Uniform distribution from 0.0001 to 500
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
    'learning_rate_init': stats.uniform(0.0001, 0.1), # Small learning rates for better convergence
    'max_iter': stats.randint(200, 1000), # Increase max_iter for better convergence
    'early_stopping': [True, False], # Use early stopping to prevent overfitting
    'tol': [1e-4, 1e-5, 1e-6], # Lower tolerance for more precise convergence
}

# Initialize the MLPClassifier
mlp = MLPClassifier(max_iter=100, random_state=42)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=mlp,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters found: {'activation': 'logistic', 'alpha': np.float64(3.8939292050716
42), 'early_stopping': False, 'hidden_layer_sizes': (50,), 'learning_rate': 'constan
t', 'learning_rate_init': np.float64(0.030524224295953774), 'max_iter': 221, 'solve
r': 'adam', 'tol': 0.0001}
Best score: 0.9511045108594767
Tuning_time 244.1995656490326
```

## Logging Best MLPClassifier Model into MLFLOW

```
In [ ]: # Model details
name ="Tuned_MLPClassifier_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_tes
```

```

Model: Tuned_MLPClassifier_on_Balanced_Dataset
params: {'activation': 'logistic', 'alpha': np.float64(3.893929205071642), 'early_stopping': False, 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'learning_rate_init': np.float64(0.030524224295953774), 'max_iter': 221, 'solver': 'adam', 'tol': 0.0001}
Training Time: 36.7089 seconds
Testing Time: 0.8449 seconds
Tuning Time: 244.1996 seconds
Train Metrics:
Accuracy_train: 0.9401
Precision_train: 0.9206
Recall_train: 0.9634
F1_score_train: 0.9415
F2_score_train: 0.9545
Roc_auc_train: 0.9882
Pr_auc_train: 0.9889

Test Metrics:
Accuracy_test: 0.9157
Precision_test: 0.8603
Recall_test: 0.9775
F1_score_test: 0.9152
F2_score_test: 0.9516
Roc_auc_test: 0.9854
Pr_auc_test: 0.9801

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9511

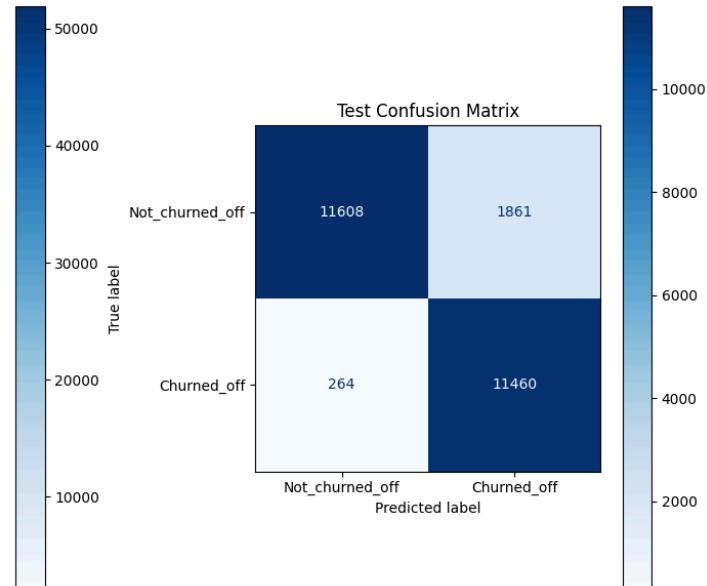
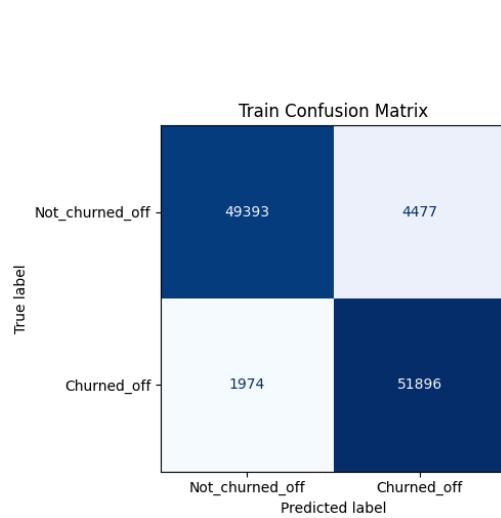
```

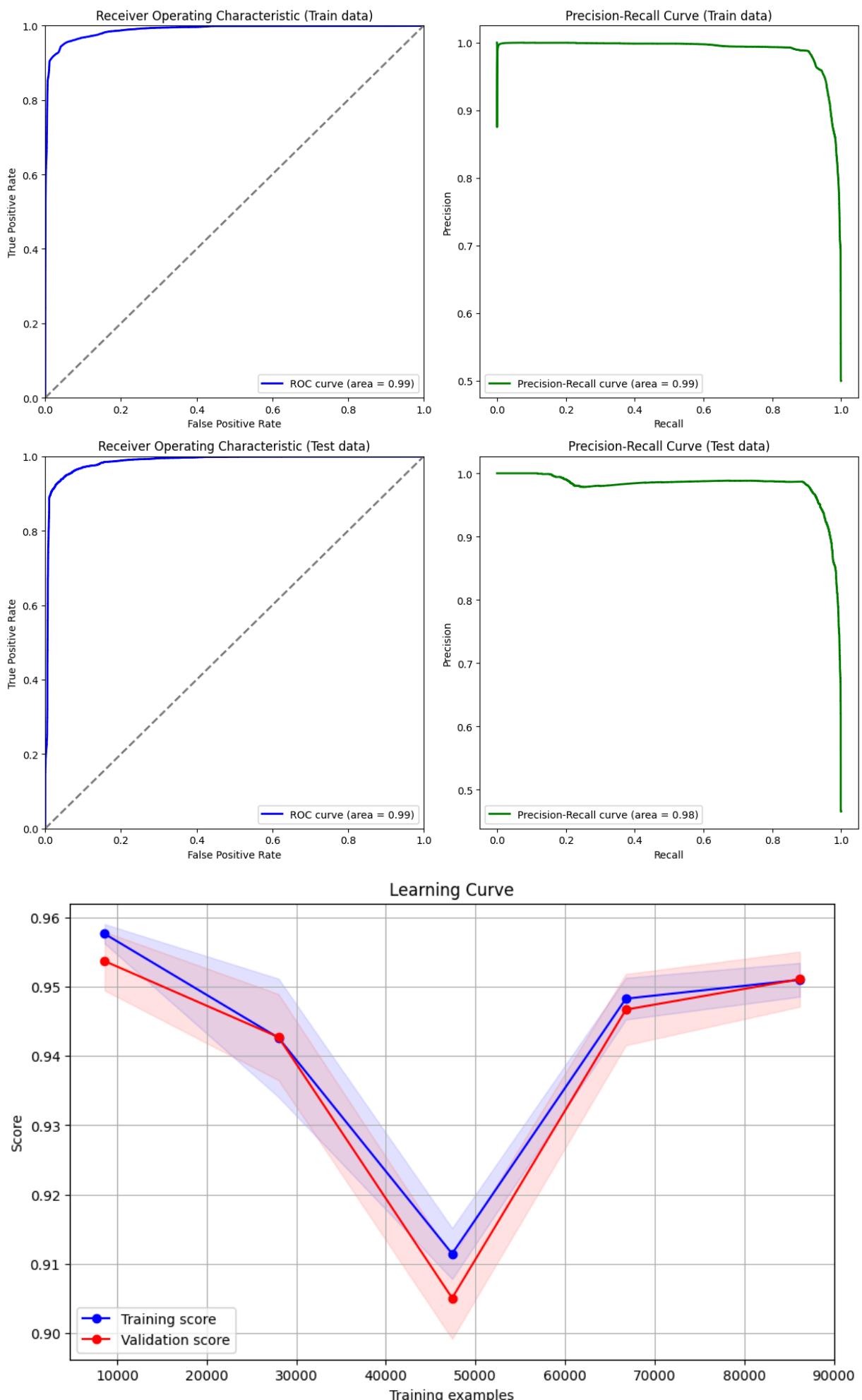
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.96	0.92	0.94	53870
1	0.92	0.96	0.94	53870
accuracy			0.94	107740
macro avg	0.94	0.94	0.94	107740
weighted avg	0.94	0.94	0.94	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	0.98	0.86	0.92	13469
1	0.86	0.98	0.92	11724
accuracy			0.92	25193
macro avg	0.92	0.92	0.92	25193
weighted avg	0.92	0.92	0.92	25193





2025/05/16 18:19:08 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging is completed

## K NEAREST NEIGHBORS MODEL

### Imbalanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'n_neighbors': stats.randint(1, 35),
    'metric': ['euclidean', 'manhattan']
}

# Initialize the KNN model
knn = KNeighborsClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=knn,
    param_distributions=param_dist,
    n_iter=3, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)
```

Fitting 5 folds for each of 3 candidates, totalling 15 fits  
 Best parameters found: {'metric': 'euclidean', 'n\_neighbors': 8}  
 Best score: 0.9949685914956594  
 Tuning\_time 130.5499165058136

### Logging Best K Nearest Neighbor Model into MLFLOW

```
In [ ]: # Model details
name = "Tuned_KNN_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_te
```

Model: Tuned\_KNN\_on\_Imbalanced\_Dataset

params: {'metric': 'euclidean', 'n\_neighbors': 8}

Training Time: 0.0876 seconds

Testing Time: 84.4574 seconds

Tuning Time: 130.5499 seconds

Train Metrics:

Accuracy\_train: 0.9963

Precision\_train: 0.9971

Recall\_train: 0.9948

F1\_score\_train: 0.9960

F2\_score\_train: 0.9953

Roc\_auc\_train: 1.0000

Pr\_auc\_train: 1.0000

Test Metrics:

Accuracy\_test: 0.9940

Precision\_test: 0.9951

Recall\_test: 0.9919

F1\_score\_test: 0.9935

F2\_score\_test: 0.9925

Roc\_auc\_test: 0.9989

Pr\_auc\_test: 0.9991

Tuning Metrics:

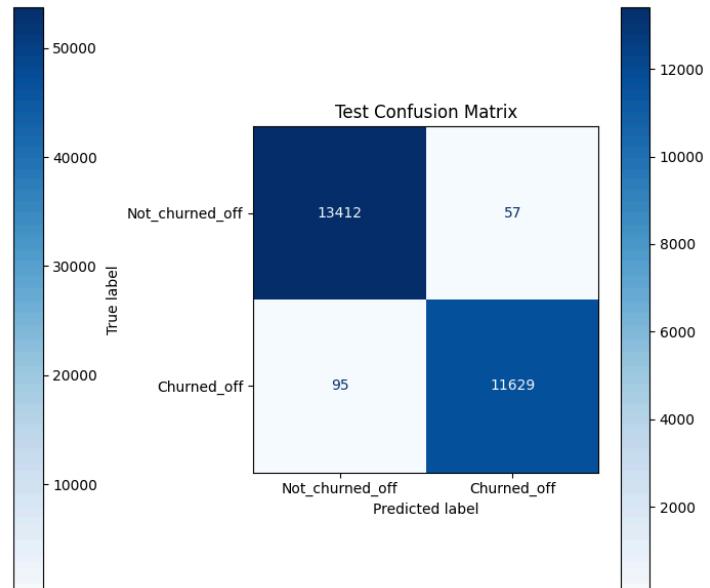
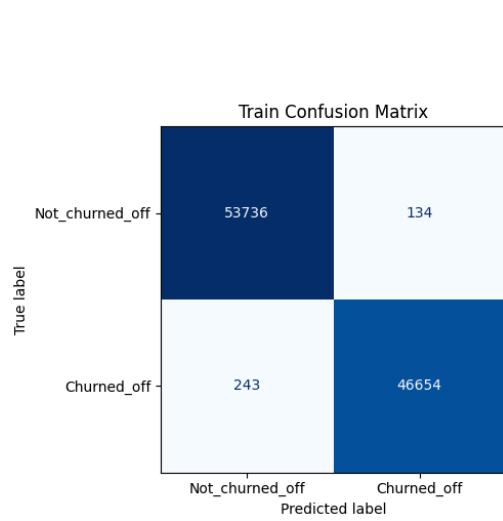
hyper\_parameter\_tuning\_best\_est\_score: 0.9950

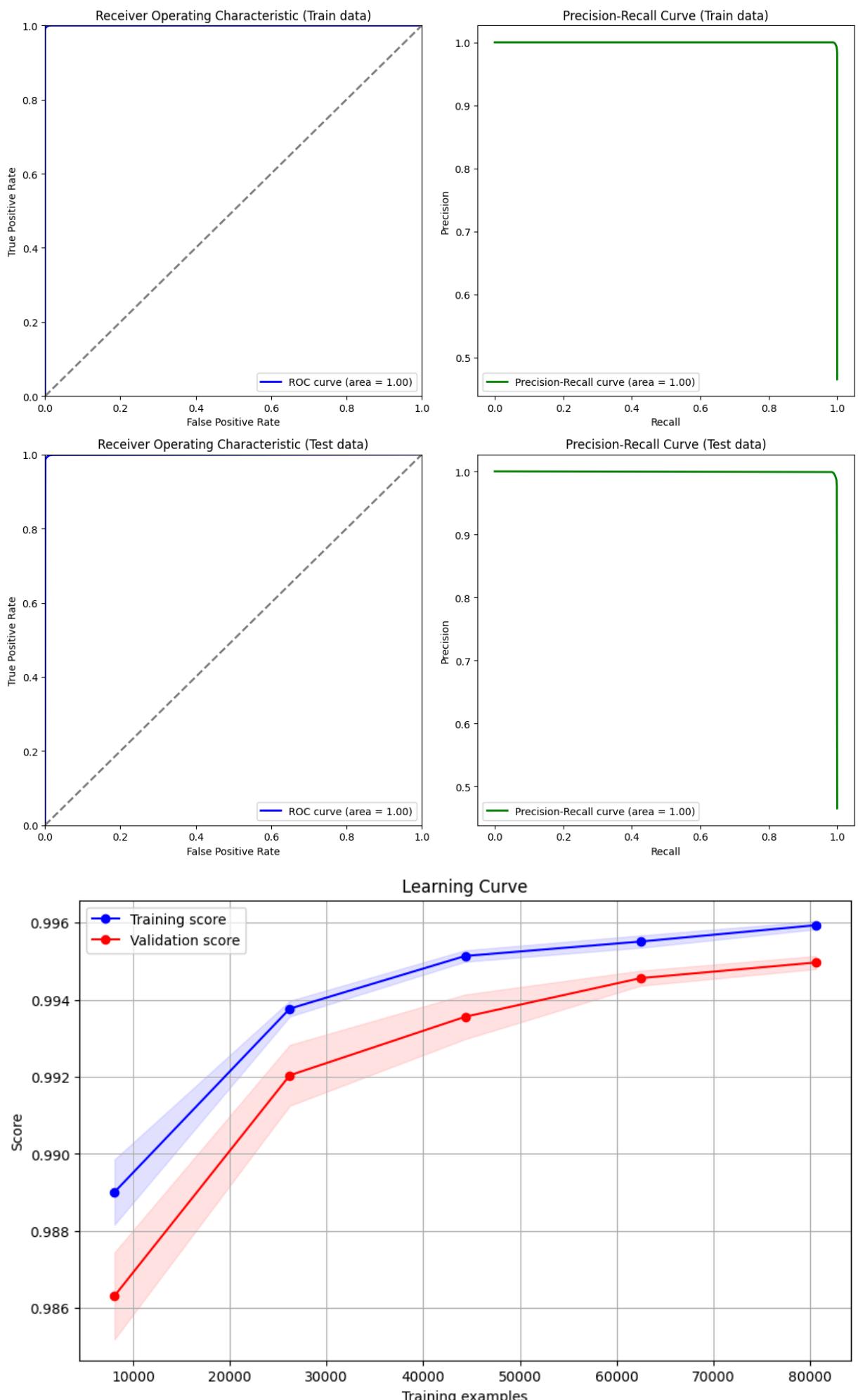
Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	0.99	1.00	46897
accuracy			1.00	100767
macro avg	1.00	1.00	1.00	100767
weighted avg	1.00	1.00	1.00	100767

Test Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	0.99	13469
1	1.00	0.99	0.99	11724
accuracy			0.99	25193
macro avg	0.99	0.99	0.99	25193
weighted avg	0.99	0.99	0.99	25193





```
2025/05/16 18:33:57 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'n_neighbors': stats.randint(1, 35),
```

```

        'metric': ['euclidean', 'manhattan']
    }

# Initialize the KNN model
knn = KNeighborsClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=knn,
    param_distributions=param_dist,
    n_iter=3, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits  
Best parameters found: {'metric': 'euclidean', 'n\_neighbors': 8}  
Best score: 0.9950714683497308  
Tuning\_time 163.29809188842773

### Logging Best K Nearest Neighbor Model into MLFLOW

```

In [ ]: # Model details
name = "Tuned_KNN_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df, feature_importance_df = log_time_and_feature_importances_df(time_df, feature_
mlflow_logging_and_metric_printing(model, name, bal_type, X_train_bal, y_train_bal, X_tes

```

```

Model: Tuned_KNN_on_Balanced_Dataset
params: {'metric': 'euclidean', 'n_neighbors': 8}
Training Time: 0.1110 seconds
Testing Time: 99.0554 seconds
Tuning Time: 163.2981 seconds
Train Metrics:
Accuracy_train: 0.9965
Precision_train: 0.9972
Recall_train: 0.9958
F1_score_train: 0.9965
F2_score_train: 0.9961
Roc_auc_train: 1.0000
Pr_auc_train: 1.0000

Test Metrics:
Accuracy_test: 0.9938
Precision_test: 0.9942
Recall_test: 0.9925
F1_score_test: 0.9933
F2_score_test: 0.9928
Roc_auc_test: 0.9988
Pr_auc_test: 0.9989

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9951

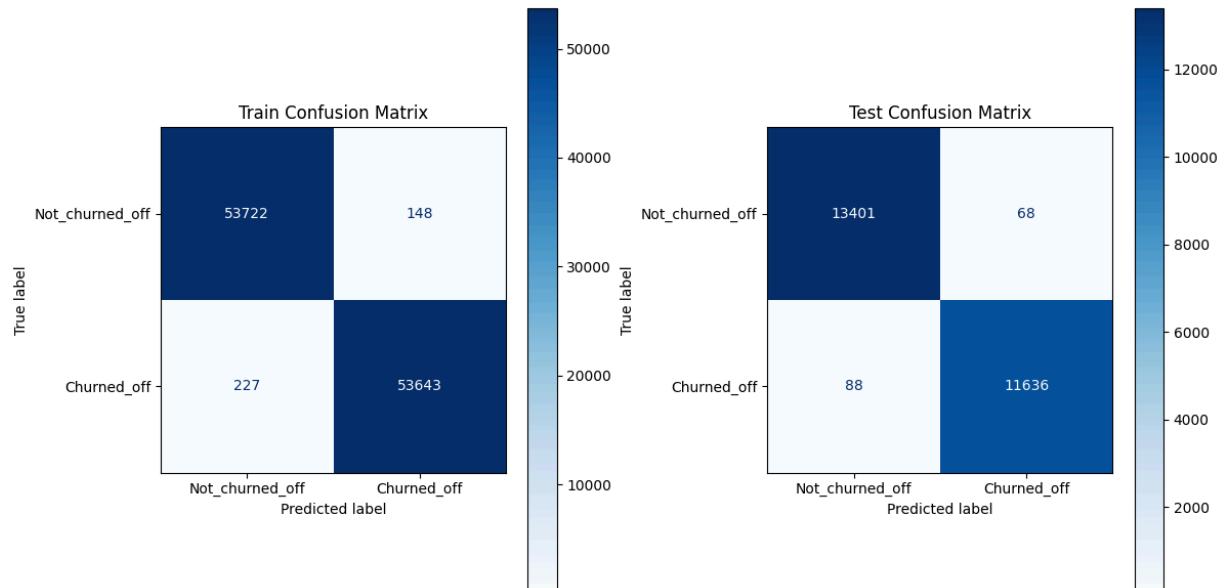
```

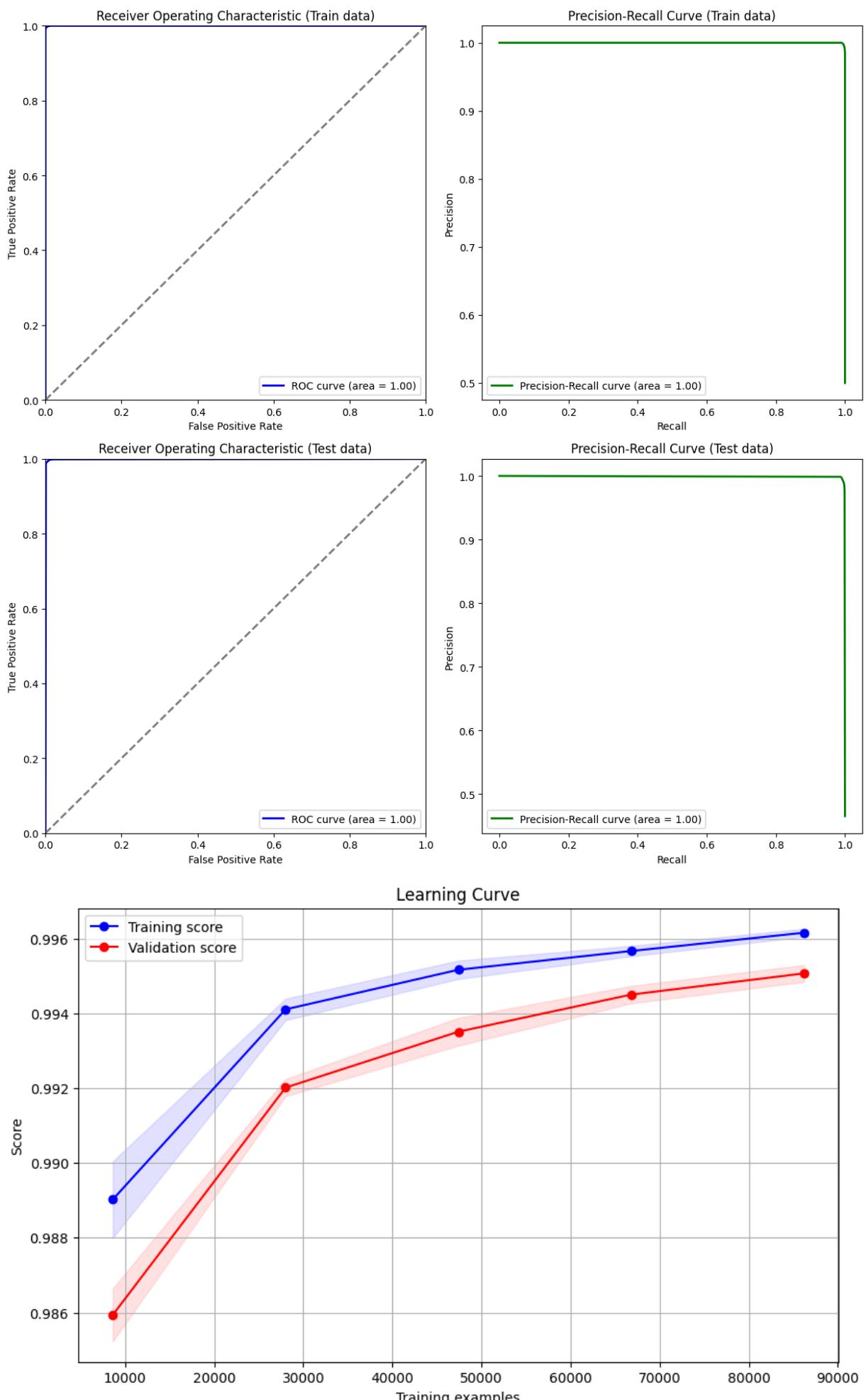
#### Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	53870
accuracy			1.00	107740
macro avg	1.00	1.00	1.00	107740
weighted avg	1.00	1.00	1.00	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	13469
1	0.99	0.99	0.99	11724
accuracy			0.99	25193
macro avg	0.99	0.99	0.99	25193
weighted avg	0.99	0.99	0.99	25193





2025/05/16 18:51:37 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging is completed

## DECISION TREE MODEL

Imbalanced Dataset

Hyper parameter Tuning

```
In [ ]: # Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'splitter': ['best', 'random'],
    'max_depth': stats.randint(2, 20), # Random integers for max_depth
    'min_samples_split': stats.randint(2, 20), # Random integers for min_samples_split
    'min_samples_leaf': stats.randint(1, 20), # Random integers for min_samples_leaf
    'min_weight_fraction_leaf': stats.uniform(0.0, 0.5), # Uniform distribution for min_weight_fraction_leaf
    'max_features': [None, 'auto', 'sqrt', 'log2'], # Options for max_features
    'max_leaf_nodes': stats.randint(2, 100), # Random integers for max_leaf_nodes
    'min_impurity_decrease': stats.uniform(0.0, 0.1), # Uniform distribution for min_impurity_decrease
    'ccp_alpha': stats.uniform(0.0, 0.1), # Uniform distribution for ccp_alpha
    'random_state': [42] # Fixed random state for reproducibility
}

# Initialize the Decision Tree classifier
dtc = DecisionTreeClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=dtc,
    param_distributions=param_dist,
    n_iter=200, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)
```

Fitting 5 folds for each of 200 candidates, totalling 1000 fits  
 Best parameters found: {'ccp\_alpha': np.float64(0.03745401188473625), 'criterion': 'gini', 'max\_depth': 16, 'max\_features': 'sqrt', 'max\_leaf\_nodes': 73, 'min\_impurity\_decrease': np.float64(0.05986584841970366), 'min\_samples\_leaf': 7, 'min\_samples\_split': 12, 'min\_weight\_fraction\_leaf': np.float64(0.22962444598293358), 'random\_state': 42, 'splitter': 'best'}  
 Best score: 0.9219288159581385  
 Tuning\_time 57.644357204437256

## Logging Best Support Vector Machine Model into MLFLOW

```
In [ ]: # Model details
name = "Tuned_Decision_Tree_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
```

```

testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes

Model: Tuned_Decimal_Tree_on_Imbalanced_Dataset
params: {'ccp_alpha': np.float64(0.03745401188473625), 'criterion': 'gini', 'max_dept
h': 16, 'max_features': 'sqrt', 'max_leaf_nodes': 73, 'min_impurity_decrease': np.flo
at64(0.05986584841970366), 'min_samples_leaf': 7, 'min_samples_split': 12, 'min_weigh
t_fraction_leaf': np.float64(0.22962444598293358), 'random_state': 42, 'splitter': 'b
est'}
Training Time: 0.1792 seconds
Testing Time: 0.0656 seconds
Tuning Time: 57.6444 seconds
Train Metrics:
Accuracy_train: 0.9219
Precision_train: 0.9193
Recall_train: 0.9124
F1_score_train: 0.9158
F2_score_train: 0.9138
Roc_auc_train: 0.9213
Pr_auc_train: 0.9362

Test Metrics:
Accuracy_test: 0.9232
Precision_test: 0.9184
Recall_test: 0.9163
F1_score_test: 0.9173
F2_score_test: 0.9167
Roc_auc_test: 0.9227
Pr_auc_test: 0.9368

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9219

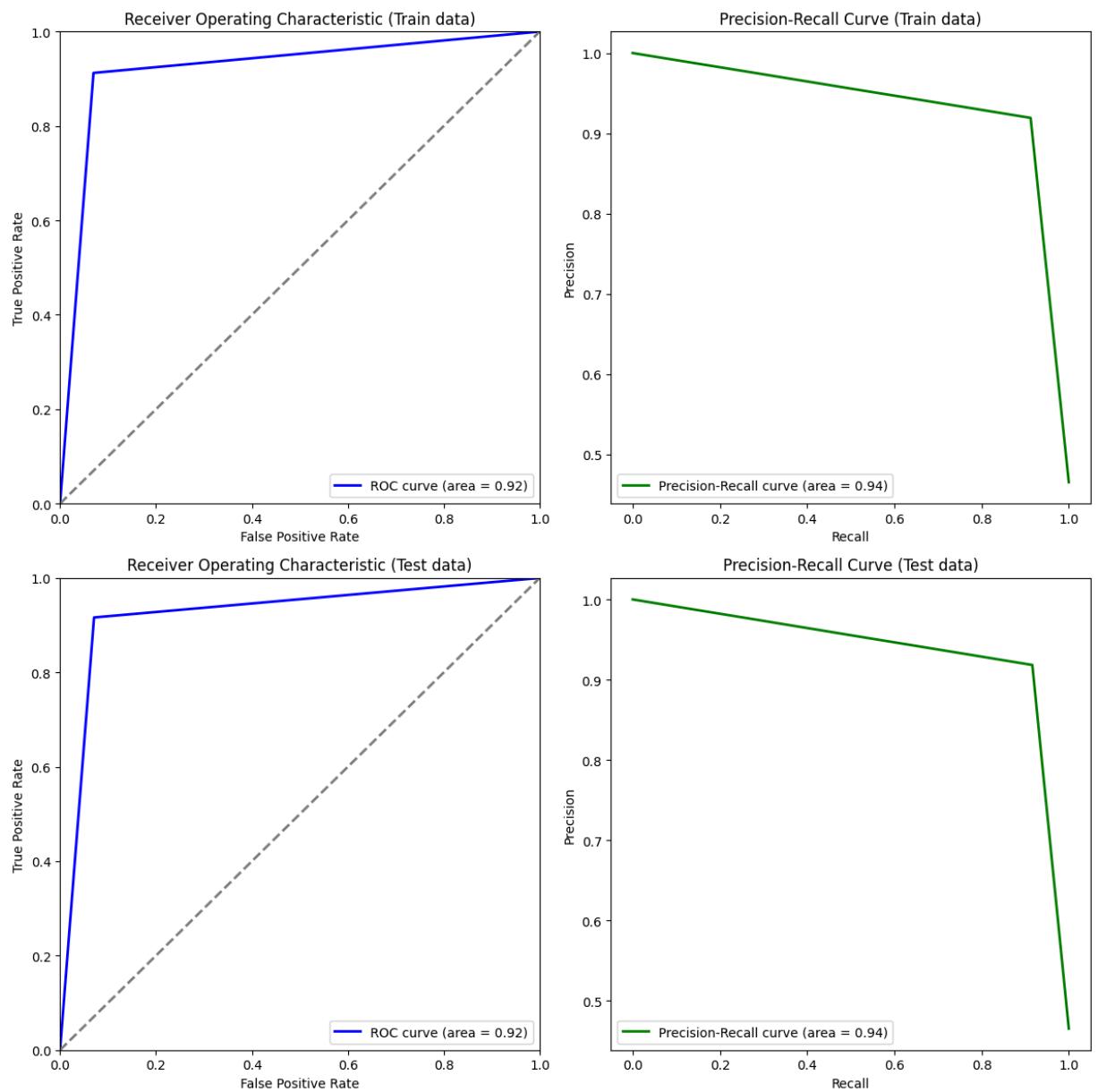
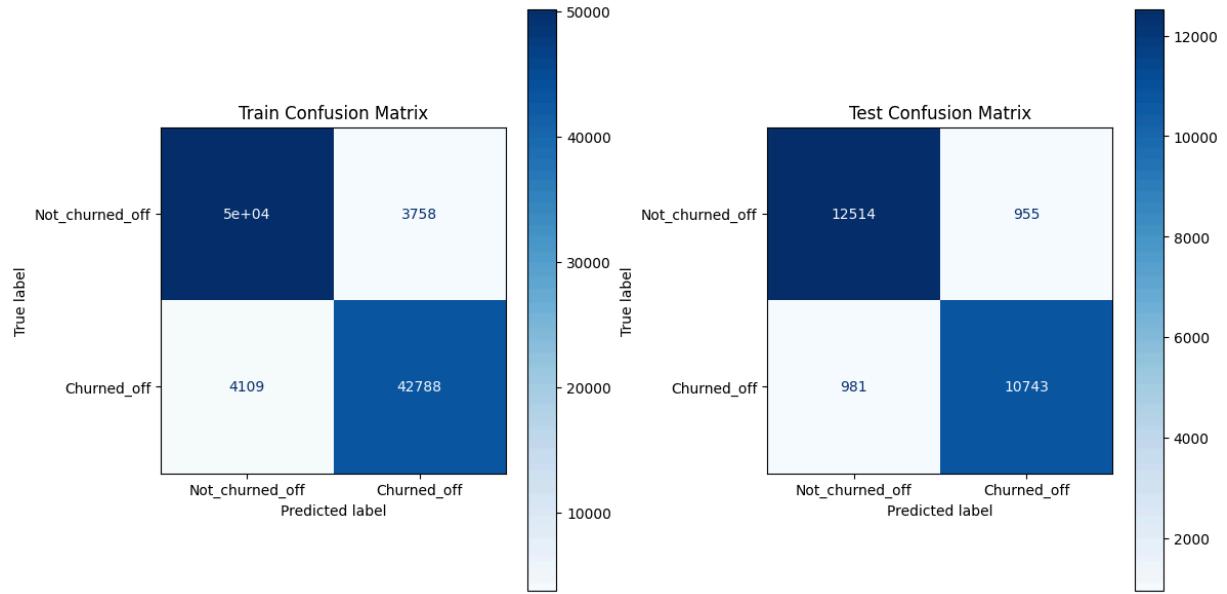
Train Classification Report:
      precision    recall   f1-score   support
          0       0.92      0.93      0.93     53870
          1       0.92      0.91      0.92     46897

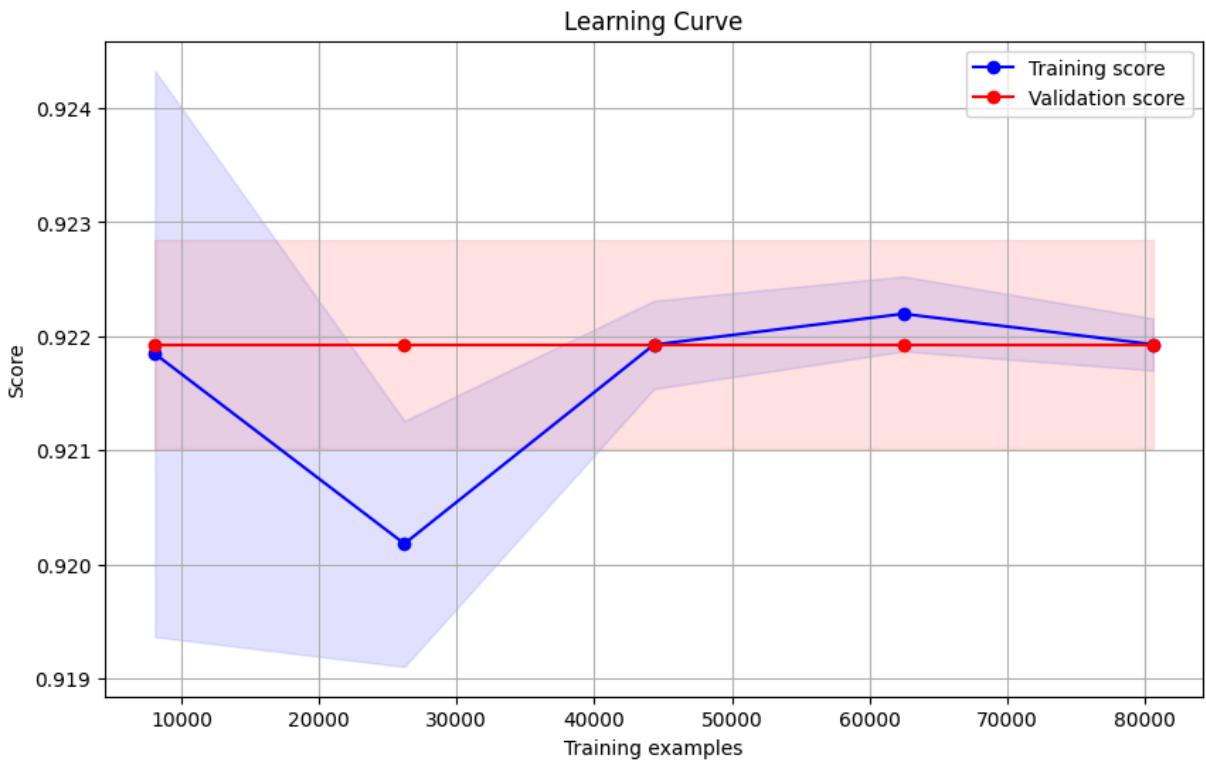
      accuracy                           0.92     100767
      macro avg       0.92      0.92      0.92     100767
  weighted avg       0.92      0.92      0.92     100767

Test Classification Report:
      precision    recall   f1-score   support
          0       0.93      0.93      0.93     13469
          1       0.92      0.92      0.92     11724

      accuracy                           0.92     25193
      macro avg       0.92      0.92      0.92     25193
  weighted avg       0.92      0.92      0.92     25193

```





2025/05/16 18:52:58 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'splitter': ['best', 'random'],
    'max_depth': stats.randint(2, 20), # Random integers for max_depth
    'min_samples_split': stats.randint(2, 20), # Random integers for min_samples_split
    'min_samples_leaf': stats.randint(1, 20), # Random integers for min_samples_leaf
    'min_weight_fraction_leaf': stats.uniform(0.0, 0.5), # Uniform distribution for min_weight_fraction_leaf
    'max_features': [None, 'auto', 'sqrt', 'log2'], # Options for max_features
    'max_leaf_nodes': stats.randint(2, 100), # Random integers for max_leaf_nodes
    'min_impurity_decrease': stats.uniform(0.0, 0.1), # Uniform distribution for min_impurity_decrease
    'ccp_alpha': stats.uniform(0.0, 0.1), # Uniform distribution for ccp_alpha
    'random_state': [42] # Fixed random state for reproducibility
}

# Initialize the Decision Tree classifier
dtc = DecisionTreeClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=dtc,
    param_distributions=param_dist,
    n_iter=200, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
Best parameters found: {'ccp_alpha': np.float64(0.03745401188473625), 'criterion': 'gini', 'max_depth': 16, 'max_features': 'sqrt', 'max_leaf_nodes': 73, 'min_impurity_decrease': np.float64(0.05986584841970366), 'min_samples_leaf': 7, 'min_samples_split': 12, 'min_weight_fraction_leaf': np.float64(0.22962444598293358), 'random_state': 42, 'splitter': 'best'}
Best score: 0.9215147577501392
Tuning_time 386.1706097126007
```

### Logging Best Decision Tree Model into MLFLOW

```
In [ ]: # Model details
name = "Tuned_Decision_Tree_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_te
```

```

Model: Tuned_Decision_Tree_on_Balanced_Dataset
params: {'ccp_alpha': np.float64(0.03745401188473625), 'criterion': 'gini', 'max_depth': 16, 'max_features': 'sqrt', 'max_leaf_nodes': 73, 'min_impurity_decrease': np.float64(0.05986584841970366), 'min_samples_leaf': 7, 'min_samples_split': 12, 'min_weight_fraction_leaf': np.float64(0.22962444598293358), 'random_state': 42, 'splitter': 'best'}
Training Time: 0.2437 seconds
Testing Time: 0.1159 seconds
Tuning Time: 386.1706 seconds
Train Metrics:
Accuracy_train: 0.9215
Precision_train: 0.9290
Recall_train: 0.9128
F1_score_train: 0.9208
F2_score_train: 0.9160
Roc_auc_train: 0.9215
Pr_auc_train: 0.9427

Test Metrics:
Accuracy_test: 0.9232
Precision_test: 0.9184
Recall_test: 0.9163
F1_score_test: 0.9173
F2_score_test: 0.9167
Roc_auc_test: 0.9227
Pr_auc_test: 0.9368

```

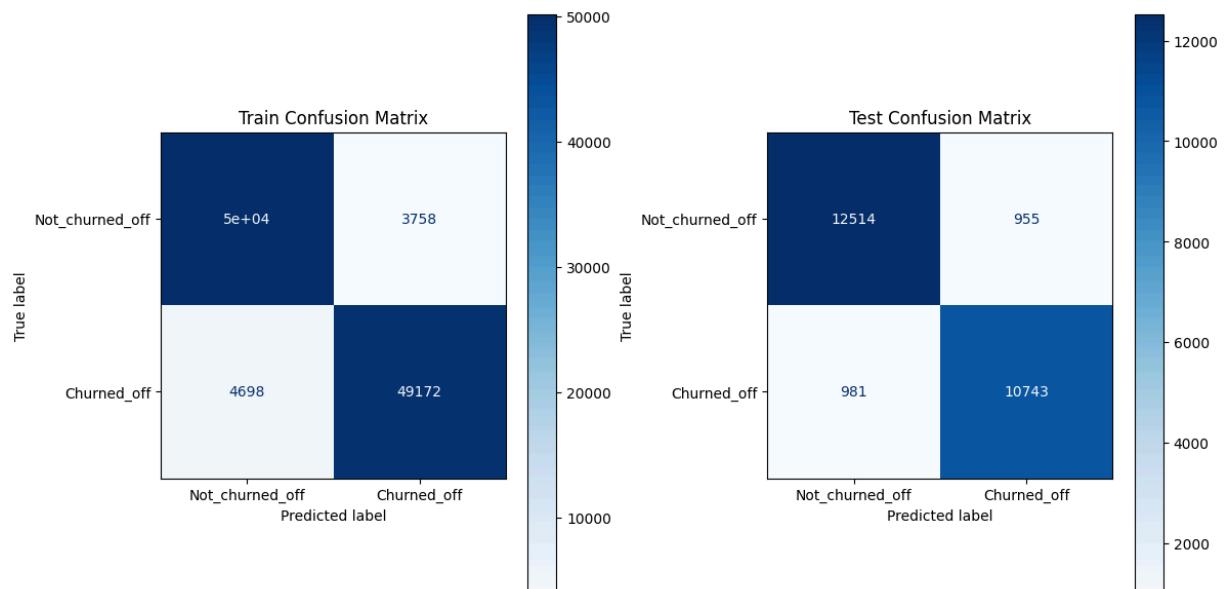
Tuning Metrics:  
hyper\_parameter\_tuning\_best\_est\_score: 0.9215

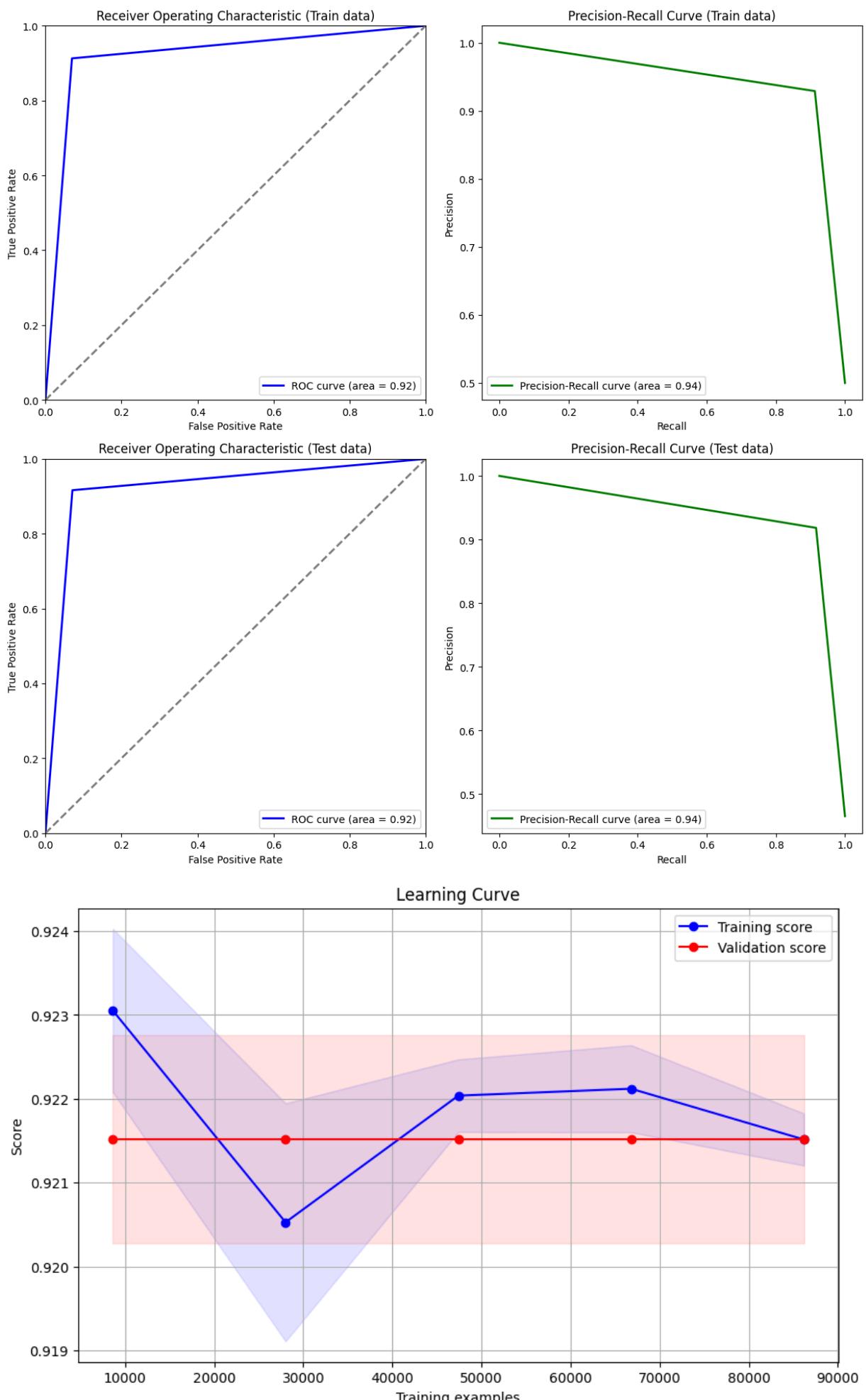
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.91	0.93	0.92	53870
1	0.93	0.91	0.92	53870
accuracy			0.92	107740
macro avg	0.92	0.92	0.92	107740
weighted avg	0.92	0.92	0.92	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	0.93	0.93	0.93	13469
1	0.92	0.92	0.92	11724
accuracy			0.92	25193
macro avg	0.92	0.92	0.92	25193
weighted avg	0.92	0.92	0.92	25193





```
2025/05/16 18:59:54 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## RANDOM FOREST MODEL

Imbalanced Dataset

Hyper parameter Tuning

```
In [ ]: # Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'n_estimators': stats.randint(50, 100), # Random integers for n_estimators
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': stats.randint(2, 20), # Random integers for max_depth
    'min_samples_split': stats.randint(2, 20), # Random integers for min_samples_split
    'min_samples_leaf': stats.randint(1, 20), # Random integers for min_samples_leaf
    'min_weight_fraction_leaf': stats.uniform(0.0, 0.5), # Uniform distribution for min_weight_fraction_leaf
    'max_features': ['auto', 'sqrt', 'log2', None], # Options for max_features
    'max_leaf_nodes': stats.randint(2, 100), # Random integers for max_leaf_nodes
    'min_impurity_decrease': stats.uniform(0.0, 0.1), # Uniform distribution for min_impurity_decrease
    'bootstrap': [True, False], # Whether bootstrap samples are used when building trees
    'oob_score': [True, False], # Whether to use out-of-bag samples to estimate the OOB error
    'ccp_alpha': stats.uniform(0.0, 0.1), # Uniform distribution for ccp_alpha
    'random_state': [42] # Fixed random state for reproducibility
}

# Initialize the RandomForest classifier
rfc = RandomForestClassifier(class_weight="balanced")

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=rfc,
    param_distributions=param_dist,
    n_iter=20, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)

Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best parameters found: {'bootstrap': False, 'ccp_alpha': np.float64(0.04477831645730917), 'criterion': 'entropy', 'max_depth': 5, 'max_features': 'log2', 'max_leaf_node': 17, 'min_impurity_decrease': np.float64(0.03696544560614045), 'min_samples_leaf': 17, 'min_samples_split': 7, 'min_weight_fraction_leaf': np.float64(0.1838915663596266), 'n_estimators': 83, 'oob_score': False, 'random_state': 42}
Best score: 0.9296992057598708
Tuning_time 271.72219586372375
```

```
In [ ]: rfc_imb = random_search.best_estimator_
```

## Logging Best Random Forest Model into MLFLOW

```
In [ ]: # Model details
name = "Tuned_Random_Forest_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
```

```

y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes

```

Model: Tuned\_Random\_Forest\_on\_Imbalanced\_Dataset  
 params: {'bootstrap': False, 'ccp\_alpha': np.float64(0.04477831645730917), 'criterion': 'entropy', 'max\_depth': 5, 'max\_features': 'log2', 'max\_leaf\_nodes': 76, 'min\_impurity\_decrease': np.float64(0.03696544560614045), 'min\_samples\_leaf': 17, 'min\_samples\_split': 7, 'min\_weight\_fraction\_leaf': np.float64(0.1838915663596266), 'n\_estimators': 83, 'oob\_score': False, 'random\_state': 42}  
 Training Time: 8.1180 seconds  
 Testing Time: 0.6067 seconds  
 Tuning Time: 271.7222 seconds  
 Train Metrics:  
 Accuracy\_train: 0.9297  
 Precision\_train: 0.9394  
 Recall\_train: 0.9074  
 F1\_score\_train: 0.9231  
 F2\_score\_train: 0.9136  
 Roc\_auc\_train: 0.9824  
 Pr\_auc\_train: 0.9815  
 Test Metrics:  
 Accuracy\_test: 0.9245  
 Precision\_test: 0.9212  
 Recall\_test: 0.9161  
 F1\_score\_test: 0.9186  
 F2\_score\_test: 0.9171  
 Roc\_auc\_test: 0.9829  
 Pr\_auc\_test: 0.9820

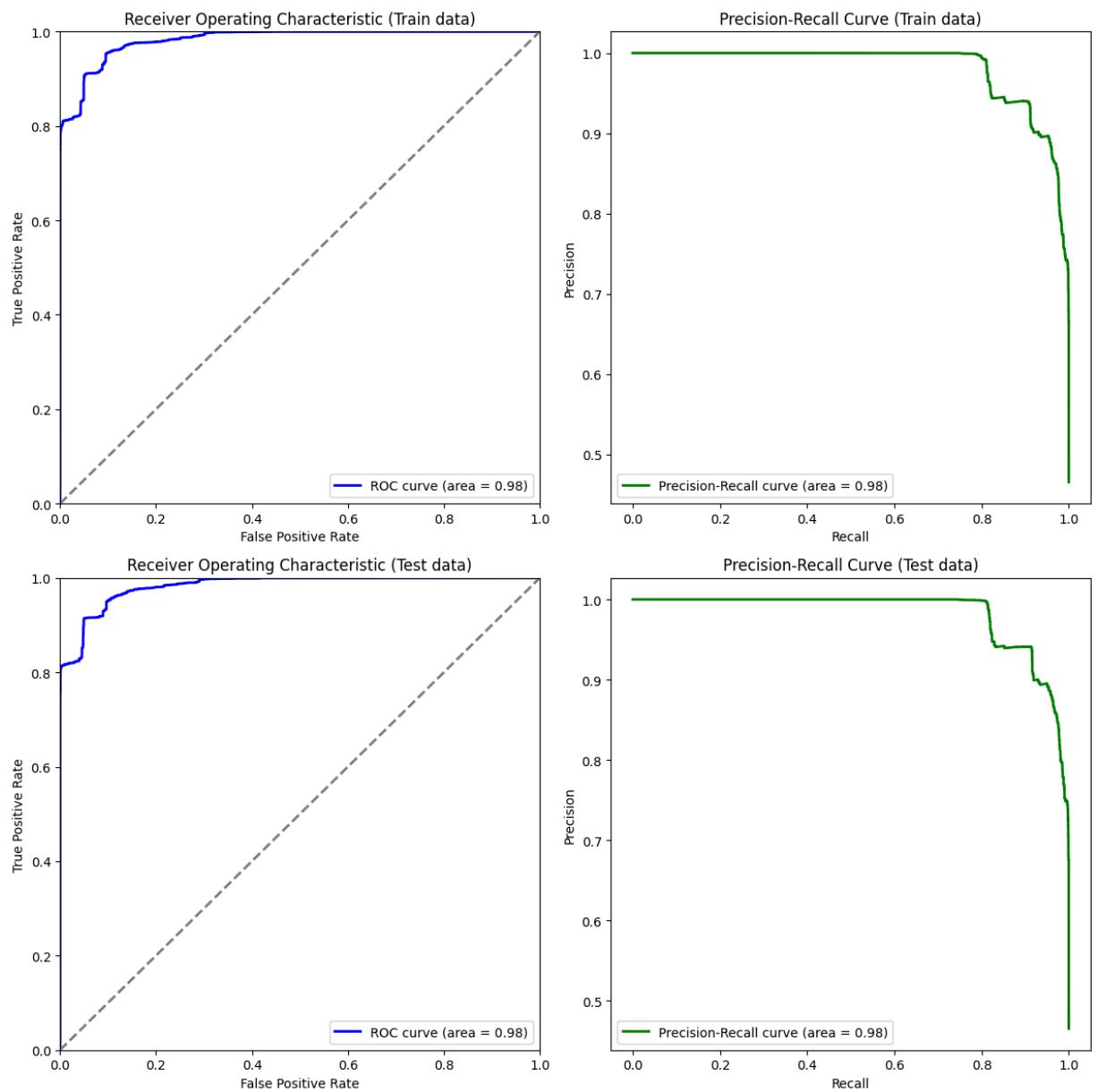
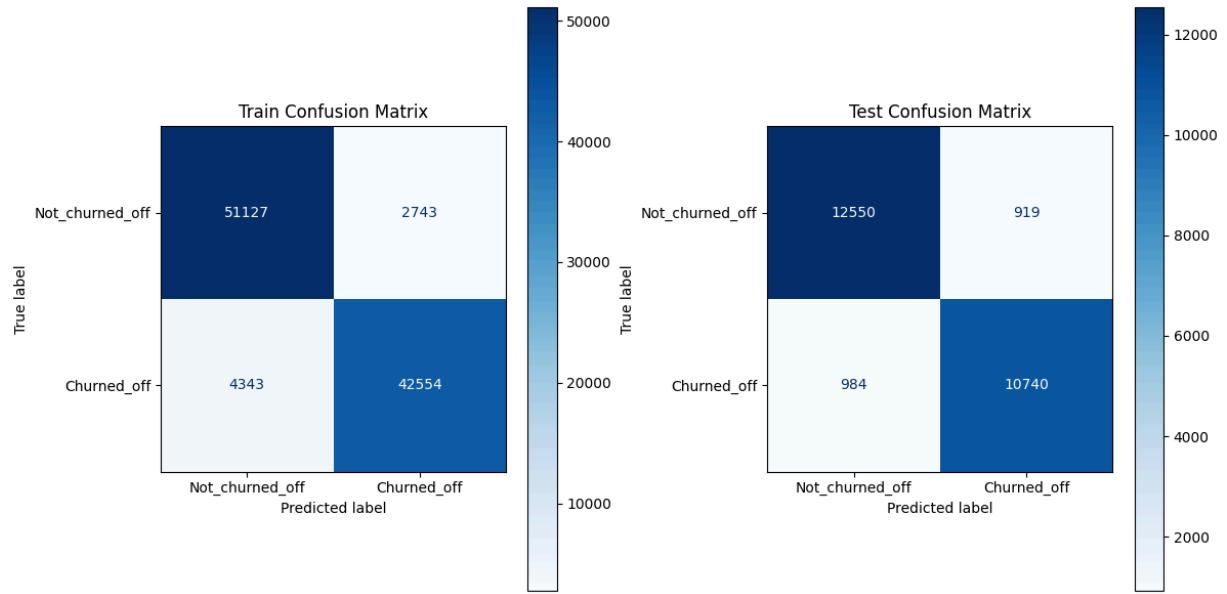
Tuning Metrics:  
 hyper\_parameter\_tuning\_best\_est\_score: 0.9297

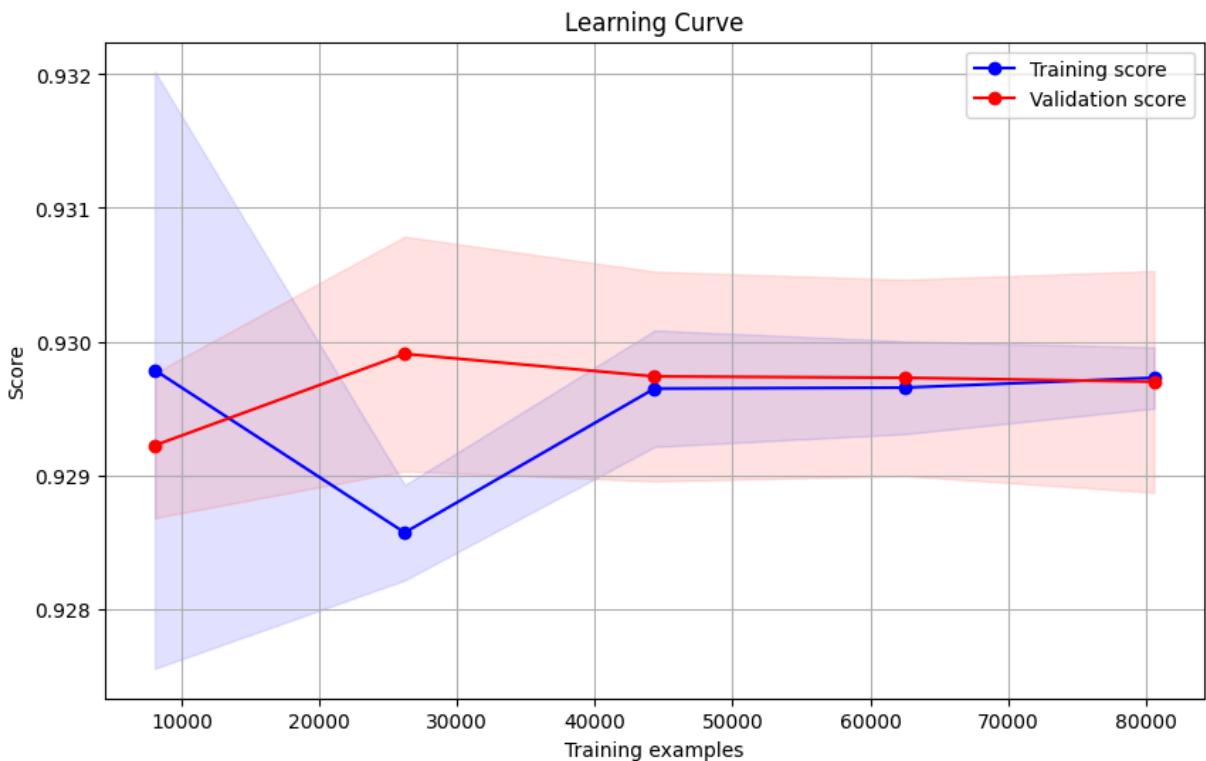
Train Classification Report:

	precision	recall	f1-score	support
0	0.92	0.95	0.94	53870
1	0.94	0.91	0.92	46897
accuracy			0.93	100767
macro avg	0.93	0.93	0.93	100767
weighted avg	0.93	0.93	0.93	100767

Test Classification Report:

	precision	recall	f1-score	support
0	0.93	0.93	0.93	13469
1	0.92	0.92	0.92	11724
accuracy			0.92	25193
macro avg	0.92	0.92	0.92	25193
weighted avg	0.92	0.92	0.92	25193





2025/05/16 19:05:24 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'n_estimators': stats.randint(50, 100), # Random integers for n_estimators
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': stats.randint(2, 20), # Random integers for max_depth
    'min_samples_split': stats.randint(2, 20), # Random integers for min_samples_split
    'min_samples_leaf': stats.randint(1, 20), # Random integers for min_samples_leaf
    'min_weight_fraction_leaf': stats.uniform(0.0, 0.5), # Uniform distribution for min_weight_fraction_leaf
    'max_features': ['auto', 'sqrt', 'log2', None], # Options for max_features
    'max_leaf_nodes': stats.randint(2, 100), # Random integers for max_leaf_nodes
    'min_impurity_decrease': stats.uniform(0.0, 0.1), # Uniform distribution for min_impurity_decrease
    'bootstrap': [True, False], # Whether bootstrap samples are used when building
    'oob_score': [True, False], # Whether to use out-of-bag samples to estimate the oob_score
    'ccp_alpha': stats.uniform(0.0, 0.1), # Uniform distribution for ccp_alpha
    'random_state': [42] # Fixed random state for reproducibility
}

# Initialize the RandomForest classifier
rfc = RandomForestClassifier(class_weight="balanced")

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=rfc,
    param_distributions=param_dist,
    n_iter=20, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
```

```
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best parameters found: {'bootstrap': False, 'ccp_alpha': np.float64(0.044778316457309
17), 'criterion': 'entropy', 'max_depth': 5, 'max_features': 'log2', 'max_leaf_node
s': 76, 'min_impurity_decrease': np.float64(0.03696544560614045), 'min_samples_leaf': 17, 'min_samples_split': 7, 'min_weight_fraction_leaf': np.float64(0.183891566359626
6), 'n_estimators': 83, 'oob_score': False, 'random_state': 42}
Best score: 0.9284573974382774
Tuning_time 301.9873044490814
```

```
In [ ]: rfc_bal = random_search.best_estimator_
```

## Logging Best Random Forest Model into MLFLOW

```
# Model details
name = "Tuned_Random_Forest_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_te
```

```

Model: Tuned_Random_Forest_on_Balanced_Dataset
params: {'bootstrap': False, 'ccp_alpha': np.float64(0.04477831645730917), 'criterion': 'entropy', 'max_depth': 5, 'max_features': 'log2', 'max_leaf_nodes': 76, 'min_impurity_decrease': np.float64(0.03696544560614045), 'min_samples_leaf': 17, 'min_samples_split': 7, 'min_weight_fraction_leaf': np.float64(0.1838915663596266), 'n_estimators': 83, 'oob_score': False, 'random_state': 42}
Training Time: 8.5680 seconds
Testing Time: 0.6366 seconds
Tuning Time: 301.9873 seconds
Train Metrics:
Accuracy_train: 0.9285
Precision_train: 0.9467
Recall_train: 0.9082
F1_score_train: 0.9270
F2_score_train: 0.9156
Roc_auc_train: 0.9824
Pr_auc_train: 0.9836

Test Metrics:
Accuracy_test: 0.9239
Precision_test: 0.9201
Recall_test: 0.9160
F1_score_test: 0.9180
F2_score_test: 0.9168
Roc_auc_test: 0.9828
Pr_auc_test: 0.9820

```

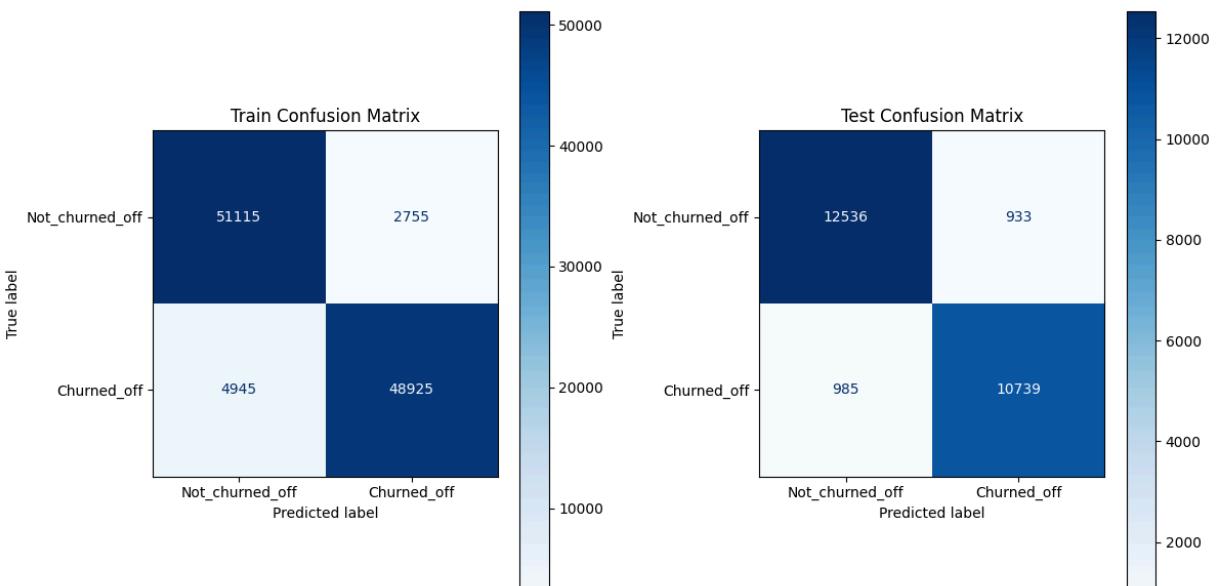
Tuning Metrics:  
hyper\_parameter\_tuning\_best\_est\_score: 0.9285

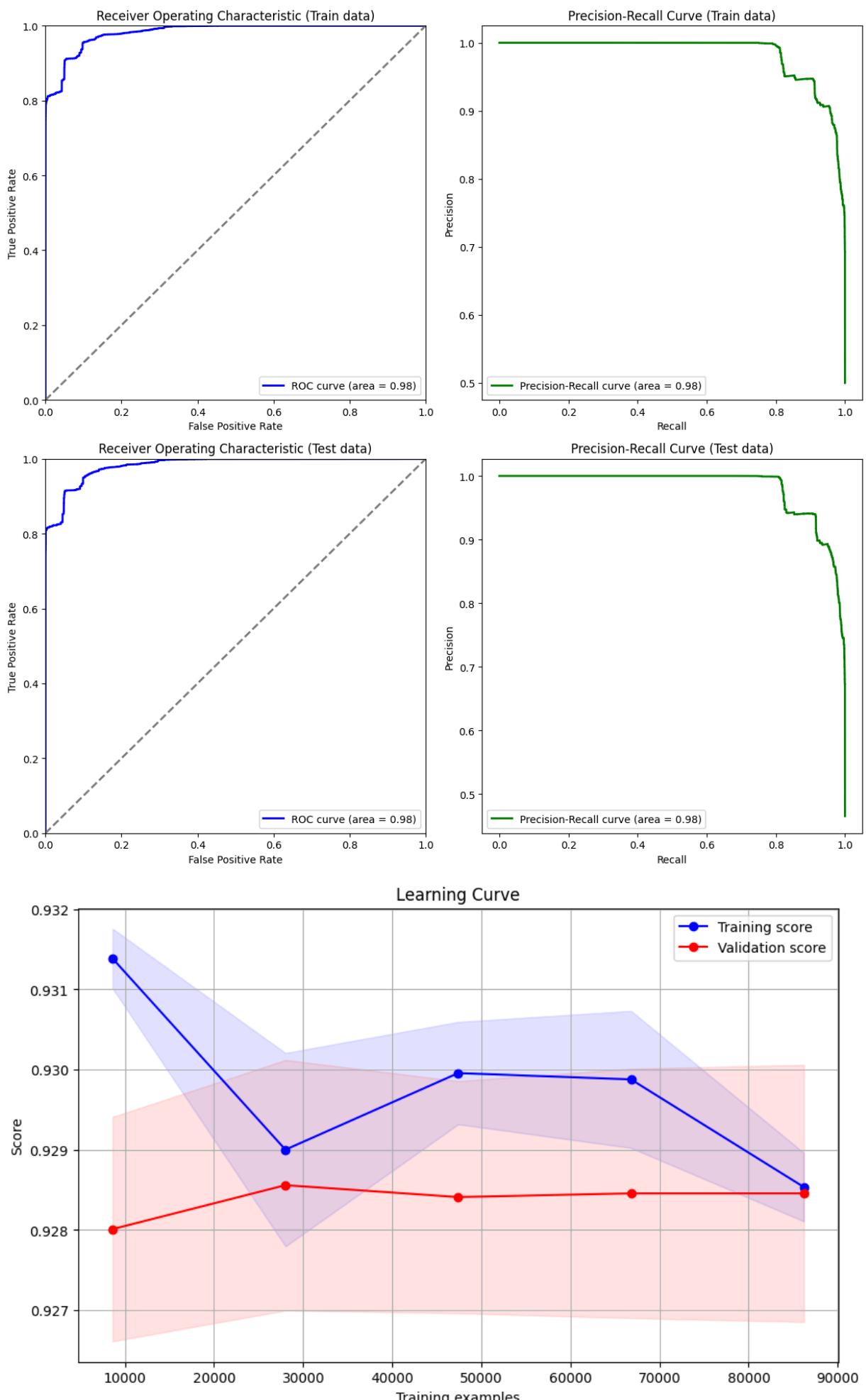
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.91	0.95	0.93	53870
1	0.95	0.91	0.93	53870
accuracy			0.93	107740
macro avg	0.93	0.93	0.93	107740
weighted avg	0.93	0.93	0.93	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	0.93	0.93	0.93	13469
1	0.92	0.92	0.92	11724
accuracy			0.92	25193
macro avg	0.92	0.92	0.92	25193
weighted avg	0.92	0.92	0.92	25193





```
2025/05/16 19:11:31 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## BAGGING CLASSIFIER ON BEST\_RF MODEL

Imbalanced Dataset

Hyper parameter Tuning

In [ ]: rfc\_imb

```

Out[ ]: RandomForestClassifier
RandomForestClassifier(bootstrap=False,
                      ccp_alpha=np.float64(0.04477831645730917),
                      class_weight='balanced', criterion='entropy',
                      max_depth=5, max_features='log2', max_leaf_nodes=7
6,
                      min_impurity_decrease=np.float64(0.036965445606140
45),
                      min_samples_leaf=17, min_samples_split=7,
                      min_weight_fraction_leaf=np.float64(0.183891566359 ▾

```

```

In [ ]: # Base RandomForest model with the provided parameters
base_rf = RandomForestClassifier(class_weight='balanced', criterion='log_loss', max_d
min_samples_leaf=10, min_samples_split=13, n_estimators=80,
min_weight_fraction_leaf=np.float64(0.12), random_state=42)

# Define the parameter grid for Bagging
start_tune_time = time.time()
param_dist = {
    'n_estimators': stats.randint(10, 20), # Number of base estimators in the ensemble
    'max_samples': stats.uniform(0.1, 1.0), # Fraction of samples to draw from X to
    'max_features': stats.uniform(0.1, 1.0), # Fraction of features to draw from X
    'bootstrap': [True, False], # Whether samples are drawn with replacement
    'bootstrap_features': [True, False], # Whether features are drawn with replacement
    'random_state': [42] # Fixed random state for reproducibility
}

# Initialize the Bagging classifier with the RandomForest as the base estimator
bagging_clf = BaggingClassifier(base_estimator=base_rf, n_jobs=-1)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=bagging_clf,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=5,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits  
Best parameters found: {'bootstrap': True, 'bootstrap\_features': False, 'max\_features': np.float64(0.7116531604882809), 'max\_samples': np.float64(0.10706630521971741), 'n\_estimators': 18, 'random\_state': 42}  
Best score: 0.9310389320448198  
Tuning\_time 411.7904386520386

### Logging Best Bagging RF Model into MLFLOW

```

In [ ]: # Model details
name = "Tuned_Bagging_RF_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()

```

```
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes
```

```

Model: Tuned_Bagging_RF_on_Imbalanced_Dataset
params: {'bootstrap': True, 'bootstrap_features': False, 'max_features': np.float64(0.7116531604882809), 'max_samples': np.float64(0.10706630521971741), 'n_estimators': 18, 'random_state': 42}
Training Time: 11.5620 seconds
Testing Time: 4.3423 seconds
Tuning Time: 411.7904 seconds
Train Metrics:
Accuracy_train: 0.9310
Precision_train: 0.9399
Recall_train: 0.9100
F1_score_train: 0.9247
F2_score_train: 0.9158
Roc_auc_train: 0.9863
Pr_auc_train: 0.9856

Test Metrics:
Accuracy_test: 0.9400
Precision_test: 0.9253
Recall_test: 0.9475
F1_score_test: 0.9363
F2_score_test: 0.9430
Roc_auc_test: 0.9868
Pr_auc_test: 0.9863

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9310

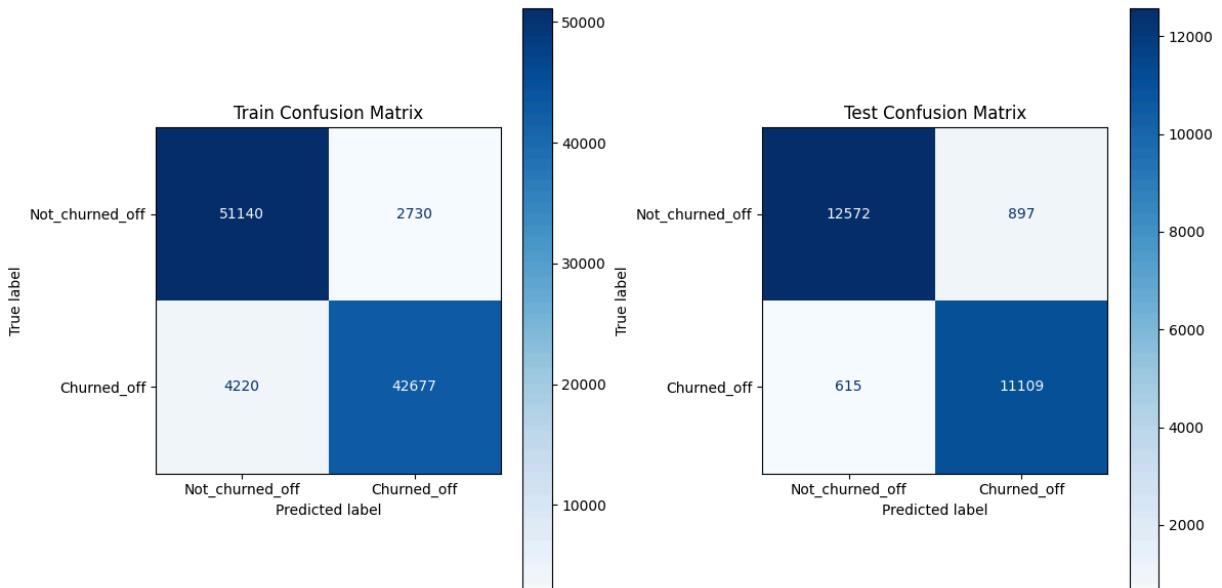
```

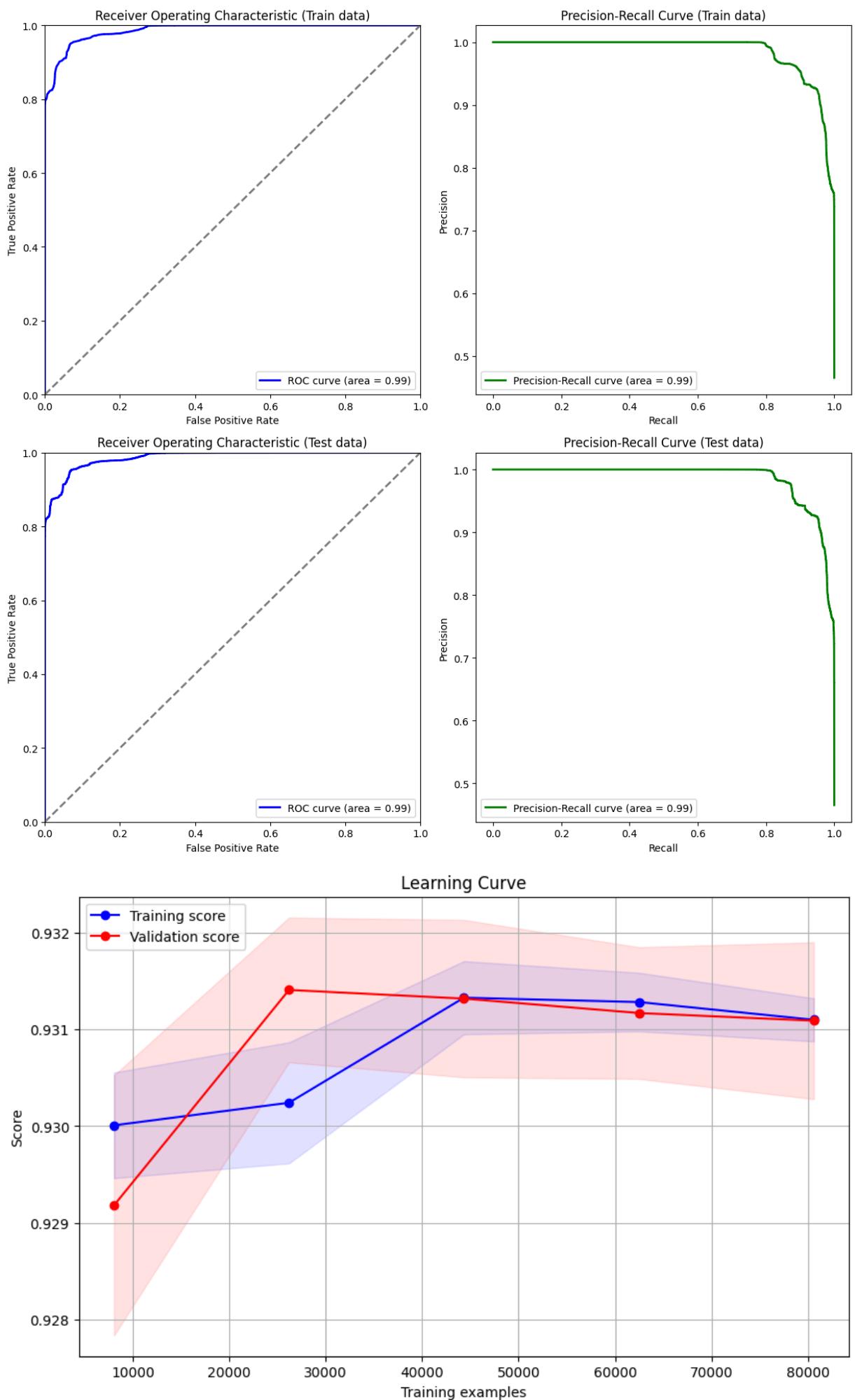
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.92	0.95	0.94	53870
1	0.94	0.91	0.92	46897
accuracy			0.93	100767
macro avg	0.93	0.93	0.93	100767
weighted avg	0.93	0.93	0.93	100767

#### Test Classification Report:

	precision	recall	f1-score	support
0	0.95	0.93	0.94	13469
1	0.93	0.95	0.94	11724
accuracy			0.94	25193
macro avg	0.94	0.94	0.94	25193
weighted avg	0.94	0.94	0.94	25193





```
2025/05/16 19:21:50 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

In [ ]: rfc\_bal

Out[ ]:

```
RandomForestClassifier(bootstrap=False,
                      ccp_alpha=np.float64(0.04477831645730917),
                      class_weight='balanced', criterion='entropy',
                      max_depth=5, max_features='log2', max_leaf_nodes=7
                      6,
                      min_impurity_decrease=np.float64(0.036965445606140
                      45),
                      min_samples_leaf=17, min_samples_split=7,
                      min_weight_fraction_leaf=np.float64(0.183891566359
```

In [ ]:

```
# Base RandomForest model with the provided parameters
base_rf = RandomForestClassifier(class_weight='balanced', criterion='log_loss', max_d
                                min_samples_leaf=10, min_samples_split=13, n_estimators=80,
                                min_weight_fraction_leaf=np.float64(0.12), random_state=42)

# Define the parameter grid for Bagging
start_tune_time = time.time()
param_dist = {
    'n_estimators': stats.randint(10, 20), # Number of base estimators in the ensemble
    'max_samples': stats.uniform(0.1, 1.0), # Fraction of samples to draw from X to
    'max_features': stats.uniform(0.1, 1.0), # Fraction of features to draw from X
    'bootstrap': [True, False], # Whether samples are drawn with replacement
    'bootstrap_features': [True, False], # Whether features are drawn with replacement
    'random_state': [42] # Fixed random state for reproducibility
}

# Initialize the Bagging classifier with the RandomForest as the base estimator
bagging_clf = BaggingClassifier(base_estimator=base_rf)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=bagging_clf,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time", tuning_time)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits  
 Best parameters found: {'bootstrap': True, 'bootstrap\_features': False, 'max\_features': np.float64(0.7116531604882809), 'max\_samples': np.float64(0.10706630521971741), 'n\_estimators': 18, 'random\_state': 42}  
 Best score: 0.9299888620753667  
 Tuning\_time 483.04098987579346

### Logging Best Bagging RF Model into MLFLOW

In [ ]:

```
# Model details
name = "Tuned_Bagging_RF_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
```

```
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_tes
```

```

Model: Tuned_Bagging_RF_on_Balanced_Dataset
params: {'bootstrap': True, 'bootstrap_features': False, 'max_features': np.float64(0.7116531604882809), 'max_samples': np.float64(0.10706630521971741), 'n_estimators': 18, 'random_state': 42}
Training Time: 41.9877 seconds
Testing Time: 16.1428 seconds
Tuning Time: 483.0410 seconds
Train Metrics:
Accuracy_train: 0.9300
Precision_train: 0.9474
Recall_train: 0.9104
F1_score_train: 0.9286
F2_score_train: 0.9176
Roc_auc_train: 0.9864
Pr_auc_train: 0.9874

Test Metrics:
Accuracy_test: 0.9399
Precision_test: 0.9246
Recall_test: 0.9481
F1_score_test: 0.9362
F2_score_test: 0.9433
Roc_auc_test: 0.9868
Pr_auc_test: 0.9864

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9300

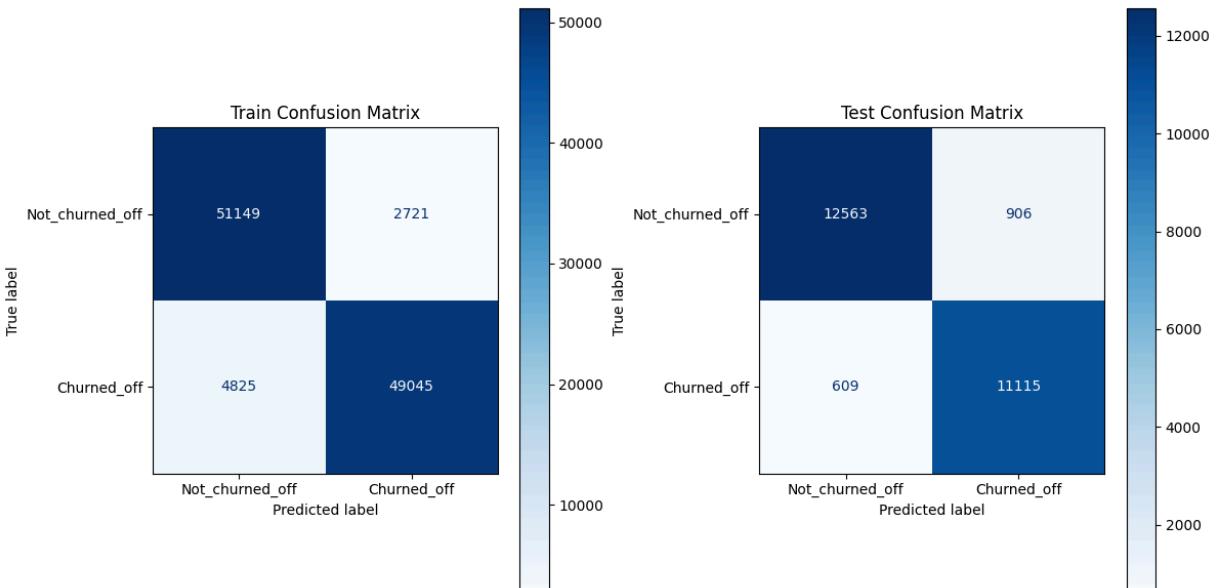
```

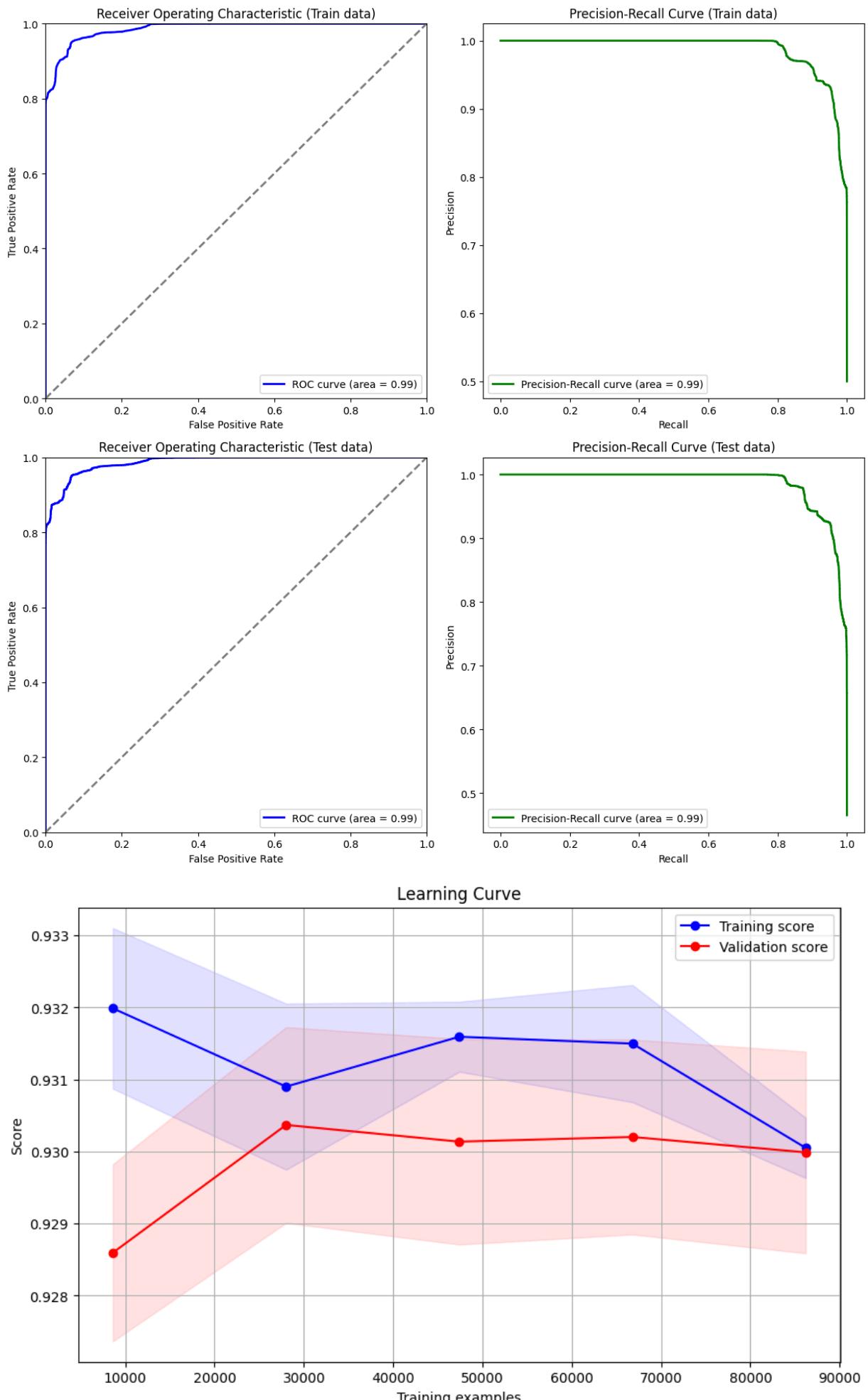
#### Train Classification Report:

	precision	recall	f1-score	support
0	0.91	0.95	0.93	53870
1	0.95	0.91	0.93	53870
accuracy			0.93	107740
macro avg	0.93	0.93	0.93	107740
weighted avg	0.93	0.93	0.93	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	0.95	0.93	0.94	13469
1	0.92	0.95	0.94	11724
accuracy			0.94	25193
macro avg	0.94	0.94	0.94	25193
weighted avg	0.94	0.94	0.94	25193





```
2025/05/16 19:34:49 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## ADABOOST MODEL

Imbalanced Dataset

Hyper parameter Tuning

```
In [ ]: # Define the base estimator (Decision Stump)
base_stump = DecisionTreeClassifier(max_depth=5)

# Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'n_estimators': stats.randint(100, 200), # Number of boosting stages
    'learning_rate': stats.uniform(0.01, 1.0), # Step size for boosting
    'algorithm': ['SAMME', 'SAMME.R'], # Algorithm to use for boosting
}

# Initialize the AdaBoost model with the decision stump as base estimator
ada_boost = AdaBoostClassifier(estimator=base_stump, random_state=42)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=ada_boost,
    param_distributions=param_dist,
    n_iter=2, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)
```

Fitting 5 folds for each of 2 candidates, totalling 10 fits  
 Best parameters found: {'algorithm': 'SAMME', 'learning\_rate': np.float64(0.7896910002727693), 'n\_estimators': 120}  
 Best score: 0.9991366222064008  
 Tuning\_time: 726.1170554161072

## Logging Best Adaboost Model into MLFLOW

```
In [ ]: # Model details
name = "Tuned_Adaboost_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")
```

```
# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_te
```

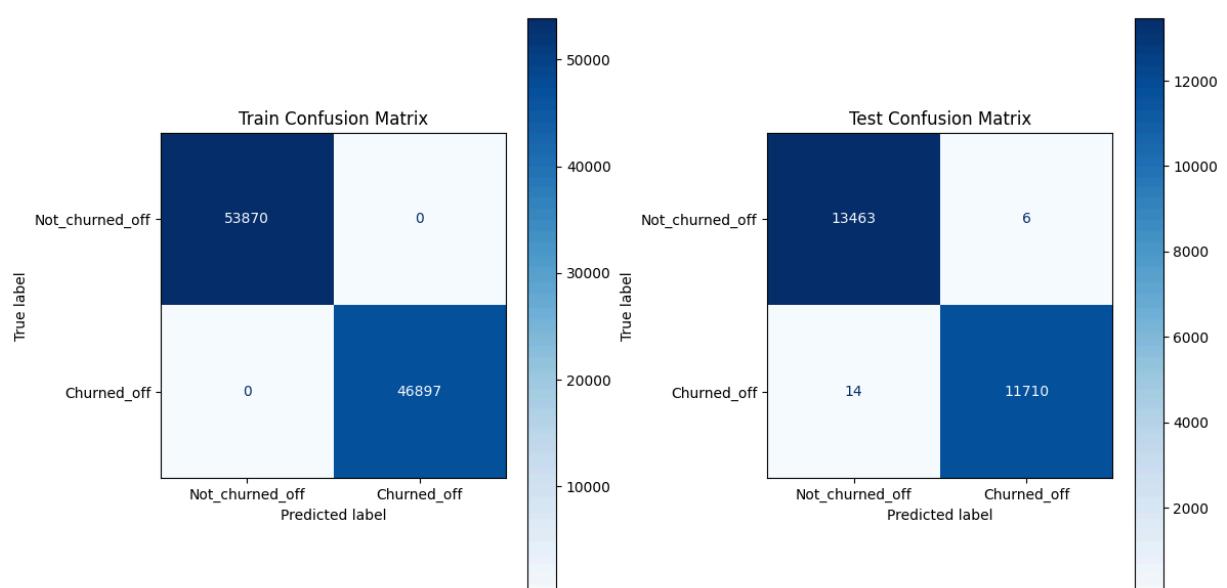
Model: Tuned\_Adaboost\_on\_Imbalanced\_Dataset  
 params: {'algorithm': 'SAMME', 'learning\_rate': np.float64(0.7896910002727693), 'n\_estimators': 120}  
 Training Time: 229.3156 seconds  
 Testing Time: 7.3397 seconds  
 Tuning Time: 726.1171 seconds  
 Train Metrics:  
 Accuracy\_train: 1.0000  
 Precision\_train: 1.0000  
 Recall\_train: 1.0000  
 F1\_score\_train: 1.0000  
 F2\_score\_train: 1.0000  
 Roc\_auc\_train: 1.0000  
 Pr\_auc\_train: 1.0000  
 Test Metrics:  
 Accuracy\_test: 0.9992  
 Precision\_test: 0.9995  
 Recall\_test: 0.9988  
 F1\_score\_test: 0.9991  
 F2\_score\_test: 0.9989  
 Roc\_auc\_test: 1.0000  
 Pr\_auc\_test: 1.0000  
 Tuning Metrics:  
 hyper\_parameter\_tuning\_best\_est\_score: 0.9991

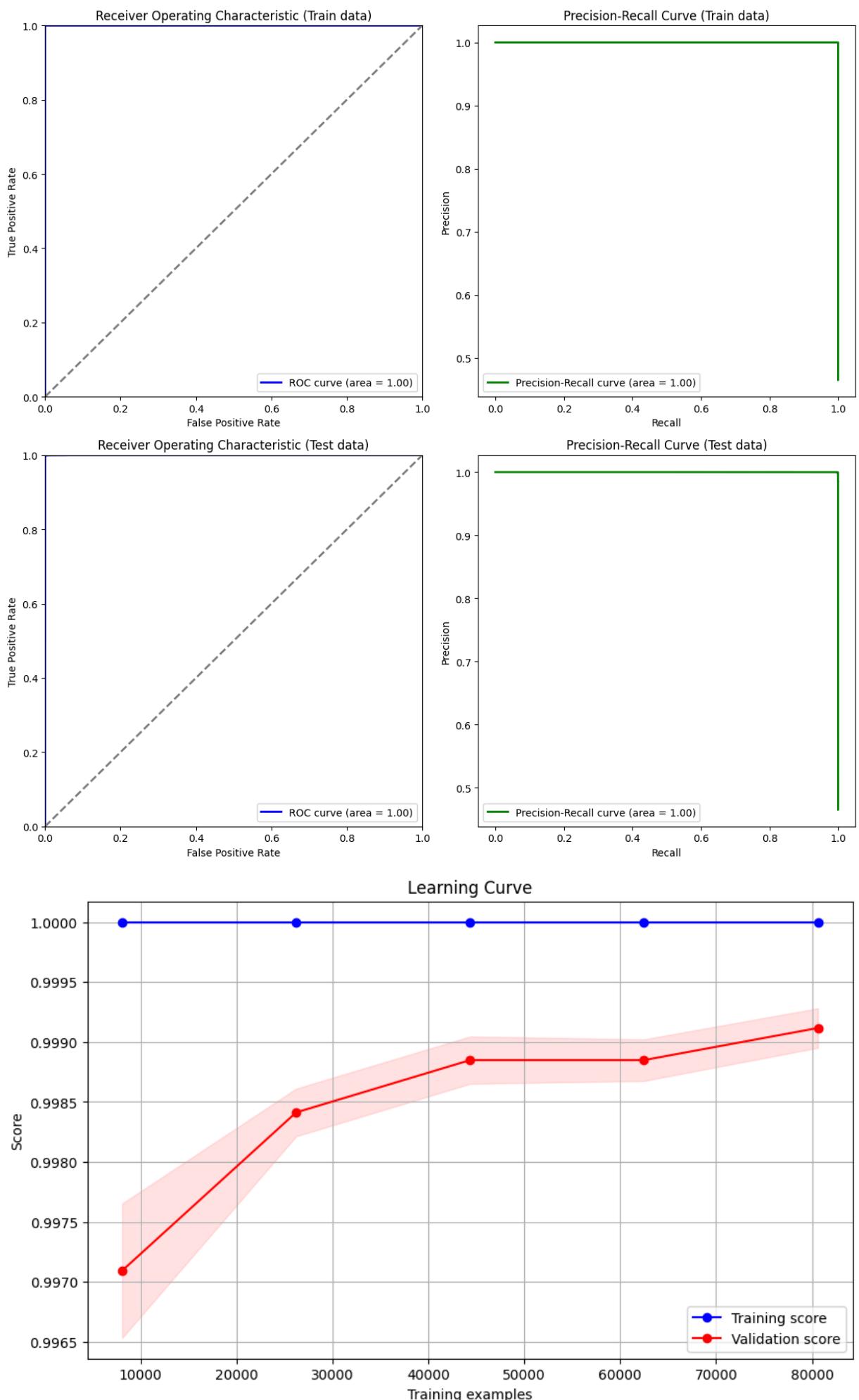
#### Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	46897
accuracy			1.00	100767
macro avg	1.00	1.00	1.00	100767
weighted avg	1.00	1.00	1.00	100767

#### Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	11724
accuracy			1.00	25193
macro avg	1.00	1.00	1.00	25193
weighted avg	1.00	1.00	1.00	25193





```
2025/05/16 20:02:55 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Define the base estimator (Decision Stump)
base_stump = DecisionTreeClassifier(max_depth=5)

# Define the parameter grid
```

```

start_tune_time = time.time()
param_dist = {
    'n_estimators': stats.randint(100, 200), # Number of boosting stages
    'learning_rate': stats.uniform(0.01, 1.0), # Step size for boosting
    'algorithm': ['SAMME', 'SAMME.R'], # Algorithm to use for boosting
}

# Initialize the AdaBoost model with the decision stump as base estimator
ada_boost = AdaBoostClassifier(estimator=base_stump, random_state=42)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=ada_boost,
    param_distributions=param_dist,
    n_iter=2, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)

```

Fitting 5 folds for each of 2 candidates, totalling 10 fits  
Best parameters found: {'algorithm': 'SAMME', 'learning\_rate': np.float64(0.7896910002727693), 'n\_estimators': 120}  
Best score: 0.99910896602933  
Tuning\_time: 791.5874993801117

### Logging Best Adaboost Model into MLFLOW

```

In [ ]: # Model details
name = "Tuned_Adaboost_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_te

```

```

Model: Tuned_Adaboost_on_Balanced_Dataset
params: {'algorithm': 'SAMME', 'learning_rate': np.float64(0.7896910002727693), 'n_estimators': 120}
Training Time: 205.0251 seconds
Testing Time: 6.9199 seconds
Tuning Time: 791.5875 seconds
Train Metrics:
Accuracy_train: 1.0000
Precision_train: 1.0000
Recall_train: 1.0000
F1_score_train: 1.0000
F2_score_train: 1.0000
Roc_auc_train: 1.0000
Pr_auc_train: 1.0000

Test Metrics:
Accuracy_test: 0.9992
Precision_test: 0.9994
Recall_test: 0.9988
F1_score_test: 0.9991
F2_score_test: 0.9989
Roc_auc_test: 1.0000
Pr_auc_test: 1.0000

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9991

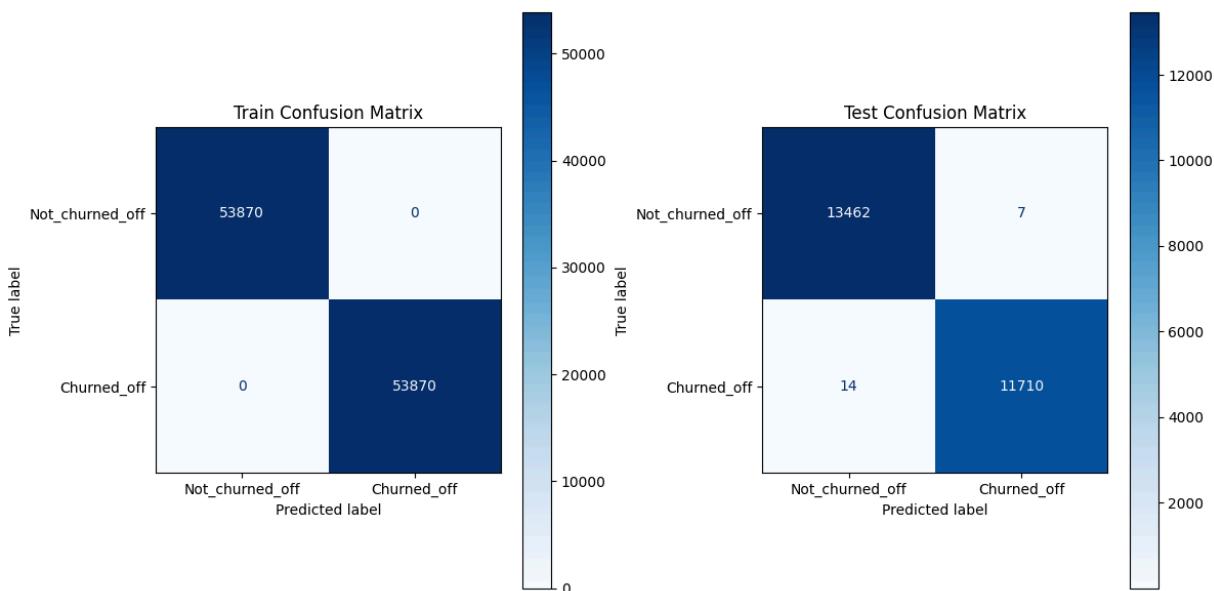
```

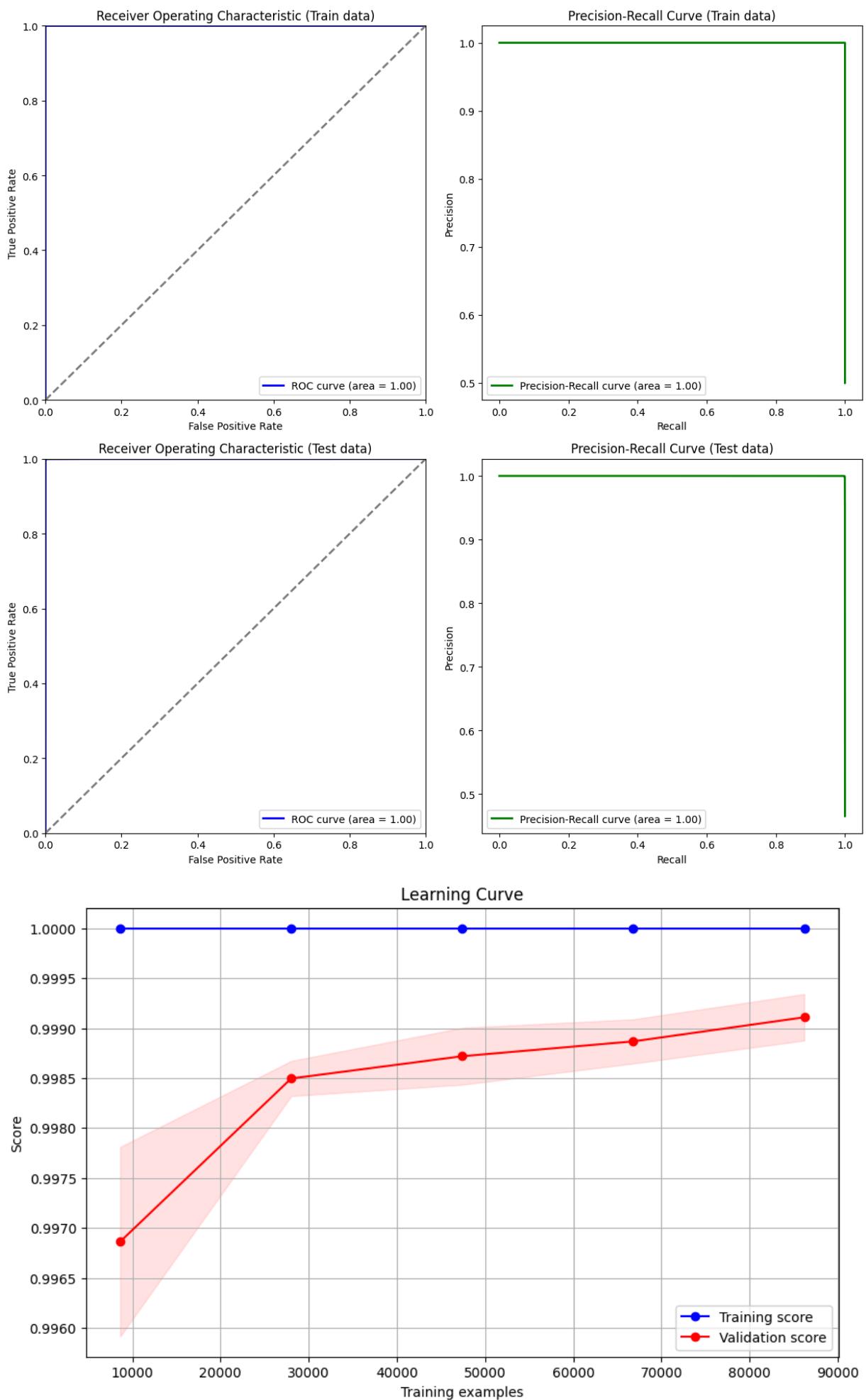
#### Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	53870
accuracy			1.00	107740
macro avg	1.00	1.00	1.00	107740
weighted avg	1.00	1.00	1.00	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	11724
accuracy			1.00	25193
macro avg	1.00	1.00	1.00	25193
weighted avg	1.00	1.00	1.00	25193





```
2025/05/16 20:32:18 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## GRADIENT BOOST MODEL

### Imbalanced Dataset

#### Hyper parameter Tuning

```
In [ ]: # Start timing the tuning process
start_tune_time = time.time()
```

```

# Define the parameter grid
param_dist = {
    'n_estimators': stats.randint(100, 200),
    'learning_rate': stats.uniform(0.01, 0.3),
    'max_depth': stats.randint(3, 15),
    'min_samples_split': stats.randint(2, 20),
    'min_samples_leaf': stats.randint(1, 20),
    'max_features': ['auto', 'sqrt', 'log2', None],
    'subsample': stats.uniform(0.7, 0.3),
    'min_impurity_decrease': stats.uniform(0.0, 0.1),
    'random_state': [42]
}

# Initialize the GradientBoostingClassifier
gbc = GradientBoostingClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=gbc,
    param_distributions=param_dist,
    n_iter=5, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=5,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_

# End timing and print tuning time
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)

```

Fitting 5 folds for each of 5 candidates, totalling 25 fits  
Best parameters found: {'learning\_rate': np.float64(0.19033450352296263), 'max\_depth': 10, 'max\_features': 'log2', 'min\_impurity\_decrease': np.float64(0.0020584494295802446), 'min\_samples\_leaf': 2, 'min\_samples\_split': 13, 'n\_estimators': 129, 'random\_state': 42, 'subsample': np.float64(0.7637017332034828)}  
Best score: 0.998531267558659  
Tuning\_time: 707.4505155086517

## Logging Best Gradient boost Model into MLFLOW

```

In [ ]: # Model details
name = "Tuned_Gradient_boost_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

```

```

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes

```

Model: Tuned\_Gradient\_boost\_on\_Imbalanced\_Dataset  
 params: {'learning\_rate': np.float64(0.19033450352296263), 'max\_depth': 10, 'max\_features': 'log2', 'min\_impurity\_decrease': np.float64(0.0020584494295802446), 'min\_samples\_leaf': 2, 'min\_samples\_split': 13, 'n\_estimators': 129, 'random\_state': 42, 'subsample': np.float64(0.7637017332034828)}

Training Time: 21.5198 seconds

Testing Time: 0.7427 seconds

Tuning Time: 707.4505 seconds

Train Metrics:

Accuracy\_train: 0.9999  
 Precision\_train: 1.0000  
 Recall\_train: 0.9999  
 F1\_score\_train: 0.9999  
 F2\_score\_train: 0.9999  
 Roc\_auc\_train: 1.0000  
 Pr\_auc\_train: 1.0000

Test Metrics:

Accuracy\_test: 0.9983  
 Precision\_test: 0.9993  
 Recall\_test: 0.9971  
 F1\_score\_test: 0.9982  
 F2\_score\_test: 0.9975  
 Roc\_auc\_test: 1.0000  
 Pr\_auc\_test: 1.0000

Tuning Metrics:

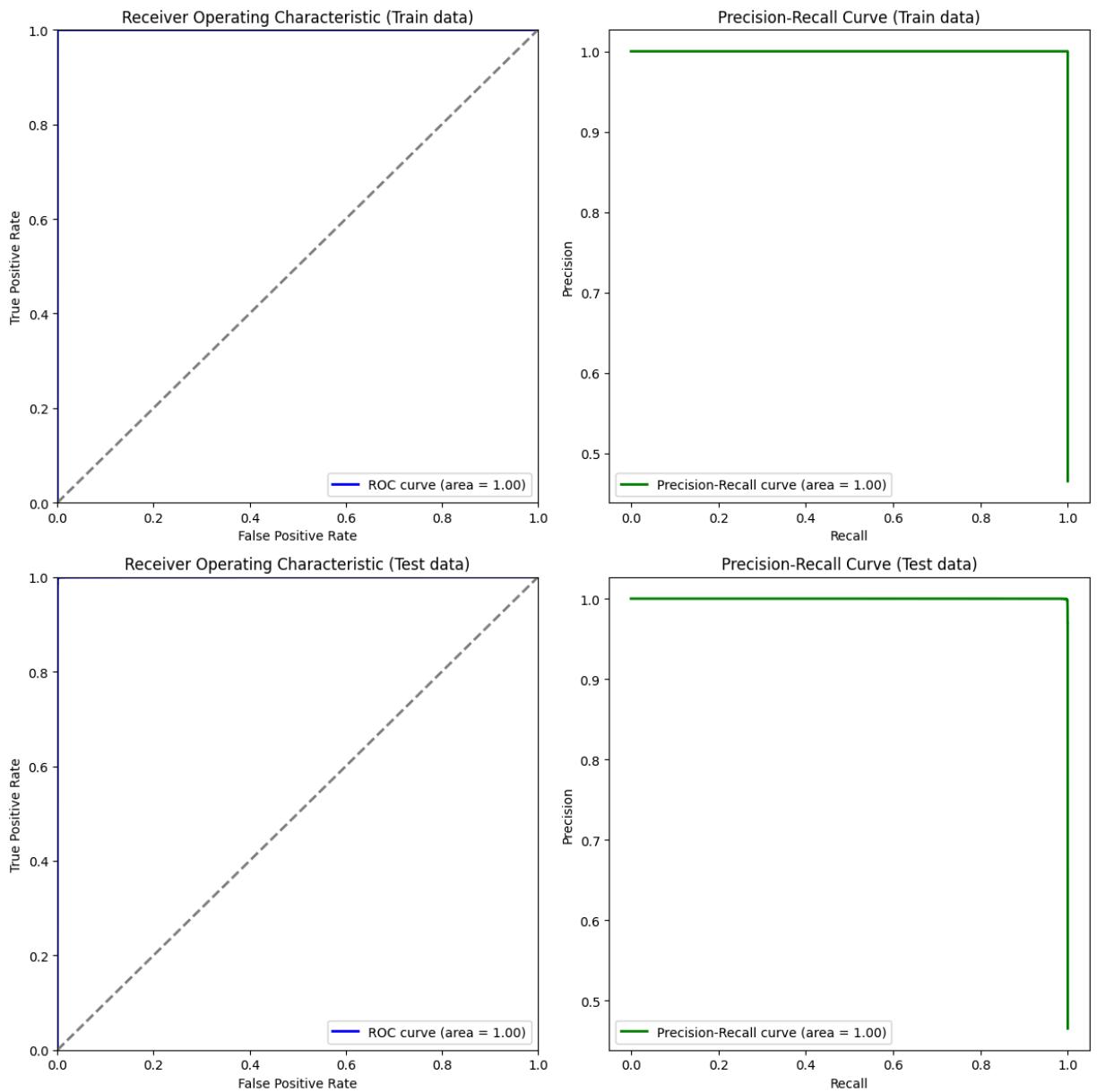
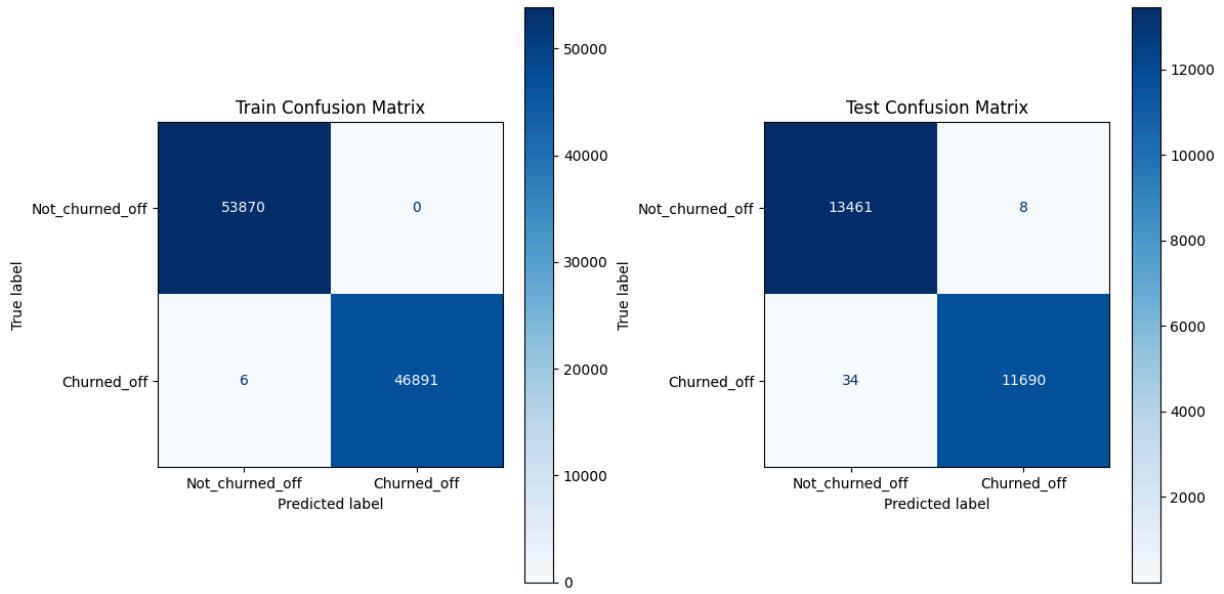
hyper\_parameter\_tuning\_best\_est\_score: 0.9985

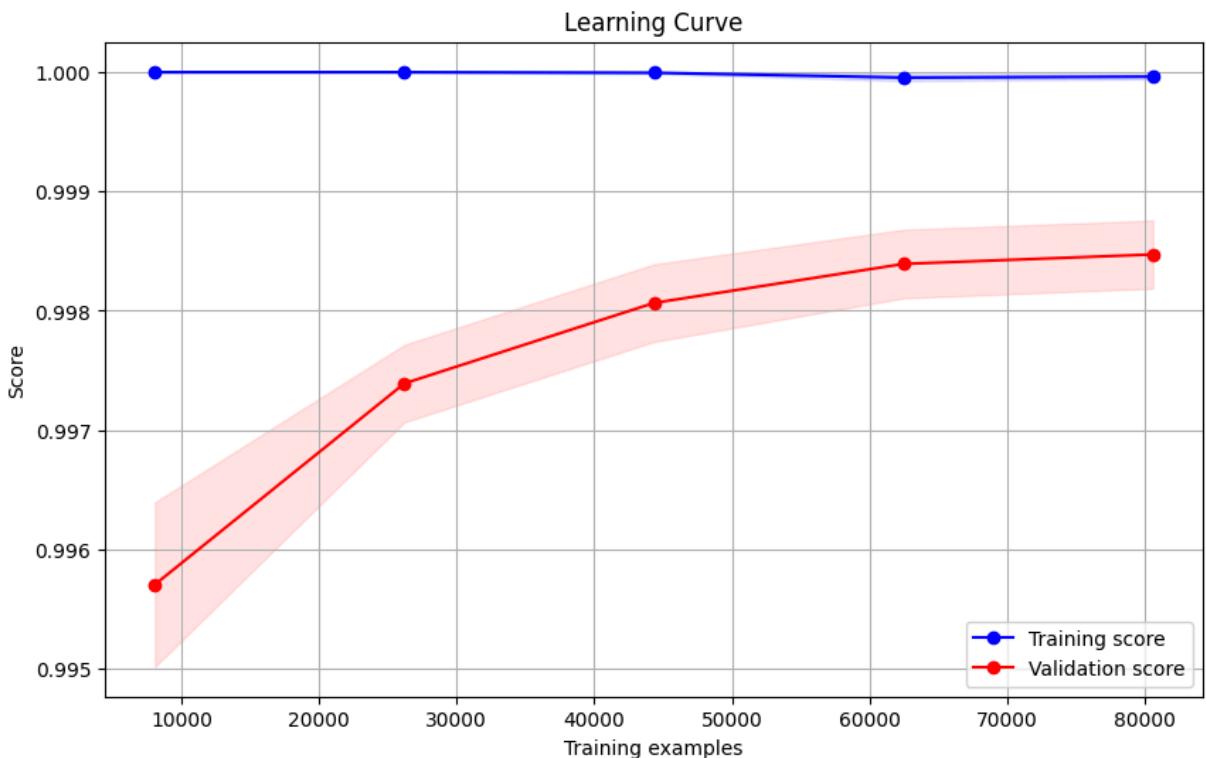
Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	46897
accuracy			1.00	100767
macro avg	1.00	1.00	1.00	100767
weighted avg	1.00	1.00	1.00	100767

Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	11724
accuracy			1.00	25193
macro avg	1.00	1.00	1.00	25193
weighted avg	1.00	1.00	1.00	25193





2025/05/16 20:45:51 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Start timing the tuning process
start_tune_time = time.time()

# Define the parameter grid
param_dist = {
    'n_estimators': stats.randint(100, 200),
    'learning_rate': stats.uniform(0.01, 0.3),
    'max_depth': stats.randint(3, 15),
    'min_samples_split': stats.randint(2, 20),
    'min_samples_leaf': stats.randint(1, 20),
    'max_features': ['auto', 'sqrt', 'log2', None],
    'subsample': stats.uniform(0.7, 0.3),
    'min_impurity_decrease': stats.uniform(0.0, 0.1),
    'random_state': [42]
}

# Initialize the GradientBoostingClassifier
gbc = GradientBoostingClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=gbc,
    param_distributions=param_dist,
    n_iter=5, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=5,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_

# End timing and print tuning time
end_tune_time = time.time()
```

```
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Best parameters found: {'learning_rate': np.float64(0.05184815819561255), 'max_dept
h': 14, 'max_features': None, 'min_impurity_decrease': np.float64(0.0366361843293691
7), 'min_samples_leaf': 15, 'min_samples_split': 13, 'n_estimators': 154, 'random_sta
te': 42, 'subsample': np.float64(0.9949692657420364)}
Best score: 0.9985613514015222
Tuning_time: 1227.9451010227203
```

## Logging Best Gradient boost Model into MLFLOW

```
In [ ]: # Model details
name = "Tuned_Gradient_boost_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_te
```

```

Model: Tuned_Gradient_boost_on_Balanced_Dataset
params: {'learning_rate': np.float64(0.05184815819561255), 'max_depth': 14, 'max_features': None, 'min_impurity_decrease': np.float64(0.03663618432936917), 'min_samples_leaf': 15, 'min_samples_split': 13, 'n_estimators': 154, 'random_state': 42, 'subsample': np.float64(0.9949692657420364)}
Training Time: 345.9908 seconds
Testing Time: 0.7643 seconds
Tuning Time: 1227.9451 seconds
Train Metrics:
Accuracy_train: 0.9997
Precision_train: 0.9999
Recall_train: 0.9996
F1_score_train: 0.9997
F2_score_train: 0.9997
Roc_auc_train: 1.0000
Pr_auc_train: 1.0000

Test Metrics:
Accuracy_test: 0.9985
Precision_test: 0.9982
Recall_test: 0.9986
F1_score_test: 0.9984
F2_score_test: 0.9986
Roc_auc_test: 1.0000
Pr_auc_test: 1.0000

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9986

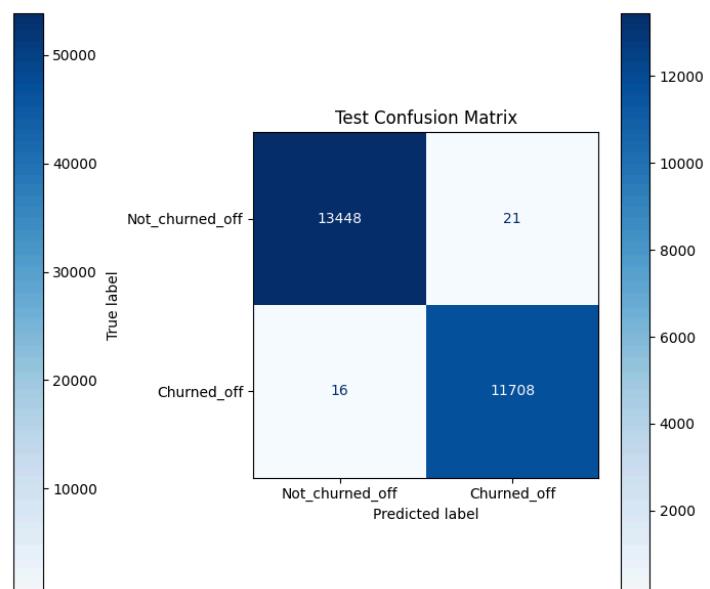
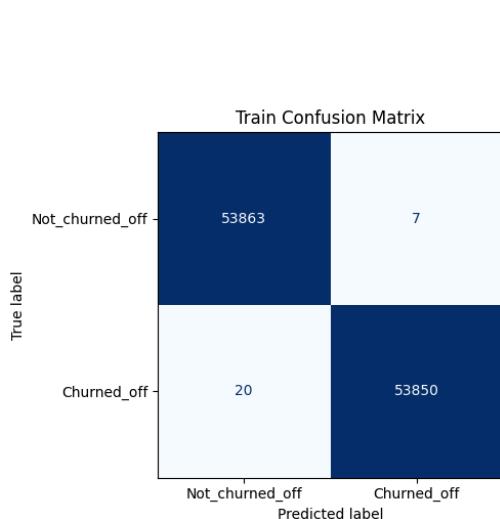
```

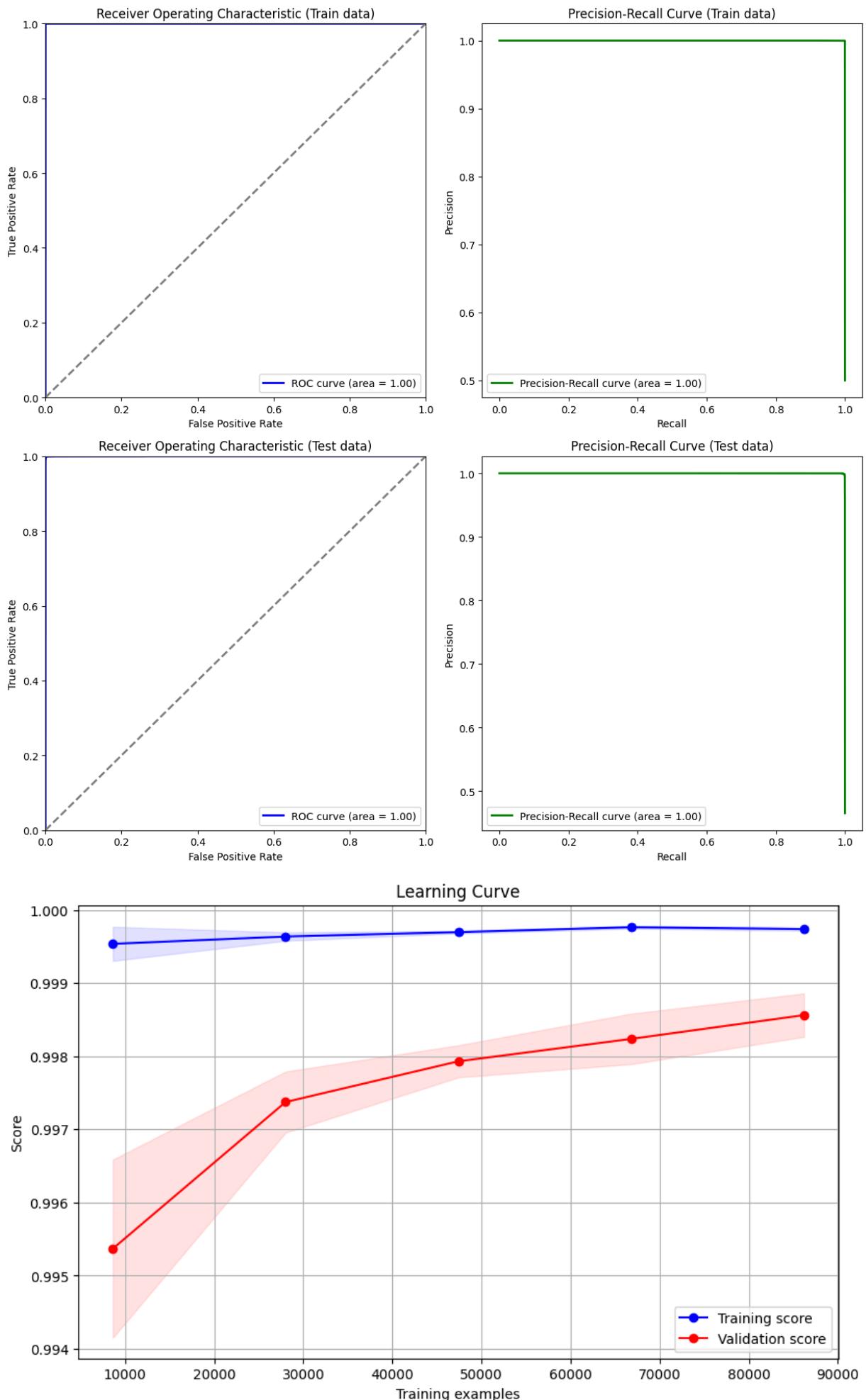
#### Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	53870
accuracy			1.00	107740
macro avg	1.00	1.00	1.00	107740
weighted avg	1.00	1.00	1.00	107740

#### Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	11724
accuracy			1.00	25193
macro avg	1.00	1.00	1.00	25193
weighted avg	1.00	1.00	1.00	25193





```
2025/05/16 21:31:12 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## XGBOOST MODEL

### Imbalanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Start timing the tuning process
start_tune_time = time.time()

# Define the parameter grid
param_dist = {
    'n_estimators': stats.randint(100, 200),           # Number of boosting rounds
    'learning_rate': stats.uniform(0.01, 0.3),          # Learning rate (shrinkage factor)
    'max_depth': stats.randint(3, 15),                  # Maximum depth of a tree
    'min_child_weight': stats.randint(1, 10),            # Minimum sum of instance weight
    'gamma': stats.uniform(0, 0.5),                      # Minimum loss reduction required
    'subsample': stats.uniform(0.7, 0.3),                # Subsample ratio of the training set
    'colsample_bytree': stats.uniform(0.7, 0.3),         # Subsample ratio of columns
    'colsample_bylevel': stats.uniform(0.7, 0.3),        # Subsample ratio of columns
    'colsample_bynode': stats.uniform(0.7, 0.3),         # Subsample ratio of columns
    'reg_alpha': stats.uniform(0, 0.5),                  # L1 regularization term on weights
    'reg_lambda': stats.uniform(0.5, 1.5),                # L2 regularization term on weights
    'scale_pos_weight': stats.uniform(0.5, 2),            # Balance of positive and negative weights
    'booster': ['gbtree', 'gblinear', 'dart'],           # Type of booster to use
    'tree_method': ['auto', 'exact', 'approx', 'hist'],   # Algorithm used to train trees
    'grow_policy': ['depthwise', 'lossguide'],            # Controls the way new nodes are added
    'objective': ['binary:logistic', 'multi:softprob'],   # Learning task and the corresponding loss function
    'sampling_method': ['uniform', 'gradient_based'],     # Method used to sample training data
    'random_state': [42]                                 # Fixed random state for reproducibility
}

# Initialize the XGBClassifier
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_dist,
    n_iter=10,  # Number of parameter settings to try
    cv=5,       # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1   # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_

# End timing and print tuning time
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits  
Best parameters found: {'booster': 'dart', 'colsample\_bylevel': np.float64(0.8416644775485848), 'colsample\_bynode': np.float64(0.7358782737814905), 'colsample\_bytree': np.float64(0.9139734361668984), 'gamma': np.float64(0.3803925243084487), 'grow\_policy': 'lossguide', 'learning\_rate': np.float64(0.08079547592468672), 'max\_depth': 9, 'min\_child\_weight': 9, 'n\_estimators': 178, 'objective': 'binary:logistic', 'random\_state': 42, 'reg\_alpha': np.float64(0.05544541040591566), 'reg\_lambda': np.float64(1.1590047527986551), 'sampling\_method': 'uniform', 'scale\_pos\_weight': np.float64(0.5628583713734685), 'subsample': np.float64(0.890923123379134), 'tree\_method': 'hist'}  
Best score: 0.9982831665239335  
Tuning\_time: 1567.3580090999603

## Logging Best XG boost Model into MLFLOW

```
In [ ]: # Model details
name = "Tuned_XG_boost_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
```

```
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes
```

```

Model: Tuned_XG_boost_on_Imbalanced_Dataset
params: {'booster': 'dart', 'colsample_bylevel': np.float64(0.8416644775485848), 'colsample_bynode': np.float64(0.7358782737814905), 'colsample_bytree': np.float64(0.9139734361668984), 'gamma': np.float64(0.3803925243084487), 'grow_policy': 'lossguide', 'learning_rate': np.float64(0.08079547592468672), 'max_depth': 9, 'min_child_weight': 9, 'n_estimators': 178, 'objective': 'binary:logistic', 'random_state': 42, 'reg_alpha': np.float64(0.05544541040591566), 'reg_lambda': np.float64(1.1590047527986551), 'subsampling': np.float64(0.890923123379134), 'tree_method': 'hist'}
Training Time: 309.1961 seconds
Testing Time: 4.0076 seconds
Tuning Time: 1567.3580 seconds
Train Metrics:
Accuracy_train: 0.9991
Precision_train: 0.9997
Recall_train: 0.9983
F1_score_train: 0.9990
F2_score_train: 0.9986
Roc_auc_train: 1.0000
Pr_auc_train: 1.0000

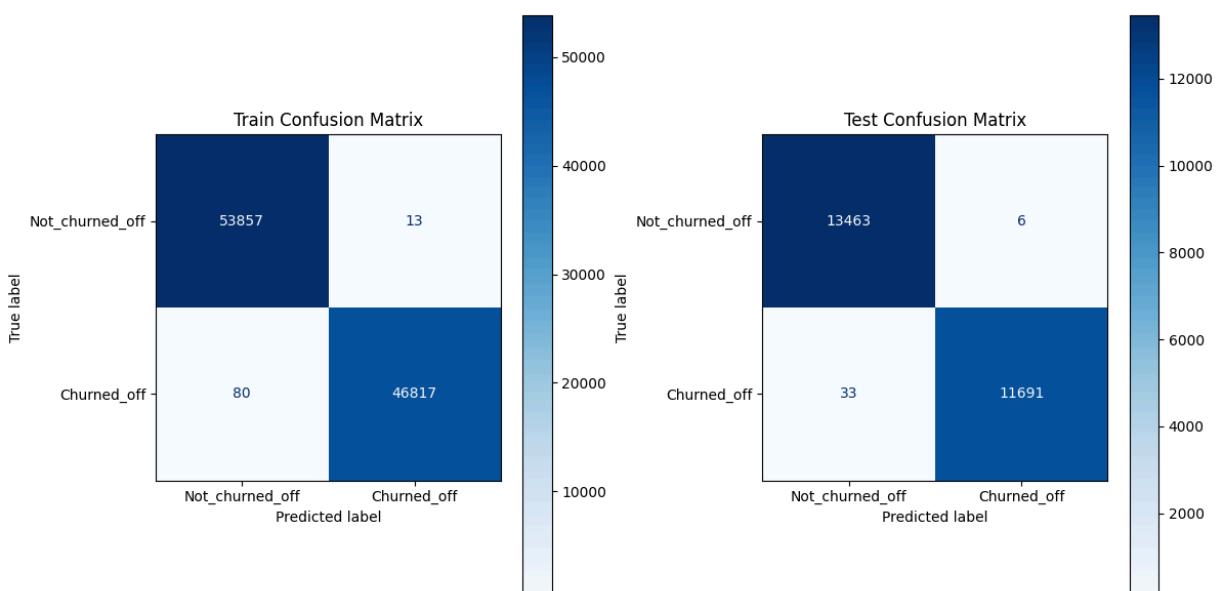
Test Metrics:
Accuracy_test: 0.9985
Precision_test: 0.9995
Recall_test: 0.9972
F1_score_test: 0.9983
F2_score_test: 0.9976
Roc_auc_test: 1.0000
Pr_auc_test: 1.0000

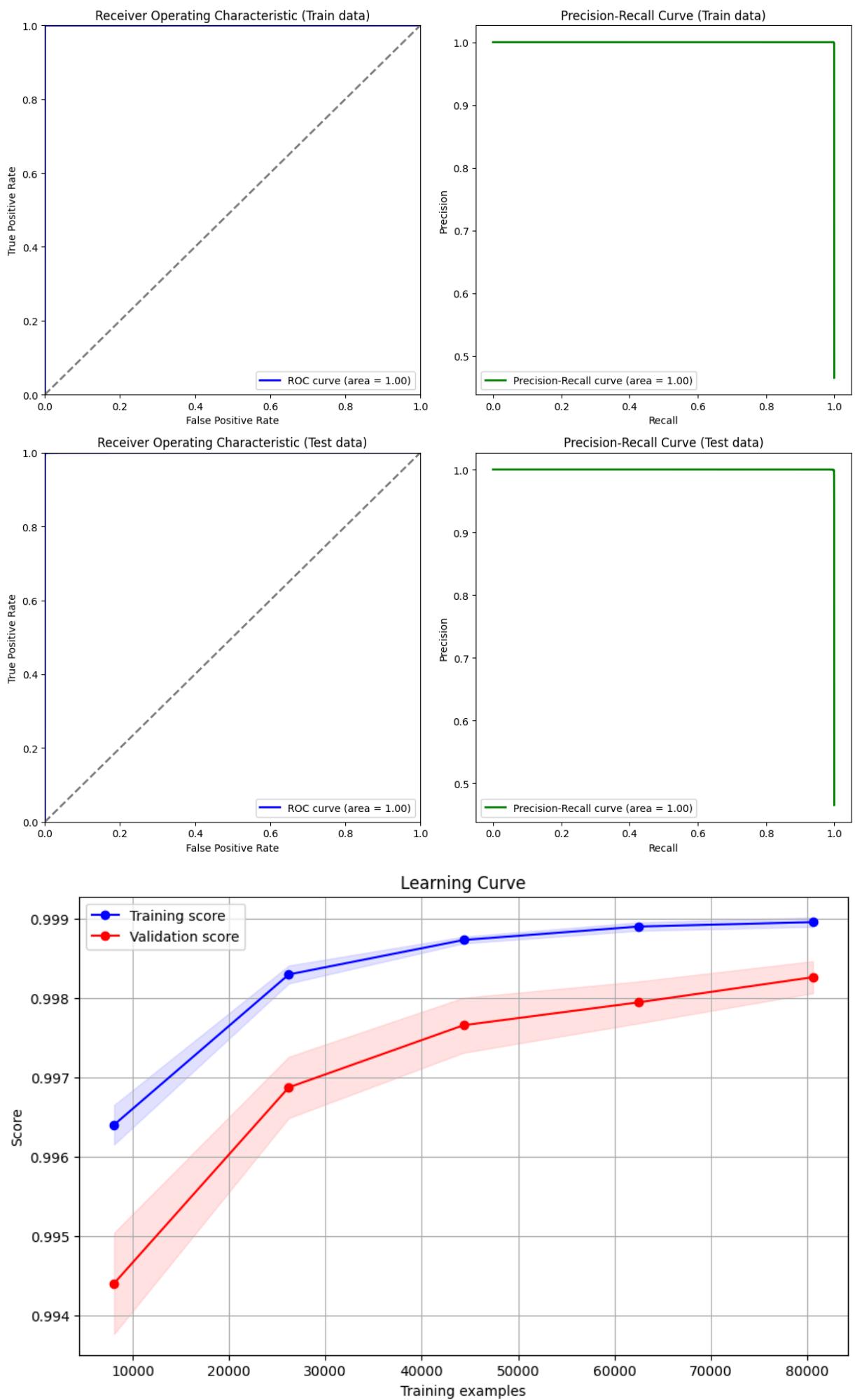
Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9983

```

Train Classification Report:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	46897
accuracy			1.00	100767
macro avg	1.00	1.00	1.00	100767
weighted avg	1.00	1.00	1.00	100767

Test Classification Report:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	11724
accuracy			1.00	25193
macro avg	1.00	1.00	1.00	25193
weighted avg	1.00	1.00	1.00	25193





```
2025/05/16 22:56:21 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Start timing the tuning process
start_tune_time = time.time()

# Define the parameter grid
```

```

param_dist = {
    'n_estimators': stats.randint(100, 200),
    'learning_rate': stats.uniform(0.01, 0.3),
    'max_depth': stats.randint(3, 15),
    'min_child_weight': stats.randint(1, 10),
    'gamma': stats.uniform(0, 0.5),
    'subsample': stats.uniform(0.7, 0.9),
    'colsample_bytree': stats.uniform(0.7, 0.9),
    'colsample_bylevel': stats.uniform(0.7, 0.9),
    'colsample_bynode': stats.uniform(0.7, 0.9),
    'reg_alpha': stats.uniform(0, 0.5),
    'reg_lambda': stats.uniform(0.5, 1.5),
    'scale_pos_weight': stats.uniform(0.5, 2),
    'booster': ['gbtree', 'gblinear', 'dart'],
    'tree_method': ['auto', 'exact', 'approx', 'hist'],
    'grow_policy': ['depthwise', 'lossguide'],
    'objective': ['binary:logistic', 'multi:softprob'],
    'sampling_method': ['uniform', 'gradient_based'],
    'random_state': [42]
}

# Initialize the XGBClassifier
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_

# End timing and print tuning time
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits  
Best parameters found: {'booster': 'dart', 'colsample\_bylevel': np.float64(0.8416644775485848), 'colsample\_bynode': np.float64(0.7358782737814905), 'colsample\_bytree': np.float64(0.9139734361668984), 'gamma': np.float64(0.3803925243084487), 'grow\_policy': 'lossguide', 'learning\_rate': np.float64(0.08079547592468672), 'max\_depth': 9, 'min\_child\_weight': 9, 'n\_estimators': 178, 'objective': 'binary:logistic', 'random\_state': 42, 'reg\_alpha': np.float64(0.05544541040591566), 'reg\_lambda': np.float64(1.1590047527986551), 'sampling\_method': 'uniform', 'scale\_pos\_weight': np.float64(0.5628583713734685), 'subsample': np.float64(0.890923123379134), 'tree\_method': 'hist'}  
Best score: 0.9982736216818265  
Tuning\_time: 1835.7246253490448

## Logging Best XG boost Model into MLFLOW

```

In [ ]: # Model details
name = "Tuned_XG_boost_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time

```

```
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {test"ing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_tes
```

Model: Tuned\_XG\_boost\_on\_Balanced\_Dataset  
params: {'booster': 'dart', 'colsample\_bylevel': np.float64(0.8416644775485848), 'colsample\_bynode': np.float64(0.7358782737814905), 'colsample\_bytree': np.float64(0.9139734361668984), 'gamma': np.float64(0.3803925243084487), 'grow\_policy': 'lossguide', 'learning\_rate': np.float64(0.08079547592468672), 'max\_depth': 9, 'min\_child\_weight': 9, 'n\_estimators': 178, 'objective': 'binary:logistic', 'random\_state': 42, 'reg\_alpha': np.float64(0.05544541040591566), 'reg\_lambda': np.float64(1.1590047527986551), 'sampling\_method': 'uniform', 'scale\_pos\_weight': np.float64(0.5628583713734685), 'subsample': np.float64(0.890923123379134), 'tree\_method': 'hist'}

Training Time: 358.9655 seconds

Testing Time: 4.4654 seconds

Tuning Time: 1835.7246 seconds

Train Metrics:

Accuracy\_train: 0.9991  
Precision\_train: 0.9998  
Recall\_train: 0.9984  
F1\_score\_train: 0.9991  
F2\_score\_train: 0.9987  
Roc\_auc\_train: 1.0000  
Pr\_auc\_train: 1.0000

Test Metrics:

Accuracy\_test: 0.9981  
Precision\_test: 0.9995  
Recall\_test: 0.9965  
F1\_score\_test: 0.9980  
F2\_score\_test: 0.9971  
Roc\_auc\_test: 1.0000  
Pr\_auc\_test: 1.0000

Tuning Metrics:

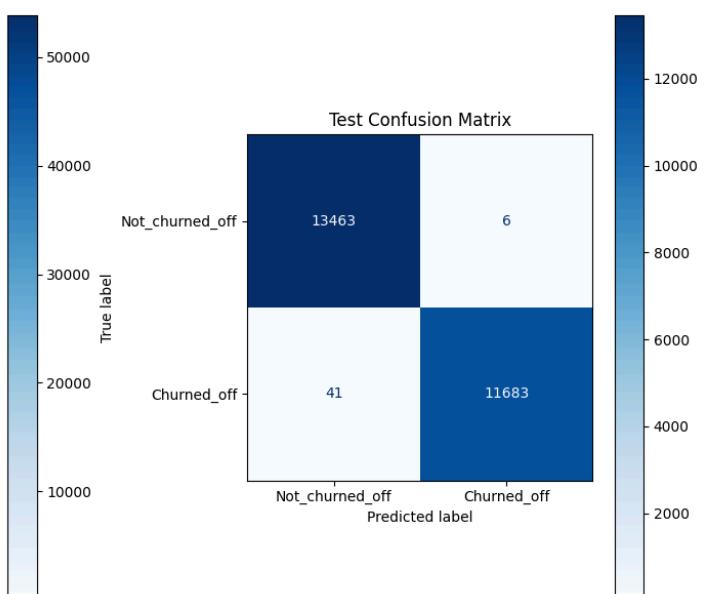
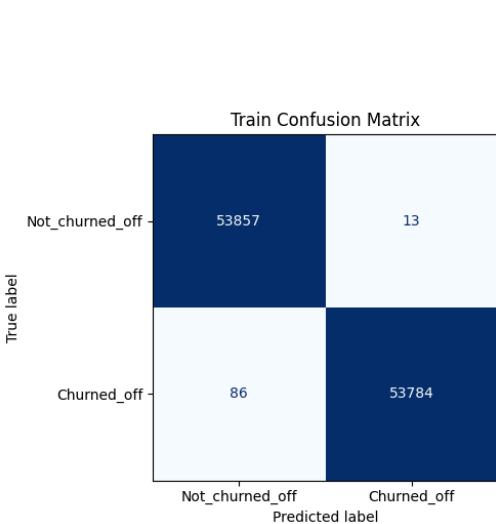
hyper\_parameter\_tuning\_best\_est\_score: 0.9983

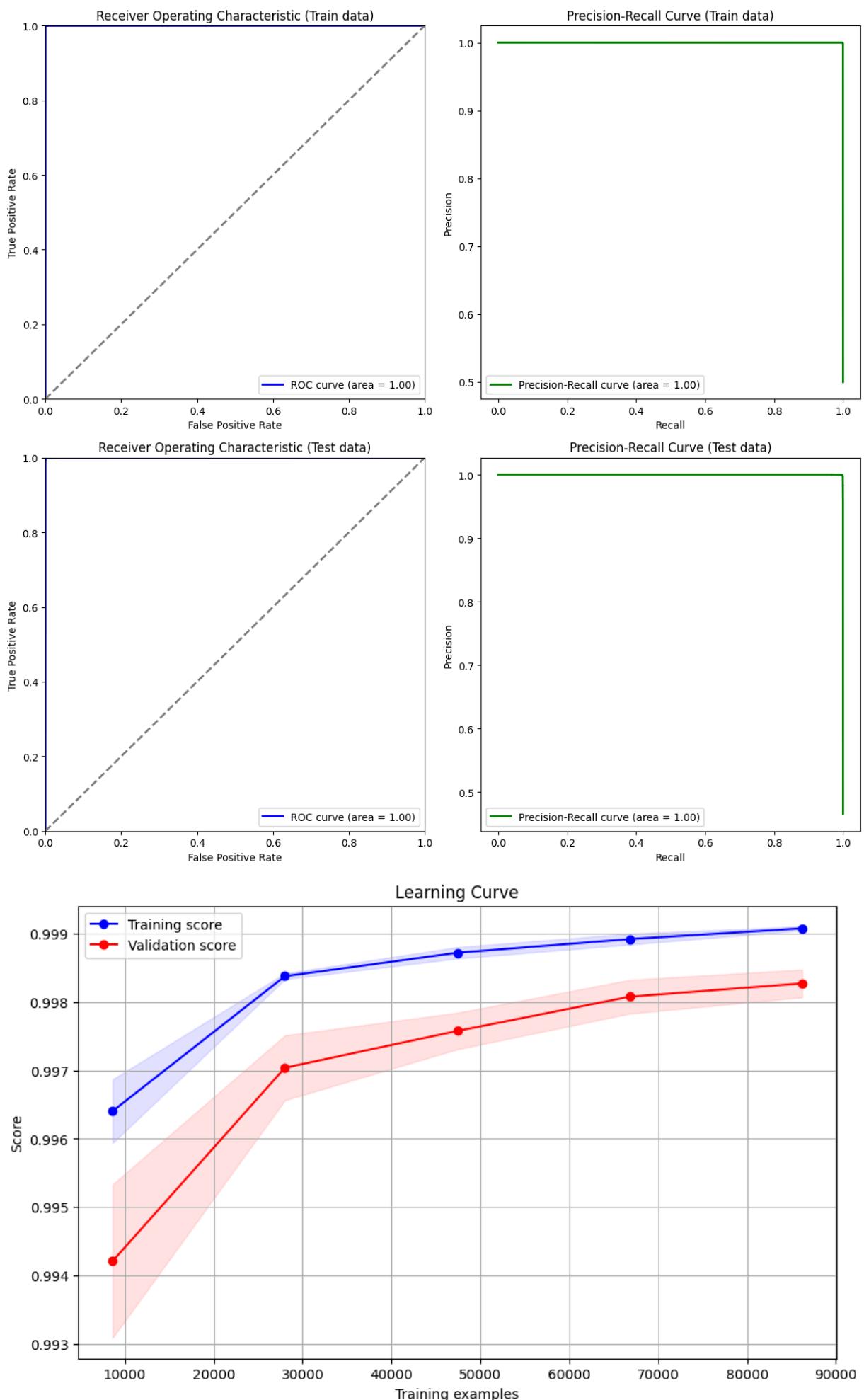
Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	53870
accuracy			1.00	107740
macro avg	1.00	1.00	1.00	107740
weighted avg	1.00	1.00	1.00	107740

Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	11724
accuracy			1.00	25193
macro avg	1.00	1.00	1.00	25193
weighted avg	1.00	1.00	1.00	25193





```
2025/05/17 00:36:06 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## LIGHT\_GB MODEL

### Imbalanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Start timing the tuning process
start_tune_time = time.time()

# Define the parameter grid
param_dist = {
    'num_leaves': stats.randint(20, 150),
    'max_depth': stats.randint(3, 15),
    'learning_rate': stats.uniform(0.01, 0.3),
    'n_estimators': stats.randint(100, 200),
    'min_child_samples': stats.randint(10, 100),
    'min_child_weight': stats.uniform(1e-3, 1e-1),
    'subsample': stats.uniform(0.5, 1.0),
    'colsample_bytree': stats.uniform(0.5, 1.0),
    'reg_alpha': stats.uniform(0, 0.5),
    'reg_lambda': stats.uniform(0.5, 1.5),
    'scale_pos_weight': stats.uniform(0.5, 2),
    'boosting_type': ['gbdt', 'dart', 'goss'],
    'objective': ['binary', 'multiclass'],
    'bagging_fraction': stats.uniform(0.5, 1.0),
    'bagging_freq': stats.randint(1, 10),
    'feature_fraction': stats.uniform(0.5, 1.0),
    'min_split_gain': stats.uniform(0, 0.1),
    'min_data_in_leaf': stats.randint(20, 100),
    'random_state': [42],
}

# Initialize the LGBMClassifier
lgbm_clf = LGBMClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=lgbm_clf,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=False,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_

# End timing and print tuning time
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)
warnings.filterwarnings('ignore')
```











```

freq': 9, 'boosting_type': 'gbdt', 'colsample_bytree': np.float64(1.302196980754039
7), 'feature_fraction': np.float64(0.5745506436797708), 'learning_rate': np.float64
(0.3060660809801552), 'max_depth': 10, 'min_child_samples': 10, 'min_child_weight': np.
float64(0.020871568153417244), 'min_data_in_leaf': 82, 'min_split_gain': np.float64
(0.08154614284548342), 'n_estimators': 180, 'num_leaves': 52, 'objective': 'binary',
'random_state': 42, 'reg_alpha': np.float64(0.03702232586704518), 'reg_lambda': np.fl
oat64(1.0376985928164089), 'scale_pos_weight': np.float64(0.7317381190502594), 'subsa
mple': np.float64(1.3631034258755936)}
Best score: 0.9987793621920226
Tuning_time: 37.226678133010864

```

### Logging Best Light GBM Model into MLFLOW

```

In [ ]: # Model details
name = "Tuned_Light_GBM_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes

```











```

d. Current value: min_data_in_leaf=82
[LightGBM] [Warning] feature_fraction is set=0.5745506436797708, colsample_bytree=1.3
021969807540397 will be ignored. Current value: feature_fraction=0.5745506436797708
[LightGBM] [Warning] bagging_fraction is set=0.7809345096873808, subsample=1.36310342
58755936 will be ignored. Current value: bagging_fraction=0.7809345096873808
[LightGBM] [Warning] bagging_freq is set=9, subsample_freq=0 will be ignored. Current
value: bagging_freq=9
[LightGBM] [Warning] min_data_in_leaf is set=82, min_child_samples=10 will be ignore
d. Current value: min_data_in_leaf=82
[LightGBM] [Warning] feature_fraction is set=0.5745506436797708, colsample_bytree=1.3
021969807540397 will be ignored. Current value: feature_fraction=0.5745506436797708
[LightGBM] [Warning] bagging_fraction is set=0.7809345096873808, subsample=1.36310342
58755936 will be ignored. Current value: bagging_fraction=0.7809345096873808
[LightGBM] [Warning] bagging_freq is set=9, subsample_freq=0 will be ignored. Current
value: bagging_freq=9
Model: Tuned_Light_GBM_on_Imbalanced_Dataset
params: {'bagging_fraction': np.float64(0.7809345096873808), 'bagging_freq': 9, 'boos
ting_type': 'gbdt', 'colsample_bytree': np.float64(1.3021969807540397), 'feature_frac
tion': np.float64(0.5745506436797708), 'learning_rate': np.float64(0.306066080980155
2), 'max_depth': 10, 'min_child_samples': 10, 'min_child_weight': np.float64(0.020871
568153417244), 'min_data_in_leaf': 82, 'min_split_gain': np.float64(0.081546142845483
42), 'n_estimators': 180, 'num_leaves': 52, 'objective': 'binary', 'random_state': 4
2, 'reg_alpha': np.float64(0.03702232586704518), 'reg_lambda': np.float64(1.037698592
8164089), 'scale_pos_weight': np.float64(0.7317381190502594), 'subsample': np.float64
(1.3631034258755936)}
Training Time: 4.9142 seconds
Testing Time: 0.5938 seconds
Tuning Time: 37.2267 seconds
[LightGBM] [Warning] min_data_in_leaf is set=82, min_child_samples=10 will be ignore
d. Current value: min_data_in_leaf=82
[LightGBM] [Warning] feature_fraction is set=0.5745506436797708, colsample_bytree=1.3
021969807540397 will be ignored. Current value: feature_fraction=0.5745506436797708
[LightGBM] [Warning] bagging_fraction is set=0.7809345096873808, subsample=1.36310342
58755936 will be ignored. Current value: bagging_fraction=0.7809345096873808
[LightGBM] [Warning] bagging_freq is set=9, subsample_freq=0 will be ignored. Current
value: bagging_freq=9
[LightGBM] [Warning] min_data_in_leaf is set=82, min_child_samples=10 will be ignore
d. Current value: min_data_in_leaf=82
[LightGBM] [Warning] feature_fraction is set=0.5745506436797708, colsample_bytree=1.3
021969807540397 will be ignored. Current value: feature_fraction=0.5745506436797708
[LightGBM] [Warning] bagging_fraction is set=0.7809345096873808, subsample=1.36310342
58755936 will be ignored. Current value: bagging_fraction=0.7809345096873808
[LightGBM] [Warning] bagging_freq is set=9, subsample_freq=0 will be ignored. Current
value: bagging_freq=9
Train Metrics:
Accuracy_train: 1.0000
Precision_train: 1.0000
Recall_train: 1.0000
F1_score_train: 1.0000
F2_score_train: 1.0000
Roc_auc_train: 1.0000
Pr_auc_train: 1.0000

Test Metrics:
Accuracy_test: 0.9988
Precision_test: 0.9995
Recall_test: 0.9980
F1_score_test: 0.9988
F2_score_test: 0.9983
Roc_auc_test: 1.0000
Pr_auc_test: 1.0000

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9988

Train Classification Report:
      precision    recall   f1-score   support
      0         1.00     1.00     1.00     53870
      1         1.00     1.00     1.00     46897

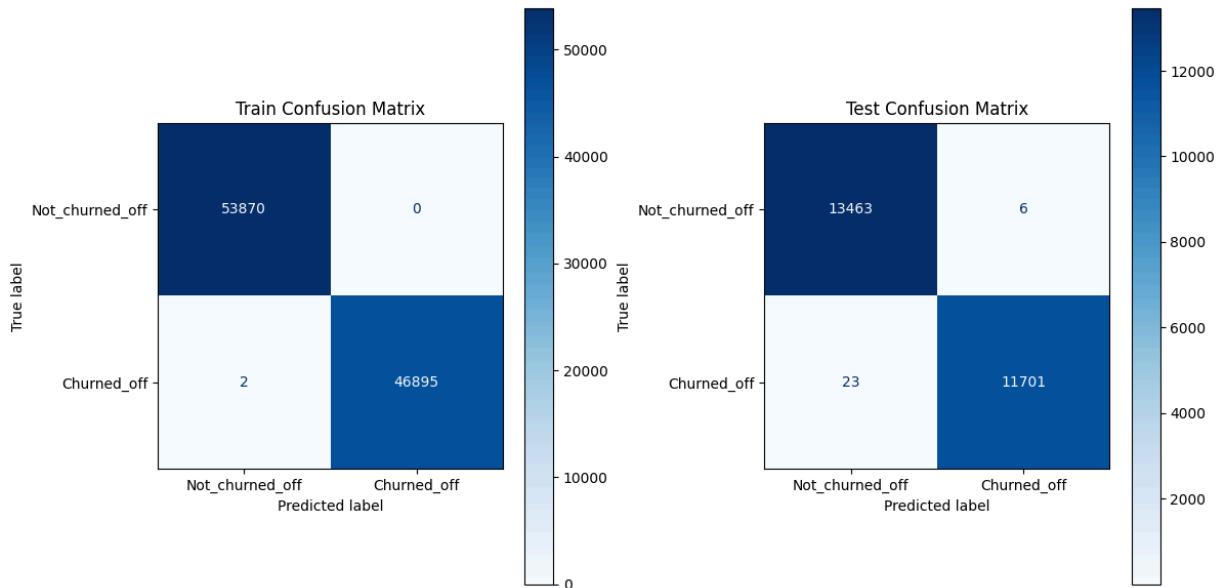
      accuracy                  1.00     100767
      macro avg       1.00     1.00     1.00     100767
      weighted avg    1.00     1.00     1.00     100767

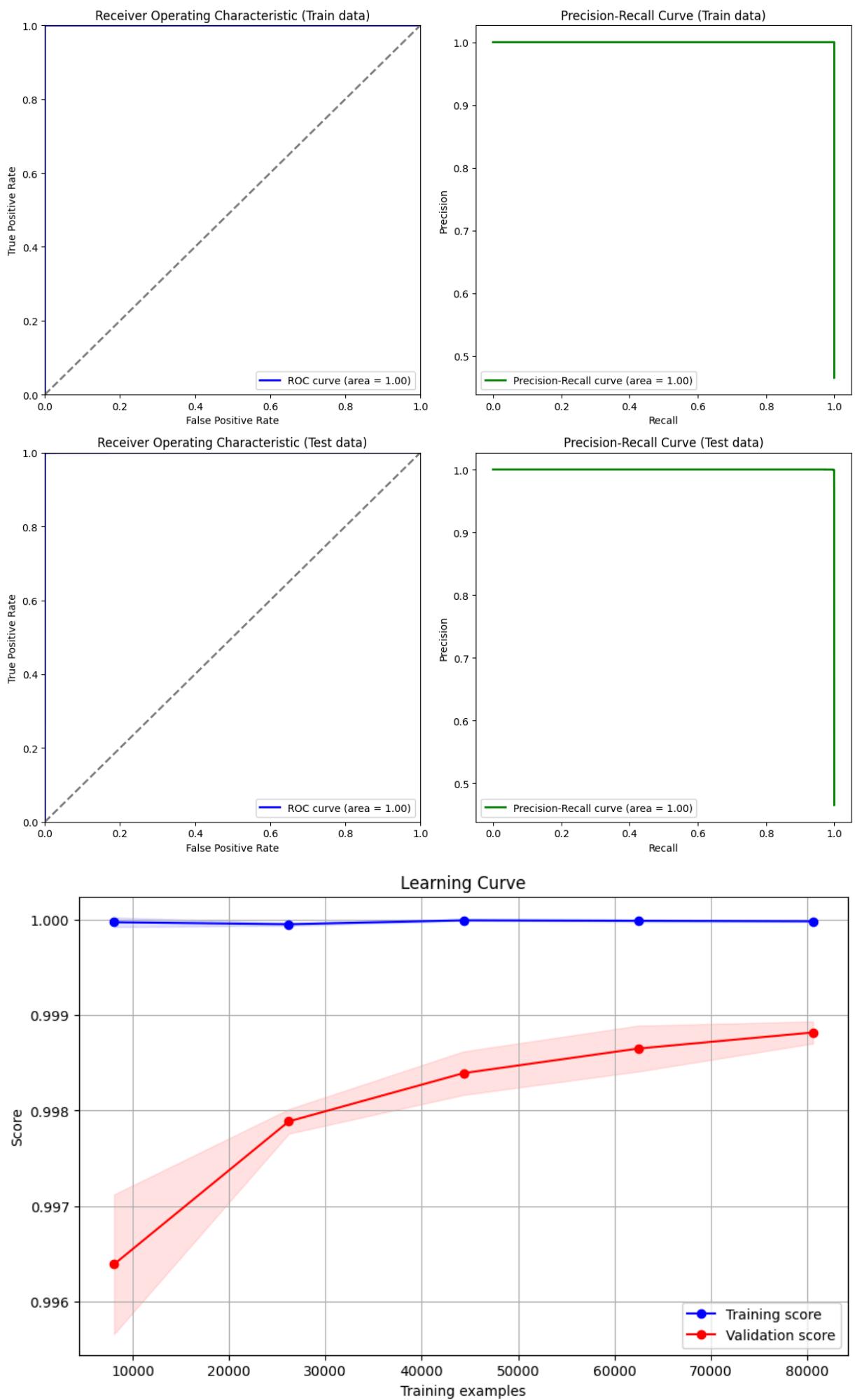
```

#### Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	11724
accuracy			1.00	25193
macro avg	1.00	1.00	1.00	25193
weighted avg	1.00	1.00	1.00	25193

[LightGBM] [Warning] min\_data\_in\_leaf is set=82, min\_child\_samples=10 will be ignored. Current value: min\_data\_in\_leaf=82  
[LightGBM] [Warning] feature\_fraction is set=0.5745506436797708, colsample\_bytree=1.3021969807540397 will be ignored. Current value: feature\_fraction=0.5745506436797708  
[LightGBM] [Warning] bagging\_fraction is set=0.7809345096873808, subsample=1.3631034258755936 will be ignored. Current value: bagging\_fraction=0.7809345096873808  
[LightGBM] [Warning] bagging\_freq is set=9, subsample\_freq=0 will be ignored. Current value: bagging\_freq=9  
[LightGBM] [Warning] min\_data\_in\_leaf is set=82, min\_child\_samples=10 will be ignored. Current value: min\_data\_in\_leaf=82  
[LightGBM] [Warning] feature\_fraction is set=0.5745506436797708, colsample\_bytree=1.3021969807540397 will be ignored. Current value: feature\_fraction=0.5745506436797708  
[LightGBM] [Warning] bagging\_fraction is set=0.7809345096873808, subsample=1.3631034258755936 will be ignored. Current value: bagging\_fraction=0.7809345096873808  
[LightGBM] [Warning] bagging\_freq is set=9, subsample\_freq=0 will be ignored. Current value: bagging\_freq=9





```
2025/05/17 00:38:05 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## Balanced Dataset

### Hyper parameter Tuning

```
In [ ]: # Start timing the tuning process
start_tune_time = time.time()

# Define the parameter grid
```

```

param_dist = {
    'num_leaves': stats.randint(20, 150),
    'max_depth': stats.randint(3, 15),
    'learning_rate': stats.uniform(0.01, 0.3),
    'n_estimators': stats.randint(100, 200),
    'min_child_samples': stats.randint(10, 100),
    'min_child_weight': stats.uniform(1e-3, 1e-1),
    'subsample': stats.uniform(0.5, 1.0),
    'colsample_bytree': stats.uniform(0.5, 1.0),
    'reg_alpha': stats.uniform(0, 0.5),
    'reg_lambda': stats.uniform(0.5, 1.5),
    'scale_pos_weight': stats.uniform(0.5, 2),
    'boosting_type': ['gbdt', 'dart', 'goss'],
    'objective': ['binary', 'multiclass'],
    'bagging_fraction': stats.uniform(0.5, 1.0),
    'bagging_freq': stats.randint(1, 10),
    'feature_fraction': stats.uniform(0.5, 1.0),
    'min_split_gain': stats.uniform(0, 0.1),
    'min_data_in_leaf': stats.randint(20, 100),
    'random_state': [42],
}

# Initialize the LGBMClassifier
lgbm_clf = LGBMClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=lgbm_clf,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=False,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_

# End timing and print tuning time
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)
warnings.filterwarnings('ignore')

```











```
p.float64(0.020871568153417244), 'min_data_in_leaf': 82, 'min_split_gain': np.float64(0.08154614284548342), 'n_estimators': 180, 'num_leaves': 52, 'objective': 'binary', 'random_state': 42, 'reg_alpha': np.float64(0.03702232586704518), 'reg_lambda': np.float64(1.0376985928164089), 'scale_pos_weight': np.float64(0.7317381190502594), 'subsample': np.float64(1.3631034258755936)}  
Best score: 0.9987005754594394  
Tuning_time: 43.81748700141907
```

## Logging Best Light GBM boost Model into MLFLOW

```
In [ ]:  
# Model details  
name = "Tuned_Light_GBM_on_Balanced_Dataset"  
bal_type = "Balanced"  
model = random_search.best_estimator_  
params = random_search.best_params_  
  
# Record training time  
start_train_time = time.time()  
model.fit(X_train_bal, y_train_bal)  
end_train_time = time.time()  
  
# Calculate training time  
training_time = end_train_time - start_train_time  
  
# Record testing time  
start_test_time = time.time()  
y_pred_bal_train = model.predict(X_train_bal)  
y_pred_bal_test = model.predict(X_test_bal)  
end_test_time = time.time()  
  
# Calculate testing time  
testing_time = end_test_time - start_test_time  
  
# Print model name and times  
print(f"Model: {name}")  
print(f"params: {params}")  
print(f"Training Time: {training_time:.4f} seconds")  
print(f"Testing Time: {testing_time:.4f} seconds")  
print(f"Tuning Time: {tuning_time:.4f} seconds")  
  
# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing  
time_df,feature_importance_df = log_time_and_feature_importances_df(time_df,feature_<br/>mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_te
```









```
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] min_data_in_leaf is set=82, min_child_samples=10 will be ignore
d. Current value: min_data_in_leaf=82
[LightGBM] [Warning] feature_fraction is set=0.5745506436797708, colsample_bytree=1.3
021969807540397 will be ignored. Current value: feature_fraction=0.5745506436797708
```

```

[LightGBM] [Warning] bagging_fraction is set=0.7809345096873808, subsample=1.36310342
58755936 will be ignored. Current value: bagging_fraction=0.7809345096873808
[LightGBM] [Warning] bagging_freq is set=9, subsample_freq=0 will be ignored. Current
value: bagging_freq=9
[LightGBM] [Warning] min_data_in_leaf is set=82, min_child_samples=10 will be ignore
d. Current value: min_data_in_leaf=82
[LightGBM] [Warning] feature_fraction is set=0.5745506436797708, colsample_bytree=1.3
021969807540397 will be ignored. Current value: feature_fraction=0.5745506436797708
[LightGBM] [Warning] bagging_fraction is set=0.7809345096873808, subsample=1.36310342
58755936 will be ignored. Current value: bagging_fraction=0.7809345096873808
[LightGBM] [Warning] bagging_freq is set=9, subsample_freq=0 will be ignored. Current
value: bagging_freq=9
Model: Tuned_Light_GBM_on_Balanced_Dataset
params: {'bagging_fraction': np.float64(0.7809345096873808), 'bagging_freq': 9, 'boos
ting_type': 'gbdt', 'colsample_bytree': np.float64(1.3021969807540397), 'feature_frac
tion': np.float64(0.5745506436797708), 'learning_rate': np.float64(0.306066080980155
2), 'max_depth': 10, 'min_child_samples': 10, 'min_child_weight': np.float64(0.020871
568153417244), 'min_data_in_leaf': 82, 'min_split_gain': np.float64(0.081546142845483
42), 'n_estimators': 180, 'num_leaves': 52, 'objective': 'binary', 'random_state': 4
2, 'reg_alpha': np.float64(0.03702232586704518), 'reg_lambda': np.float64(1.037698592
8164089), 'scale_pos_weight': np.float64(0.7317381190502594), 'subsample': np.float64
(1.3631034258755936)}
Training Time: 4.7298 seconds
Testing Time: 0.6816 seconds
Tuning Time: 43.8175 seconds
[LightGBM] [Warning] min_data_in_leaf is set=82, min_child_samples=10 will be ignore
d. Current value: min_data_in_leaf=82
[LightGBM] [Warning] feature_fraction is set=0.5745506436797708, colsample_bytree=1.3
021969807540397 will be ignored. Current value: feature_fraction=0.5745506436797708
[LightGBM] [Warning] bagging_fraction is set=0.7809345096873808, subsample=1.36310342
58755936 will be ignored. Current value: bagging_fraction=0.7809345096873808
[LightGBM] [Warning] bagging_freq is set=9, subsample_freq=0 will be ignored. Current
value: bagging_freq=9
[LightGBM] [Warning] min_data_in_leaf is set=82, min_child_samples=10 will be ignore
d. Current value: min_data_in_leaf=82
[LightGBM] [Warning] feature_fraction is set=0.5745506436797708, colsample_bytree=1.3
021969807540397 will be ignored. Current value: feature_fraction=0.5745506436797708
[LightGBM] [Warning] bagging_fraction is set=0.7809345096873808, subsample=1.36310342
58755936 will be ignored. Current value: bagging_fraction=0.7809345096873808
[LightGBM] [Warning] bagging_freq is set=9, subsample_freq=0 will be ignored. Current
value: bagging_freq=9
Train Metrics:
Accuracy_train: 1.0000
Precision_train: 1.0000
Recall_train: 1.0000
F1_score_train: 1.0000
F2_score_train: 1.0000
Roc_auc_train: 1.0000
Pr_auc_train: 1.0000

Test Metrics:
Accuracy_test: 0.9988
Precision_test: 0.9995
Recall_test: 0.9980
F1_score_test: 0.9988
F2_score_test: 0.9983
Roc_auc_test: 1.0000
Pr_auc_test: 1.0000

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9987

Train Classification Report:
precision    recall   f1-score   support
0          1.00      1.00      1.00      53870
1          1.00      1.00      1.00      53870

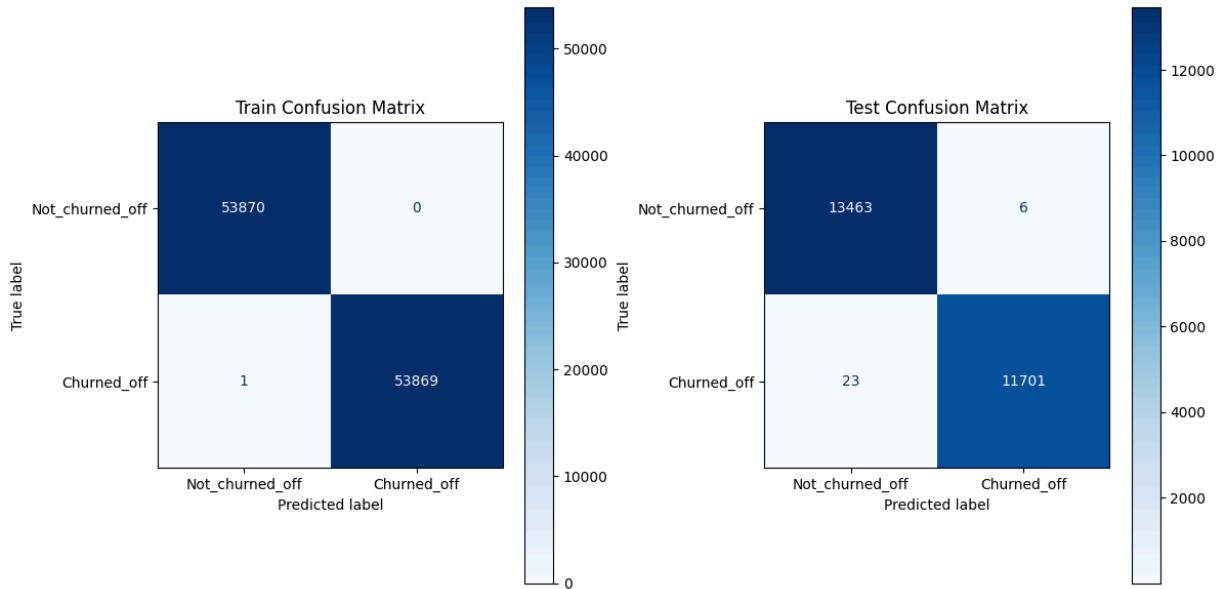
accuracy                  1.00      107740
macro avg       1.00      1.00      1.00      107740
weighted avg     1.00      1.00      1.00      107740

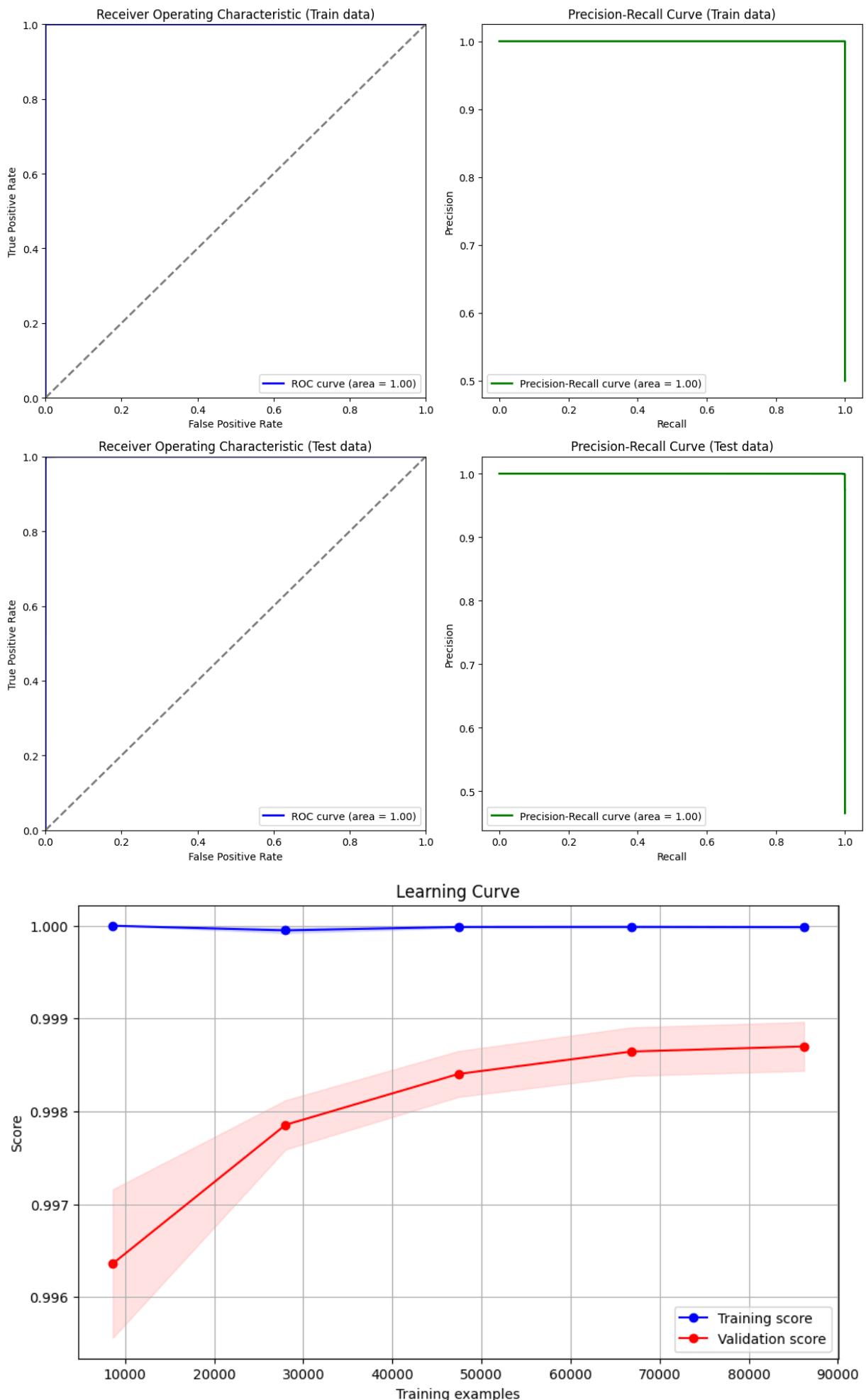
Test Classification Report:

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	11724
accuracy			1.00	25193
macro avg	1.00	1.00	1.00	25193
weighted avg	1.00	1.00	1.00	25193

[LightGBM] [Warning] min\_data\_in\_leaf is set=82, min\_child\_samples=10 will be ignored. Current value: min\_data\_in\_leaf=82  
[LightGBM] [Warning] feature\_fraction is set=0.5745506436797708, colsample\_bytree=1.3021969807540397 will be ignored. Current value: feature\_fraction=0.5745506436797708  
[LightGBM] [Warning] bagging\_fraction is set=0.7809345096873808, subsample=1.3631034258755936 will be ignored. Current value: bagging\_fraction=0.7809345096873808  
[LightGBM] [Warning] bagging\_freq is set=9, subsample\_freq=0 will be ignored. Current value: bagging\_freq=9  
[LightGBM] [Warning] min\_data\_in\_leaf is set=82, min\_child\_samples=10 will be ignored. Current value: min\_data\_in\_leaf=82  
[LightGBM] [Warning] feature\_fraction is set=0.5745506436797708, colsample\_bytree=1.3021969807540397 will be ignored. Current value: feature\_fraction=0.5745506436797708  
[LightGBM] [Warning] bagging\_fraction is set=0.7809345096873808, subsample=1.3631034258755936 will be ignored. Current value: bagging\_fraction=0.7809345096873808  
[LightGBM] [Warning] bagging\_freq is set=9, subsample\_freq=0 will be ignored. Current value: bagging\_freq=9





```
2025/05/17 00:40:16 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

## CHAPTER 5: RESULTS EVALUATION

### ALL\_LOGGED\_METRICS

```
In [ ]: all_logged_metrics_df = all_logged_metrics()
```

```
all_logged_metrics_df
```

Out[ ]:

	run_name	bal_type	params_dict	met
0	Tuned_Light_GBM_on_Balanced_Dataset	Balanced	{'params.subsample': '1.3631034258755936', 'pa...}	
1	Tuned_Light_GBM_on_Imbalanced_Dataset	Imbalanced	{'params.subsample': '1.3631034258755936', 'pa...}	
2	Tuned_XG_boost_on_Balanced_Dataset	Balanced	{'params.subsample': '0.890923123379134', 'par...}	
3	Tuned_XG_boost_on_Imbalanced_Dataset	Imbalanced	{'params.subsample': '0.890923123379134', 'par...}	
4	Tuned_Gradient_boost_on_Balanced_Dataset	Balanced	{'params.subsample': '0.9949692657420364', 'pa...}	
5	Tuned_Gradient_boost_on_Imbalanced_Dataset	Imbalanced	{'params.subsample': '0.7637017332034828', 'pa...}	
6	Tuned_Adaboost_on_Balanced_Dataset	Balanced	{'params.n_estimators': '120', 'params.bal_typ...}	
7	Tuned_Adaboost_on_Imbalanced_Dataset	Imbalanced	{'params.n_estimators': '120', 'params.bal_typ...}	
8	Tuned_Bagging_RF_on_Balanced_Dataset	Balanced	{'params.n_estimators': '18', 'params.bal_type...}	
9	Tuned_Bagging_RF_on_Imbalanced_Dataset	Imbalanced	{'params.n_estimators': '18', 'params.bal_type...}	
10	Tuned_Random_Forest_on_Balanced_Dataset	Balanced	{'params.n_estimators': '83', 'params.max_dept...}	
11	Tuned_Random_Forest_on_Imbalanced_Dataset	Imbalanced	{'params.n_estimators': '83', 'params.max_dept...}	
12	Tuned_Decision_Tree_on_Balanced_Dataset	Balanced	{'params.max_depth': '16', 'params.bal_type': '...}	
13	Tuned_Decision_Tree_on_Imbalanced_Dataset	Imbalanced	{'params.max_depth': '16', 'params.bal_type': '...}	
14	Tuned_KNN_on_Balanced_Dataset	Balanced	{'params.bal_type': 'Balanced', 'params.metric...}	
15	Tuned_KNN_on_Imbalanced_Dataset	Imbalanced	{'params.bal_type': 'Imbalanced', 'params.metr...}	
16	Tuned_MLPClassifier_on_Balanced_Dataset	Balanced	{'params.bal_type': 'Balanced', 'params.learni...}	
17	Tuned_MLPClassifier_on_Imbalanced_Dataset	Imbalanced	{'params.bal_type': 'Imbalanced', 'params.learn...}	
18	Tuned_Logistic_Regression_on_Balanced_Dataset	Balanced	{'params.bal_type': 'Balanced', 'params.solver...}	
19	Tuned_Logistic_Regression_on_Imbalanced_Dataset	Imbalanced	{'params.bal_type': 'Imbalanced', 'params.solv...}	

		run_name	bal_type	params_dict	met
20	Simple_Logistic_Regresion_on_balanced_Dataset		Balanced	{'params.bal_type': 'Balanced', 'params.random...}	
21	Simple_Logistic_Regresion_on_Imbalanced_Dataset		Imbalanced	{'params.bal_type': 'Imbalanced', 'params.rand...}	
22		binary_kmeans_adv	None	{'params.gamma': '0.025', 'params.alpha': '0.0...}	
23		binary_gmm	None	{'params.random_state': '42', 'params.n_compon...}	9.8780
24		binary_knn	None	{'params.n_neighbors': '5', 'params.n_jobs': '...}	
25		binary_dbSCAN	None	{'params.n_jobs': '-1', 'params.eps': '0.5', '...}	
26		binary_one_class_svm	None	{'params.verbose': '0', 'params.nu': '0.1'}	
27		binary_robust_cov	None	{'params.random_state': '42', 'params.contamin...}	
28		binary_iforest	None	{'params.n_estimators': '100', 'params.random...}	
29		binary_lof	None	{'params.n_neighbors': '20', 'params.n_jobs': '...}	
30	Tuned_KNN_on_Imbalanced_Dataset		Imbalanced	{'params.bal_type': 'Imbalanced', 'params.metr...}	
31	Tuned_MLPClassifier_on_Balanced_Dataset		Balanced	{'params.bal_type': 'Balanced', 'params.learni...}	
32	Tuned_MLPClassifier_on_Imbalanced_Dataset		Imbalanced	{'params.bal_type': 'Imbalanced', 'params.lear...}	
33	Tuned_Logistic_Regresion_on_Balanced_Dataset		Balanced	{'params.bal_type': 'Balanced', 'params.solver...}	
34	Tuned_Logistic_Regresion_on_Imbalanced_Dataset		Imbalanced	{'params.bal_type': 'Imbalanced', 'params.solv...}	
35	Simple_Logistic_Regresion_on_balanced_Dataset		Balanced	{'params.bal_type': 'Balanced', 'params.random...}	
36	Simple_Logistic_Regresion_on_Imbalanced_Dataset		Imbalanced	{'params.bal_type': 'Imbalanced', 'params.rand...}	
37		binary_kmeans_adv	None	{'params.gamma': '0.25', 'params.alpha': '0.01...}	
38		binary_gmm	None	{'params.random_state': '42', 'params.n_compon...}	9.8780
39		binary_knn	None	{'params.n_neighbors': '5', 'params.n_jobs': '...}	

	run_name	bal_type	params_dict	met
40	binary_dbSCAN	None	{'params.n_jobs': '-1', 'params.eps': '0.5', ...}	
41	binary_one_class_svm	None	{'params.verbose': '0', 'params.nu': '0.1'}	
42	binary_robust_cov	None	{'params.random_state': '42', 'params.contamin...'}	
43	binary_iforest	None	{'params.n_estimators': '100', 'params.random_...'}	
44	binary_lof	None	{'params.n_neighbors': '20', 'params.n_jobs': ...}	
45	binary_ground_truth_umap	None		{}

## Fill zero inplace of Null values

```
In [ ]: all_logged_metrics_df.fillna(value = 0,inplace=True)
```

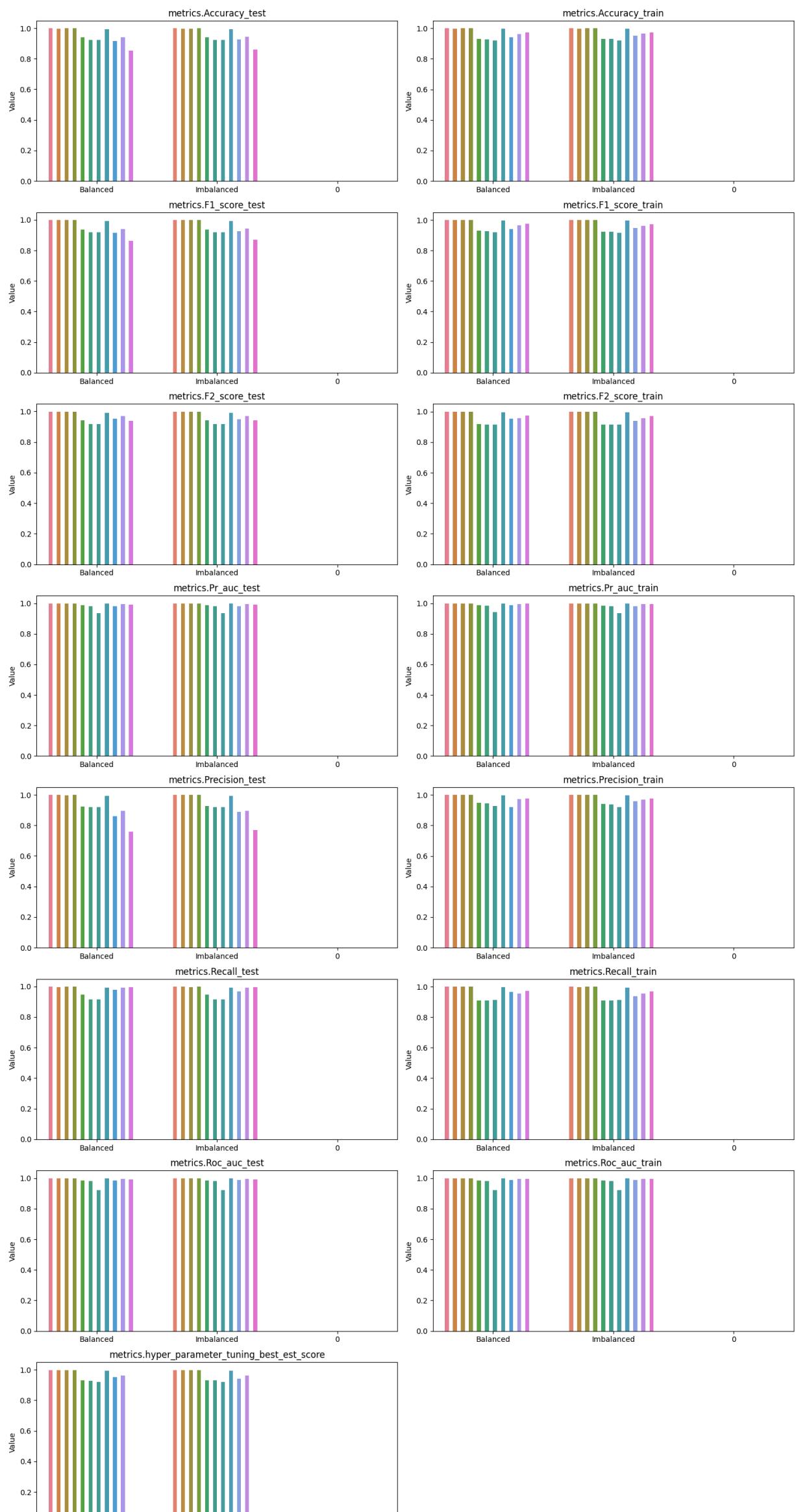
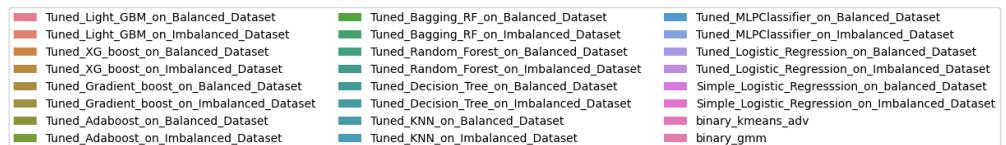
## all\_logged\_metrics\_df\_plots

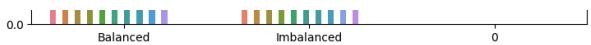
```
In [ ]: # Selecting only classification metrics
df = all_logged_metrics_df.iloc[0:24][['run_name', 'bal_type', 'params_dict',
                                         'metrics.Accuracy_test', 'metrics.Accuracy_train',
                                         'metrics.F1_score_test', 'metrics.F1_score_train',
                                         'metrics.F2_score_test', 'metrics.F2_score_train',
                                         'metrics.Pr_auc_test', 'metrics.Pr_auc_train', 'metrics.Precision_test',
                                         'metrics.Precision_train', 'metrics.Recall_test',
                                         'metrics.Recall_train', 'metrics.Roc_auc_test', 'metrics.Roc_auc_train',
                                         'metrics.hyper_parameter_tuning_best_est_score']]
```

```
In [ ]: # Considering only classification metrics to plot
df.head()
```

	run_name	bal_type	params_dict	metrics.Accura
0	Tuned_Light_GBM_on_Balanced_Dataset	Balanced	{'params.subsample': '1.3631034258755936', 'pa...}	0.
1	Tuned_Light_GBM_on_Imbalanced_Dataset	Imbalanced	{'params.subsample': '1.3631034258755936', 'pa...}	0.
2	Tuned_XG_boost_on_Balanced_Dataset	Balanced	{'params.subsample': '0.890923123379134', 'par...}	0.
3	Tuned_XG_boost_on_Imbalanced_Dataset	Imbalanced	{'params.subsample': '0.890923123379134', 'par...}	0.
4	Tuned_Gradient_boost_on_Balanced_Dataset	Balanced	{'params.subsample': '0.9949692657420364', 'pa...}	0.

```
In [ ]: all_logged_metrics_df_plots(df)
```





## printing best models according to each metric

```
In [ ]: # Assuming your DataFrame is named 'df'  
metrics_columns = df.columns[df.columns.str.startswith('metrics.')]#  
  
for metric in metrics_columns:  
    best_model = df.loc[df[metric].idxmax(), 'run_name']  
    best_params = df.loc[df[metric].idxmax(), 'params_dict']  
    print(f'{metric} -> best_model -> \'{best_model}\', best_params -> {best_params}'")
```

```

metrics.Accuracy_test -> best_model -> "Tuned_Adaboost_on_Imbalanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Imbalanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.Accuracy_train -> best_model -> "Tuned_Adaboost_on_Balanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Balanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.F1_score_test -> best_model -> "Tuned_Adaboost_on_Imbalanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Imbalanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.F1_score_train -> best_model -> "Tuned_Adaboost_on_Balanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Balanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.F2_score_test -> best_model -> "Tuned_Adaboost_on_Imbalanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Imbalanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.F2_score_train -> best_model -> "Tuned_Adaboost_on_Balanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Balanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.Pr_auc_test -> best_model -> "Tuned_Adaboost_on_Imbalanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Imbalanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.Pr_auc_train -> best_model -> "Tuned_Light_GBM_on_Imbalanced_Dataset", best_params -> {'params.subsample': '1.3631034258755936', 'params.feature_fraction': '0.5745506436797708', 'params.min_child_samples': '10', 'params.reg_lambda': '1.0376985928164089', 'params.objective': 'binary', 'params.min_child_weight': '0.020871568153417244', 'params.min_split_gain': '0.08154614284548342', 'params.bagging_freq': '9', 'params.min_data_in_leaf': '82', 'params.n_estimators': '180', 'params.num_leaves': '52', 'params.max_depth': '10', 'params.bal_type': 'Imbalanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.7317381190502594', 'params.bagging_fraction': '0.7809345096873808', 'params.reg_alpha': '0.03702232586704518', 'params.colsample_bytree': '1.3021969807540397', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.3060660809801552'}
metrics.Precision_test -> best_model -> "Tuned_Adaboost_on_Imbalanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Imbalanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.Precision_train -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.3631034258755936', 'params.feature_fraction': '0.5745506436797708', 'params.min_child_samples': '10', 'params.reg_lambda': '1.0376985928164089', 'params.objective': 'binary', 'params.min_child_weight': '0.020871568153417244', 'params.min_split_gain': '0.08154614284548342', 'params.bagging_freq': '9', 'params.min_data_in_leaf': '82', 'params.n_estimators': '180', 'params.num_leaves': '52', 'params.max_depth': '10', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.7317381190502594', 'params.bagging_fraction': '0.7809345096873808', 'params.reg_alpha': '0.03702232586704518', 'params.colsample_bytree': '1.3021969807540397', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.3060660809801552'}
metrics.Recall_test -> best_model -> "Tuned_Adaboost_on_Balanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Balanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.Recall_train -> best_model -> "Tuned_Adaboost_on_Balanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Balanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.Roc_auc_test -> best_model -> "Tuned_Adaboost_on_Imbalanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Imbalanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
metrics.Roc_auc_train -> best_model -> "Tuned_Light_GBM_on_Imbalanced_Dataset", best_params -> {'params.subsample': '1.3631034258755936', 'params.feature_fraction': '0.5745506436797708', 'params.min_child_samples': '10', 'params.reg_lambda': '1.0376985928164089', 'params.objective': 'binary', 'params.min_child_weight': '0.020871568153417244', 'params.min_split_gain': '0.08154614284548342', 'params.bagging_freq': '9', 'params.min_data_in_leaf': '82', 'params.n_estimators': '180', 'params.num_leaves': '52', 'params.max_depth': '10', 'params.bal_type': 'Imbalanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.7317381190502594', 'params.bagging_fraction': '0.7809345096873808', 'params.reg_alpha': '0.03702232586704518', 'params.colsample_bytree': '1.3021969807540397', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.3060660809801552'}
metrics.hyper_parameter_tuning_best_est_score -> best_model -> "Tuned_Adaboost_on_Balanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Imbalanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}

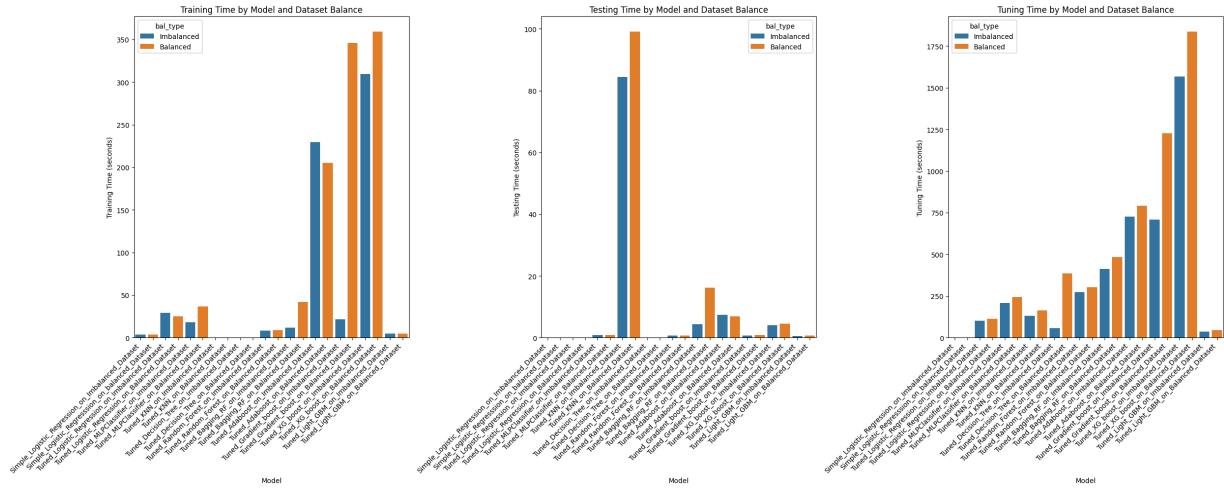
```

## Observation

Tuned\_Adaboost\_on\_Balanced\_Dataset Model provides best metrics. So we consider that model for deployment of Binary class Classification.

## TIME\_DF AND ITS PLOT

In [ ]: `time_df_plots(time_df)`



### Observation

- Training time is highest for `Tuned_XG_boost_on_Imbalanced_Dataset`
- Testing time is highest for `Tuned_KNN_on_Balanced_Dataset`
- Tuning time is highest for `'Tuned_XG_boost_on_Imbalanced_Dataset'`

we should avoid these models.

- Testing time and Accuracy is important to select the best model.
- Testing time is less for 'Logistic Regression, Decision Tree, Random Forest.'
- Boosting models like Adaboost, Gradient Boosting have little higher testing time around 5 sec. But Boosting models provide best metrics. So We use Adaboost model to deploy.

## FEATURE IMPORTANCE DF

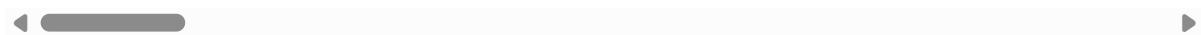
In [ ]: `feature_importance_df`

Out[ ]:

Simple\_Logistic\_Regression\_on\_Imbalanced\_Dataset Simple\_Logistic\_R

duration	-0.355736
srcbytes	0.480815
dstbytes	0.753895
wrongfragment	2.362582
urgent	0.504681
...	...
binary_one_class_svm_df	0.427894
binary_dbSCAN_labels	-0.032274
binary_knn_kth_distance	-0.803444
binary_gmm_score	1.027387
binary_kmeans_adv	2.282959

62 rows × 16 columns



In [ ]:

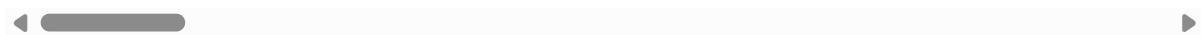
```
# normalizing the values
scaler = StandardScaler()
feature_importance_scaled = pd.DataFrame(scaler.fit_transform(feature_importance_df),
                                           index=feature_importance_df.index,
                                           columns=feature_importance_df.columns)
feature_importance_scaled
```

Out[ ]:

Simple\_Logistic\_Regression\_on\_Imbalanced\_Dataset Simple\_Logistic\_R

duration	-0.288604
srcbytes	0.241661
dstbytes	0.414758
wrongfragment	1.434457
urgent	0.256789
...	...
binary_one_class_svm_df	0.208116
binary_dbSCAN_labels	-0.083570
binary_knn_kth_distance	-0.572393
binary_gmm_score	0.588117
binary_kmeans_adv	1.383986

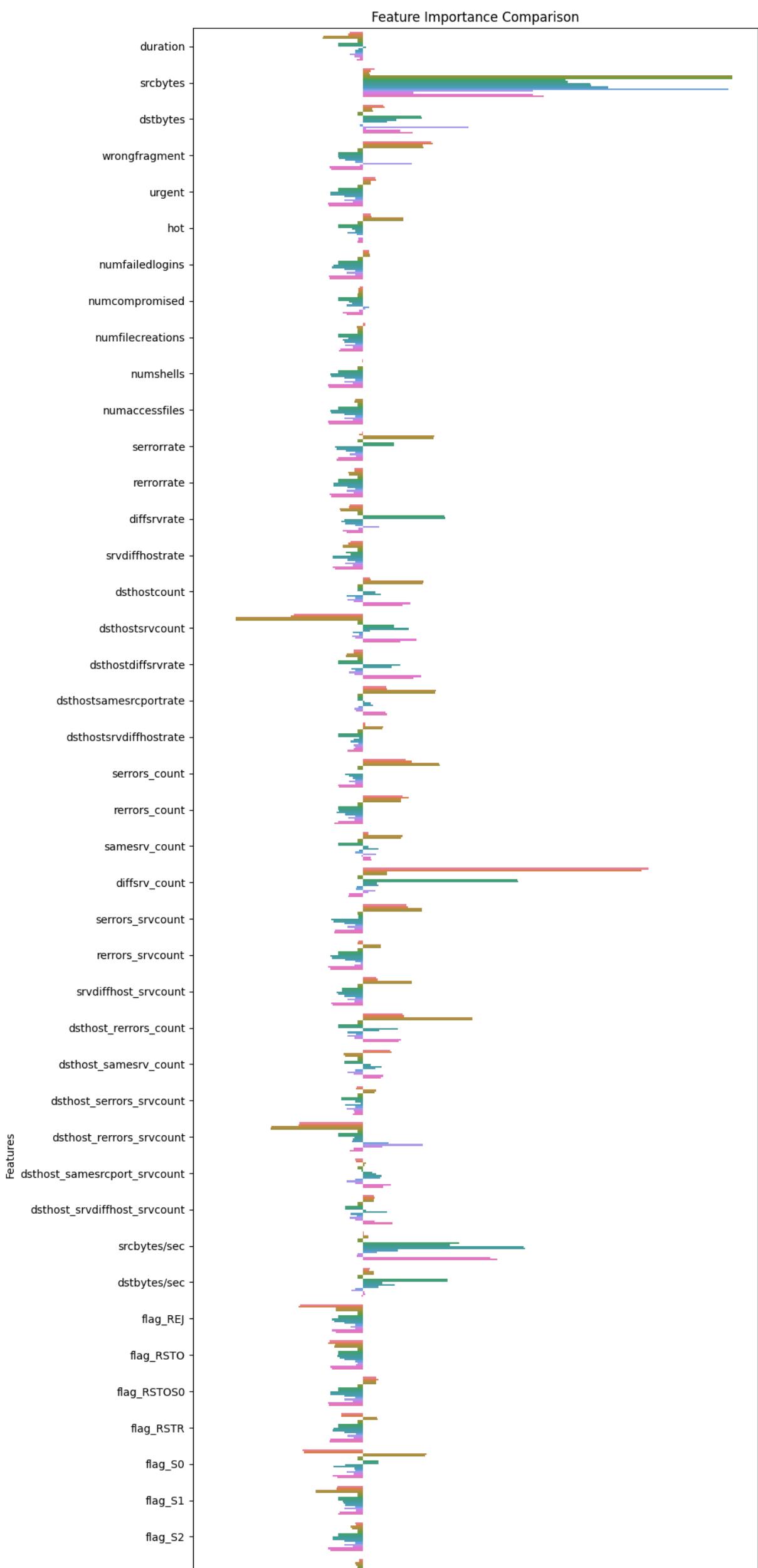
62 rows × 16 columns

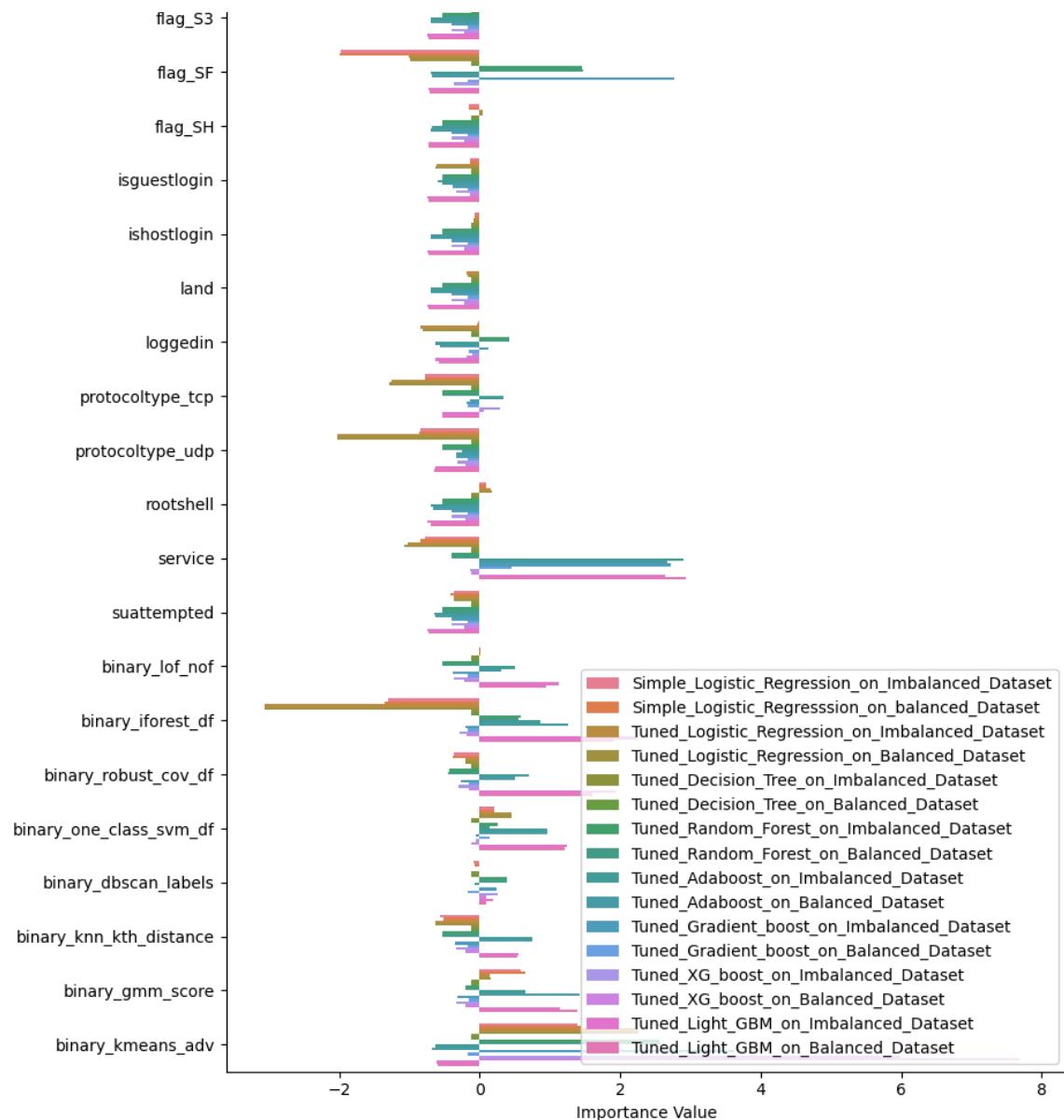


## Feature\_importance\_df\_plots

In [ ]:

```
feature_importance_plots(feature_importance_scaled)
```





## sum of all the feature\_importances and normalizing

```
In [ ]: combined_feature_importance_scaled = feature_importance_scaled.sum(axis = 1)
```

```
In [ ]: combined_feature_importance_scaled = pd.DataFrame(combined_feature_importance_scaled)
```

```
In [ ]: combined_feature_importance_scaled
```

```
Out[ ]: combined_feature_importances
```

	duration
srcbytes	-4.676412
dstbytes	59.519717
wrongfragment	8.715336
urgent	2.060311
...	-4.547147
...	...
binary_one_class_svm_df	0.950933
binary_dbSCAN_labels	5.727848
binary_knn_kth_distance	-2.177541
binary_gmm_score	4.434210
binary_kmeans_adv	26.823593

62 rows × 1 columns

```
In [ ]: scaler = StandardScaler()
combined_feature_importance_scaled = pd.DataFrame(scaler.fit_transform(combined_feat
```

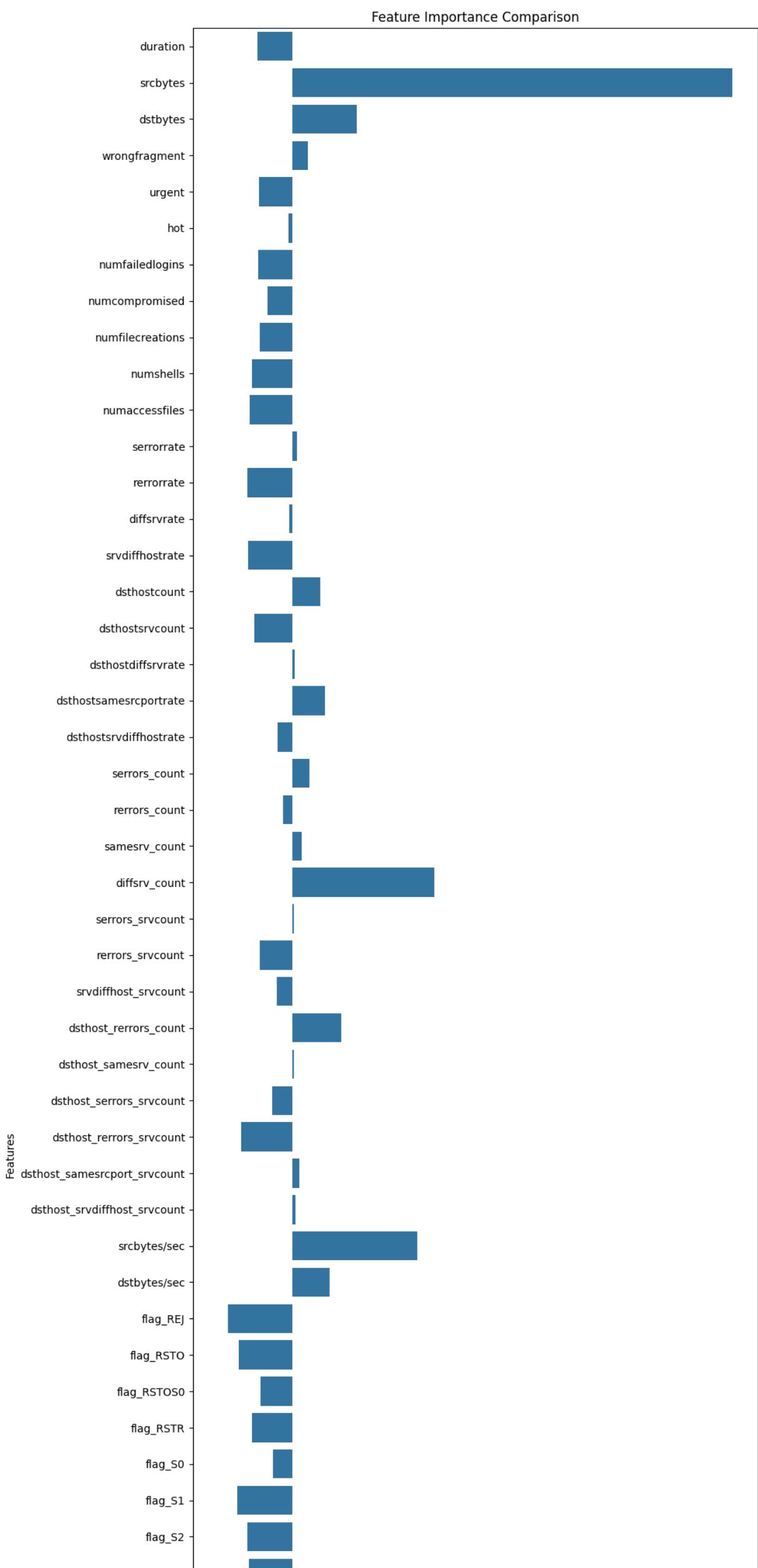
```
index=combined_feature_importance_scaled.in  
columns = combined_feature_importance_scale  
combined_feature_importance_scaled
```

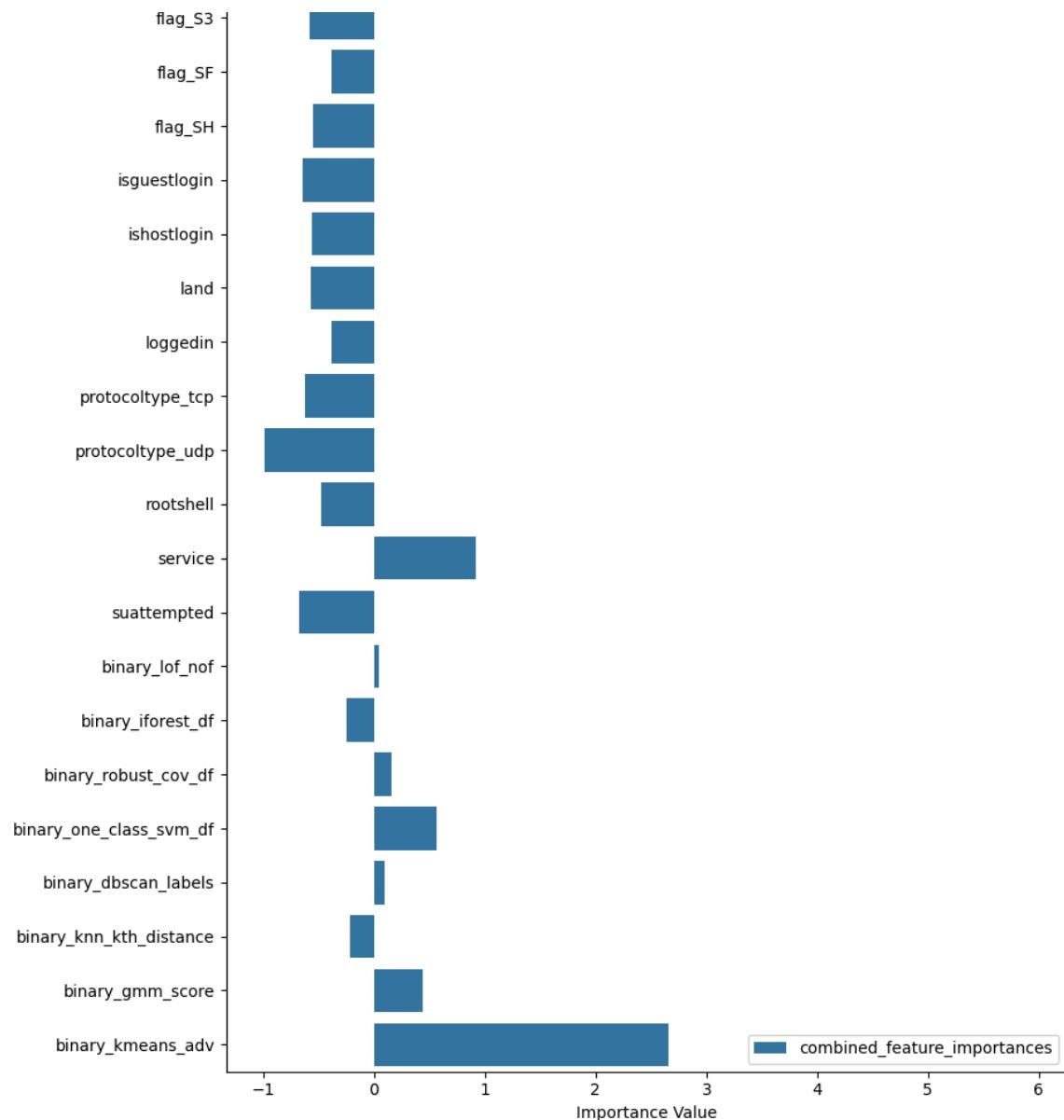
Out[ ]:

combined_feature_importances	
<b>duration</b>	-0.464125
<b>srcbytes</b>	5.907225
<b>dstbytes</b>	0.864981
<b>wrongfragment</b>	0.204482
<b>urgent</b>	-0.451296
...	...
<b>binary_one_class_svm_df</b>	0.568479
<b>binary_dbSCAN_labels</b>	0.094378
<b>binary_knn_kth_distance</b>	-0.216117
<b>binary_gmm_score</b>	0.440087
<b>binary_kmeans_adv</b>	2.662193

62 rows × 1 columns

In [ ]: `feature_importance_plots(combined_feature_importance_scaled)`





```
In [ ]: combined_feature_importance_scaled[np.abs(combined_feature_importance_scaled["combin"]]
```

```
Out[ ]: combined_feature_importances
```

<b>srcbytes</b>	5.907225
<b>binary_kmeans_adv</b>	2.662193
<b>diffsrv_count</b>	1.910609
<b>srcbytes/sec</b>	1.682608
<b>service</b>	0.920829
<b>dstbytes</b>	0.864981
<b>flag_REJ</b>	-0.868855
<b>protocoltype_udp</b>	-0.992583

### Observation

Above dataframe shows the top 10 features according combine normalised feature\_importances

## PLOT ALL LEARNING CURVES

```
In [ ]: def get_experiment_id(experiment_name):
    experiment = mlflow.get_experiment_by_name(experiment_name)
    if experiment:
        return experiment.experiment_id
    else:
        print(f"Experiment '{experiment_name}' not found.")
        return None
```

```
def get_run_ids(experiment_id):
    client = mlflow.tracking.MlflowClient()
    runs = client.search_runs(experiment_id)
    return [run.info.run_id for run in runs]

# Example usage
experiment_id = get_experiment_id("nadp_binary")
run_ids = get_run_ids(experiment_id)
```

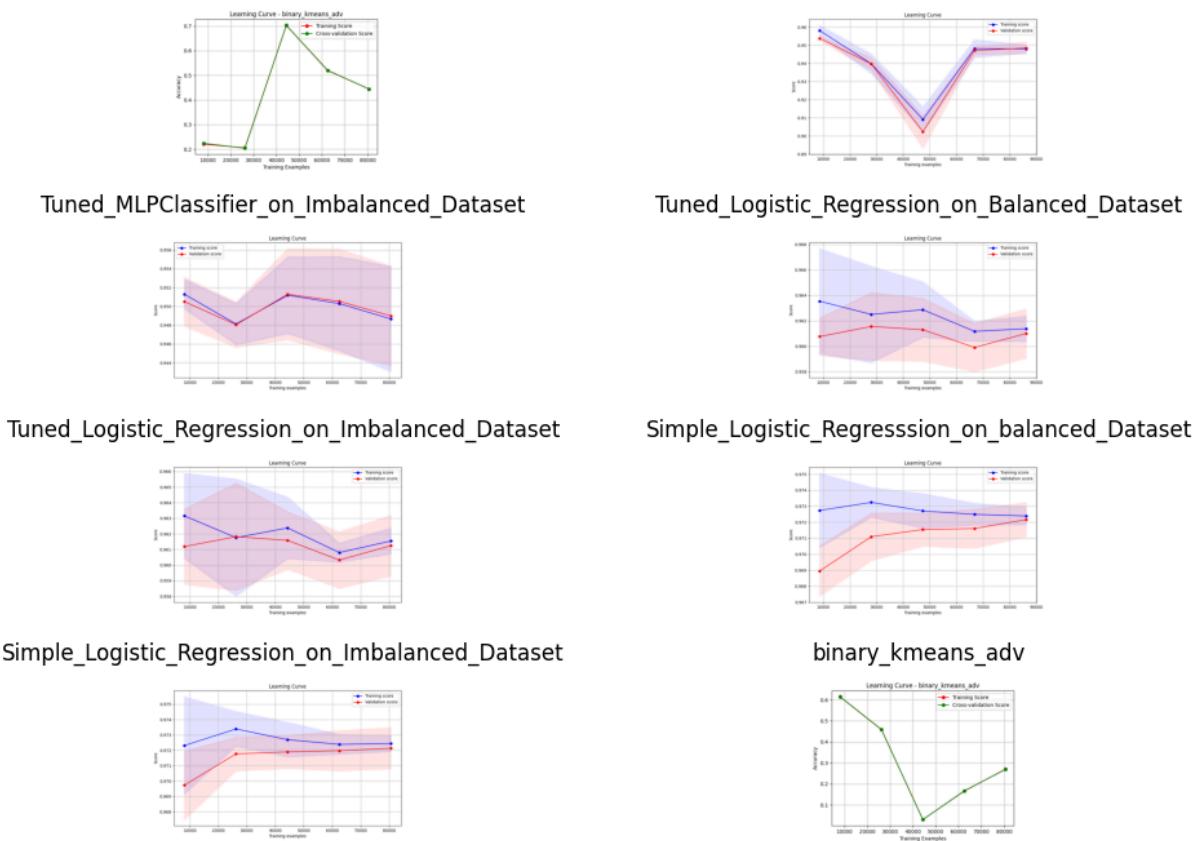
```
In [ ]: def plot_all_learning_curves(experiment_id, run_ids):
    plt.figure(figsize=(10, 40))
    plot_count = 1

    for run_id in run_ids:
        run_path = f"mlruns/{experiment_id}/{run_id}/artifacts/"
        for file in os.listdir(run_path):
            if file.endswith("_learning_curve.png"):
                img = plt.imread(os.path.join(run_path, file))
                plt.subplot(len(run_ids)//2, 2, plot_count)
                plt.imshow(img)
                plt.axis('off')
                plt.title(f"{file.replace('_learning_curve.png', '')}")
                plot_count += 1

    plt.tight_layout()
    plt.show()

# Example usage
plot_all_learning_curves(experiment_id, run_ids)
```





### Observation

- tuned LightGBM on both balanced and imbalanced datasets shows a consistent learning curve with training and validation scores improving steadily as training size increases.
- tuned XGBoost displays a similar pattern, but the validation score tends to plateau sooner compared to the training score, indicating a potential risk of overfitting.
- gradient boosting models show smoother learning curves for both balanced and imbalanced datasets, but the gap between training and validation scores suggests some overfitting on imbalanced data.
- the tuned Adaboost model exhibits steady growth in the learning curves, with balanced datasets showing a closer match between training and validation scores.
- bagging models, such as random forest and bagging RF, present varied performance, with balanced datasets achieving better validation alignment compared to imbalanced data.
- decision tree models have more erratic learning curves, with training scores significantly higher than validation scores, indicating overfitting.
- KNN models show stable performance on both balanced and imbalanced datasets, though they do not reach as high accuracy as ensemble models.
- the tuned MLP classifier shows fluctuating learning curves with occasional dips, but on the balanced dataset, it maintains closer alignment between training and validation.
- logistic regression learning curves display consistent performance, with simple logistic regression on balanced data achieving better generalization.
- the binary k-means learning curve shows a unique pattern with training scores being significantly higher and validation scores decreasing sharply, indicating potential issues with the model's ability to generalize.

Let me know if you need further details or analysis on these observations!

## PLOT ALL AUC PLOTS

```
In [ ]: def plot_all_roc_auc_plots(experiment_id, run_ids):
    plt.figure(figsize=(10, 80))
    plot_count = 1

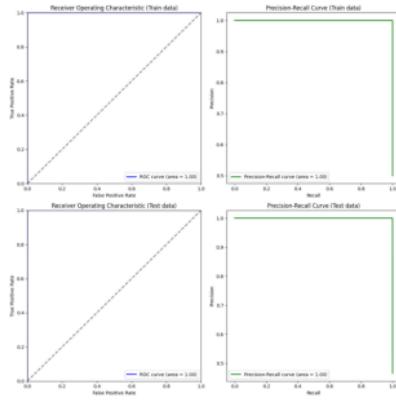
    for run_id in run_ids:
        run_path = f"mlruns/{experiment_id}/{run_id}/artifacts/"
        for file in os.listdir(run_path):
            if file.endswith("_auc_plots.png"):
```

```
    img = plt.imread(os.path.join(run_path, file))
    plt.subplot(len(run_ids)//2, 2, plot_count)
    plt.imshow(img)
    plt.axis('off')
    plt.title(f"{file.replace('_auc_plots.png', '')}")
    plot_count += 1

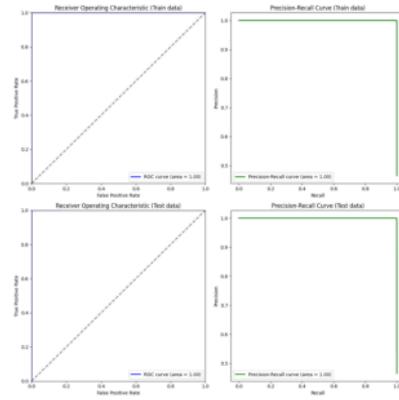
plt.tight_layout()
plt.show()

# Example usage
plot_all_roc_auc_plots(experiment_id, run_ids)
```

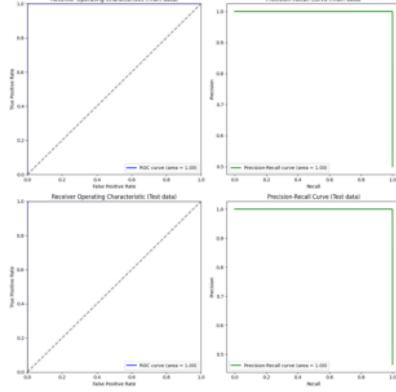
Tuned\_Light\_GBM\_on\_Balanced\_Dataset



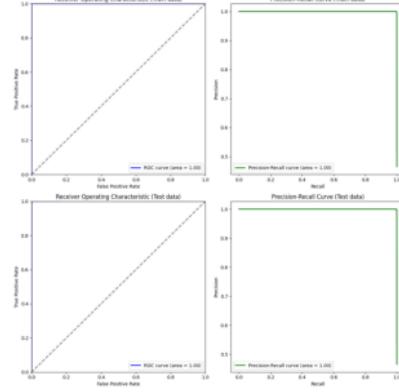
Tuned\_Light\_GBM\_on\_Imbalanced\_Dataset



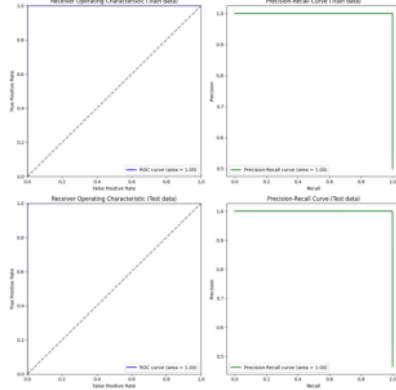
Tuned\_XG\_boost\_on\_Balanced\_Dataset



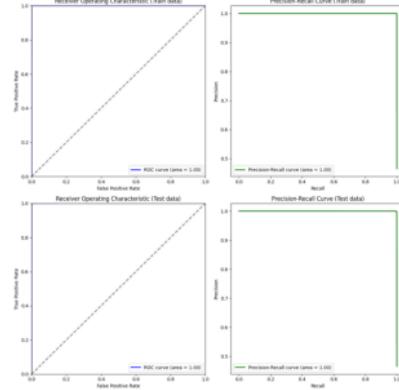
Tuned\_XG\_boost\_on\_Imbalanced\_Dataset



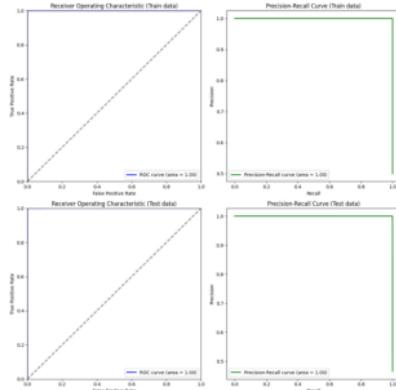
Tuned\_Gradient\_boost\_on\_Balanced\_Dataset



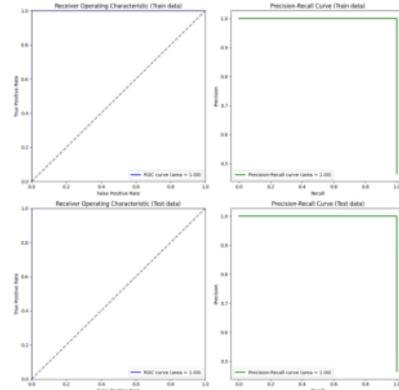
Tuned\_Gradient\_boost\_on\_Imbalanced\_Dataset



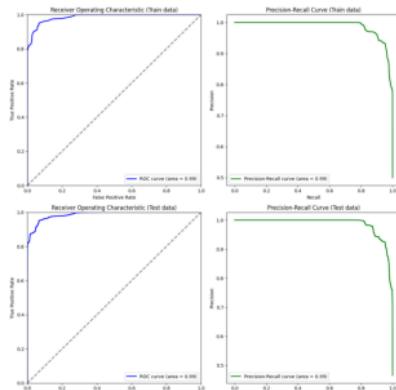
Tuned\_Adaboost\_on\_Balanced\_Dataset



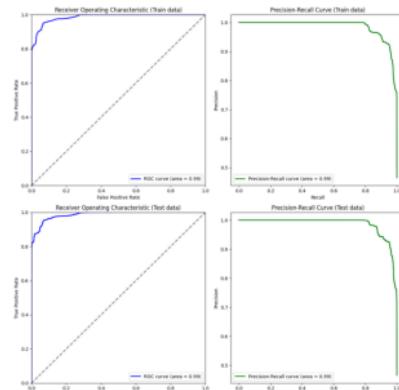
Tuned\_Adaboost\_on\_Imbalanced\_Dataset



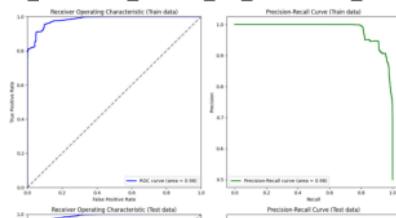
Tuned\_Bagging\_RF\_on\_Balanced\_Dataset



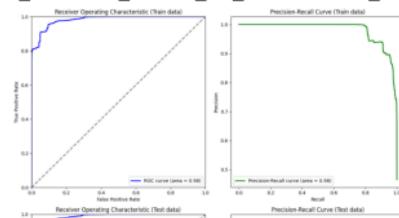
Tuned\_Bagging\_RF\_on\_Imbalanced\_Dataset

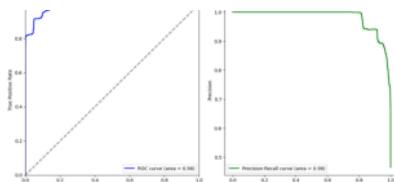


Tuned\_Random\_Forest\_on\_Balanced\_Dataset

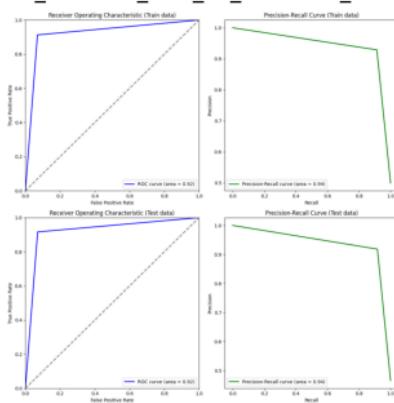


Tuned\_Random\_Forest\_on\_Imbalanced\_Dataset

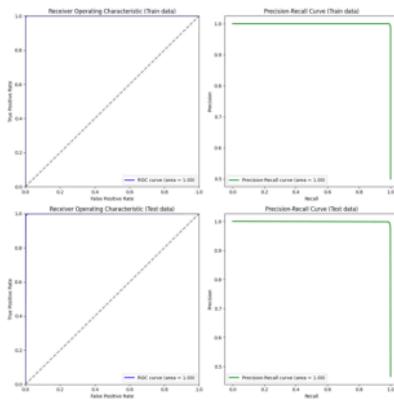




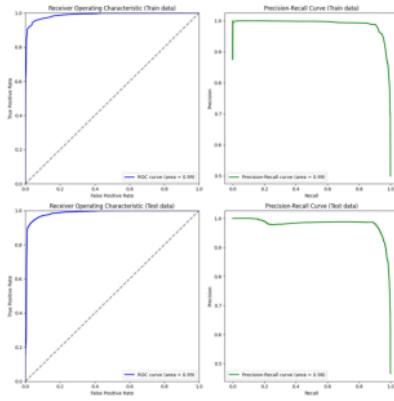
Tuned\_Dcision\_Tree\_on\_Balanced\_Dataset



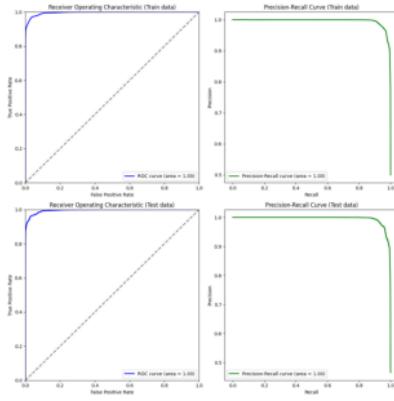
Tuned\_KNN\_on\_Balanced\_Dataset



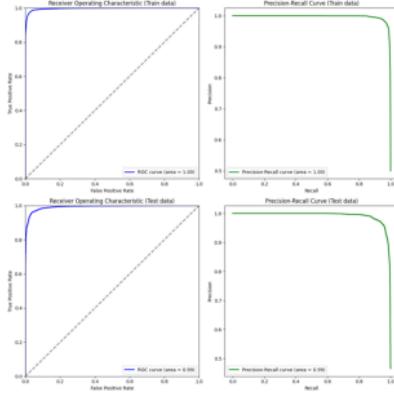
Tuned\_MLPClassifier\_on\_Balanced\_Dataset



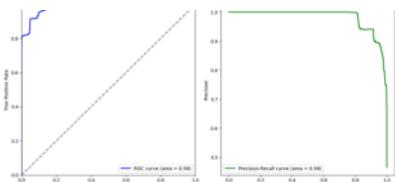
Tuned\_Logistic\_Regression\_on\_Balanced\_Dataset



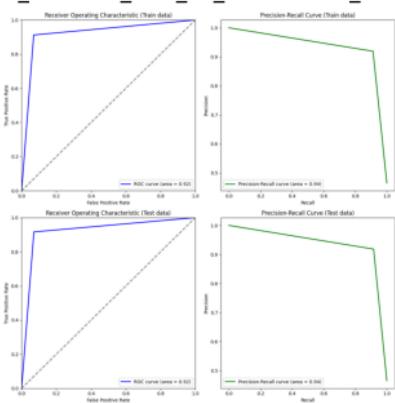
Simple\_Logistic\_Regression\_on\_balanced\_Dataset



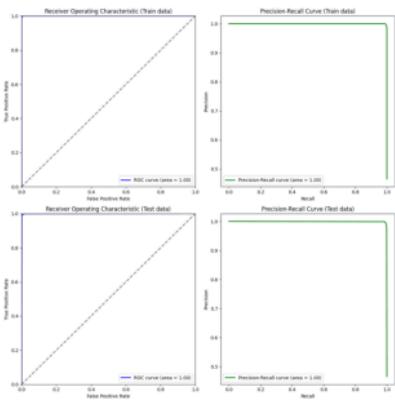
Tuned\_KNN\_on\_Imbalanced\_Dataset



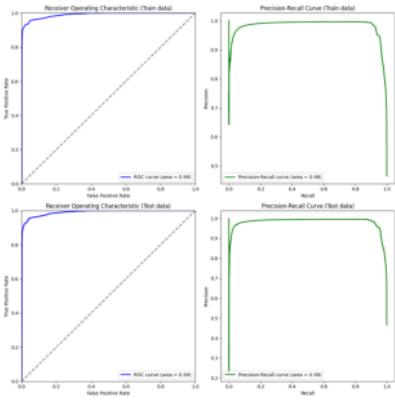
Tuned\_Dcision\_Tree\_on\_Imbalanced\_Dataset



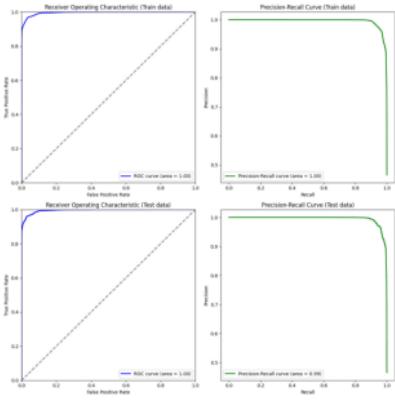
Tuned\_KNN\_on\_Imbalanced\_Dataset



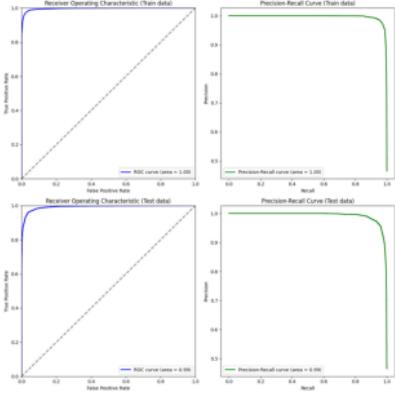
Tuned\_MLPClassifier\_on\_Imbalanced\_Dataset



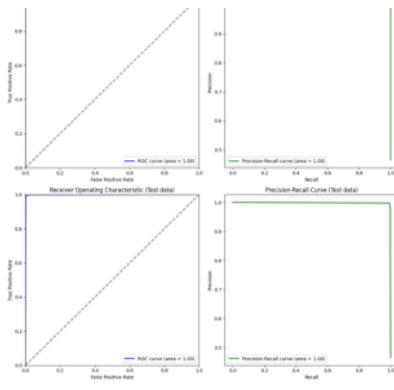
Tuned\_Logistic\_Regression\_on\_Imbalanced\_Dataset



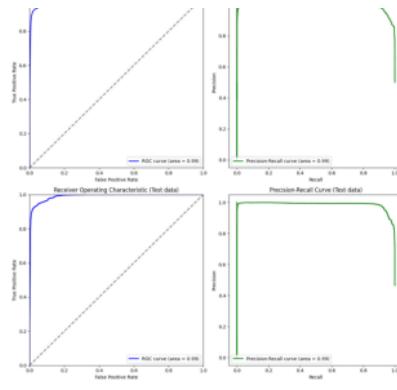
Simple\_Logistic\_Regression\_on\_Imbalanced\_Dataset



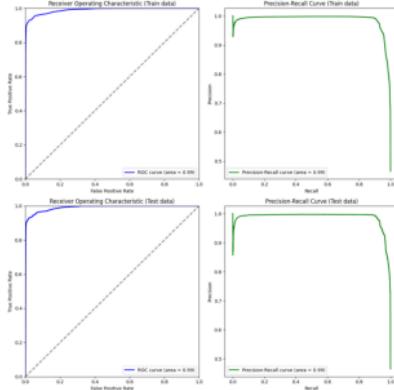
Tuned\_MLPClassifier\_on\_Balanced\_Dataset



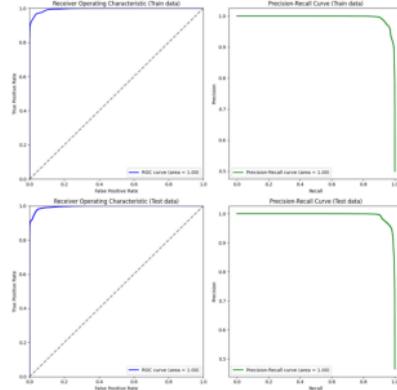
Tuned\_MLPClassifier\_on\_Imbalanced\_Dataset



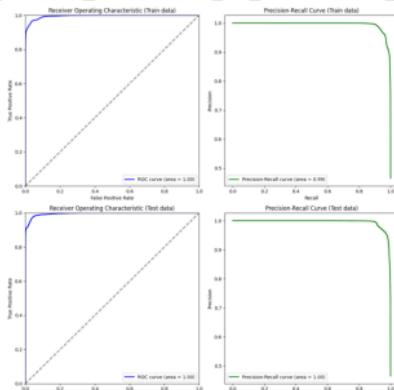
Tuned\_Logistic\_Regression\_on\_Balanced\_Dataset



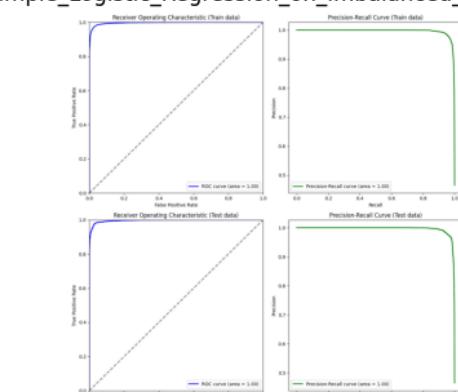
Tuned\_Logistic\_Regression\_on\_Imbalanced\_Dataset



Simple\_Logistic\_Regression\_on\_balanced\_Dataset



Simple\_Logistic\_Regression\_on\_Imbalanced\_Dataset



## Observation

- The ROC curves for tuned LightGBM models on both balanced and imbalanced datasets demonstrate high AUC scores, indicating strong classifier performance.
- Tuned XGBoost models achieve a similar high AUC with curves that approach the top left corner, suggesting excellent discrimination between classes.
- Gradient boosting models for both balanced and imbalanced datasets show robust ROC curves, signifying good predictive power.
- Tuned Adaboost exhibits a solid performance with AUC values close to 1, although slight differences in curve shape suggest variations in dataset handling.
- Bagging RF models present nearly perfect ROC curves, particularly on the balanced dataset, highlighting their effectiveness in classification.
- Random forest models also display consistently high AUC values, with the balanced dataset having a slightly better ROC curve than the imbalanced one.
- Decision tree models show high AUC scores, but the imbalance in dataset handling impacts the consistency of the curves.

- KNN models achieve good AUC, though the curves on balanced data are slightly less steep, suggesting room for improvement in class separation.
- The MLP classifier demonstrates excellent ROC curves on both balanced and imbalanced datasets, reflecting strong generalization capabilities.
- Tuned logistic regression models display reliable AUC scores, with consistent curves that show slight differences between balanced and imbalanced datasets.
- Simple logistic regression has high AUC, with curves that suggest stable performance across both balanced and imbalanced datasets.

## PRINT ALL CLASSIFICATION REPORTS

```
In [ ]: def print_all_classification_reports(experiment_id, run_ids):
    for run_id in run_ids:
        run_path = f"mlruns/{experiment_id}/{run_id}/artifacts/"
        for file in os.listdir(run_path):
            if file.endswith("_classification_report.csv"):
                report_df = pd.read_csv(os.path.join(run_path, file), index_col=0)
                print(f"\n{file.replace('_classification_report.csv', '')}:\n")
                print(report_df)
                print("\n" + "-"*80 + "\n")

# Example usage
print_all_classification_reports(experiment_id, run_ids)
```

Tuned\_Light\_GBM\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.998295	0.999555	0.998924	13469.000000
1	0.999487	0.998038	0.998762	11724.000000
accuracy	0.998849	0.998849	0.998849	0.998849
macro avg	0.998891	0.998796	0.998843	25193.000000
weighted avg	0.998850	0.998849	0.998849	25193.000000

---

Tuned\_Light\_GBM\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.999981	1.000000	0.999991	53870.000000
1	1.000000	0.999981	0.999991	53870.000000
accuracy	0.999991	0.999991	0.999991	0.999991
macro avg	0.999991	0.999991	0.999991	107740.000000
weighted avg	0.999991	0.999991	0.999991	107740.000000

---

Tuned\_Light\_GBM\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.998295	0.999555	0.998924	13469.000000
1	0.999487	0.998038	0.998762	11724.000000
accuracy	0.998849	0.998849	0.998849	0.998849
macro avg	0.998891	0.998796	0.998843	25193.000000
weighted avg	0.998850	0.998849	0.998849	25193.000000

---

Tuned\_Light\_GBM\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.999963	1.000000	0.999981	53870.000000
1	1.000000	0.999957	0.999979	46897.000000
accuracy	0.999980	0.999980	0.999980	0.99998
macro avg	0.999981	0.999979	0.999980	100767.000000
weighted avg	0.999980	0.999980	0.999980	100767.000000

---

Tuned\_XG\_boost\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.996964	0.999555	0.998258	13469.000000
1	0.999487	0.996503	0.997993	11724.000000
accuracy	0.998134	0.998134	0.998134	0.998134
macro avg	0.998225	0.998029	0.998125	25193.000000
weighted avg	0.998138	0.998134	0.998134	25193.000000

---

Tuned\_XG\_boost\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.998406	0.999759	0.999082	53870.000000
1	0.999758	0.998404	0.999080	53870.000000
accuracy	0.999081	0.999081	0.999081	0.999081
macro avg	0.999082	0.999081	0.999081	107740.000000
weighted avg	0.999082	0.999081	0.999081	107740.000000

---

Tuned\_XG\_boost\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.997555	0.999555	0.998554	13469.000000
1	0.999487	0.997185	0.998335	11724.000000
accuracy	0.998452	0.998452	0.998452	0.998452
macro avg	0.998521	0.998370	0.998444	25193.000000
weighted avg	0.998454	0.998452	0.998452	25193.000000

---

Tuned\_XG\_boost\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.998517	0.999759	0.999137	53870.000000
1	0.999722	0.998294	0.999008	46897.000000
accuracy	0.999077	0.999077	0.999077	0.999077
macro avg	0.999120	0.999026	0.999073	100767.000000
weighted avg	0.999078	0.999077	0.999077	100767.000000

---

Tuned\_Gradient\_boost\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.998812	0.998441	0.998626	13469.000000
1	0.998210	0.998635	0.998422	11724.000000
accuracy	0.998531	0.998531	0.998531	0.998531
macro avg	0.998511	0.998538	0.998524	25193.000000
weighted avg	0.998531	0.998531	0.998531	25193.000000

---

Tuned\_Gradient\_boost\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.999629	0.999870	0.999749	53870.000000
1	0.999870	0.999629	0.999749	53870.000000
accuracy	0.999749	0.999749	0.999749	0.999749
macro avg	0.999749	0.999749	0.999749	107740.000000
weighted avg	0.999749	0.999749	0.999749	107740.000000

---

Tuned\_Gradient\_boost\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.997481	0.999406	0.998442	13469.000000
1	0.999316	0.997100	0.998207	11724.000000
accuracy	0.998333	0.998333	0.998333	0.998333
macro avg	0.998398	0.998253	0.998325	25193.000000
weighted avg	0.998335	0.998333	0.998333	25193.000000

---

Tuned\_Gradient\_boost\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.999889	1.000000	0.999944	53870.000000
1	1.000000	0.999872	0.999936	46897.000000
accuracy	0.999940	0.999940	0.999940	0.99994
macro avg	0.999944	0.999936	0.999940	100767.000000
weighted avg	0.999940	0.999940	0.999940	100767.000000

---

Tuned\_Adaboost\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.998961	0.999480	0.999221	13469.000000

1	0.999403	0.998806	0.999104	11724.000000
accuracy	0.999166	0.999166	0.999166	0.999166
macro avg	0.999182	0.999143	0.999162	25193.000000
weighted avg	0.999167	0.999166	0.999166	25193.000000

---

Tuned\_Adaboost\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	1.0	1.0	1.0	53870.0
1	1.0	1.0	1.0	53870.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	107740.0
weighted avg	1.0	1.0	1.0	107740.0

---

Tuned\_Adaboost\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.998961	0.999555	0.999258	13469.000000
1	0.999488	0.998806	0.999147	11724.000000
accuracy	0.999206	0.999206	0.999206	0.999206
macro avg	0.999225	0.999180	0.999202	25193.000000
weighted avg	0.999206	0.999206	0.999206	25193.000000

---

Tuned\_Adaboost\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	1.0	1.0	1.0	53870.0
1	1.0	1.0	1.0	46897.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	100767.0
weighted avg	1.0	1.0	1.0	100767.0

---

Tuned\_Bagging\_RF\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.953766	0.932734	0.943133	13469.000000
1	0.924632	0.948055	0.936197	11724.000000
accuracy	0.939864	0.939864	0.939864	0.939864
macro avg	0.939199	0.940395	0.939665	25193.000000
weighted avg	0.940208	0.939864	0.939905	25193.000000

---

Tuned\_Bagging\_RF\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.913799	0.949490	0.931303	53870.000000
1	0.947437	0.910433	0.928566	53870.000000
accuracy	0.929961	0.929961	0.929961	0.929961
macro avg	0.930618	0.929961	0.929934	107740.000000
weighted avg	0.930618	0.929961	0.929934	107740.000000

---

Tuned\_Bagging\_RF\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.953363	0.933403	0.943277	13469.000000
1	0.925287	0.947544	0.936283	11724.000000
accuracy	0.939983	0.939983	0.939983	0.939983

```
macro avg      0.939325  0.940473  0.939780  25193.000000
weighted avg   0.940298  0.939983  0.940022  25193.000000
```

---

Tuned\_Bagging\_RF\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.923772	0.949322	0.936373	53870.000000
1	0.939877	0.910016	0.924705	46897.000000
accuracy	0.931029	0.931029	0.931029	0.931029
macro avg	0.931824	0.929669	0.930539	100767.000000
weighted avg	0.931267	0.931029	0.930943	100767.000000

---

Tuned\_Random\_Forest\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.927150	0.930730	0.928937	13469.000000
1	0.920065	0.915984	0.918020	11724.000000
accuracy	0.923868	0.923868	0.923868	0.923868
macro avg	0.923608	0.923357	0.923478	25193.000000
weighted avg	0.923853	0.923868	0.923856	25193.000000

---

Tuned\_Random\_Forest\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.911791	0.948858	0.929955	53870.000000
1	0.946691	0.908205	0.927049	53870.000000
accuracy	0.928532	0.928532	0.928532	0.928532
macro avg	0.929241	0.928532	0.928502	107740.000000
weighted avg	0.929241	0.928532	0.928502	107740.000000

---

Tuned\_Random\_Forest\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.927294	0.931769	0.929526	13469.000000
1	0.921177	0.916070	0.918616	11724.000000
accuracy	0.924463	0.924463	0.924463	0.924463
macro avg	0.924235	0.923919	0.924071	25193.000000
weighted avg	0.924447	0.924463	0.924449	25193.000000

---

Tuned\_Random\_Forest\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.921705	0.949081	0.935193	53870.000000
1	0.939444	0.907393	0.923140	46897.000000
accuracy	0.929679	0.929679	0.929679	0.929679
macro avg	0.930575	0.928237	0.929167	100767.000000
weighted avg	0.929961	0.929679	0.929584	100767.000000

---

Tuned\_Decision\_Tree\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.927306	0.929096	0.928201	13469.000000
1	0.918362	0.916325	0.917343	11724.000000
accuracy	0.923153	0.923153	0.923153	0.923153
macro avg	0.922834	0.922711	0.922772	25193.000000
weighted avg	0.923144	0.923153	0.923148	25193.000000

---

Tuned\_Decision\_Tree\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.914286	0.930239	0.922194	53870.00000
1	0.929001	0.912790	0.920824	53870.00000
accuracy	0.921515	0.921515	0.921515	0.921515
macro avg	0.921643	0.921515	0.921509	107740.00000
weighted avg	0.921643	0.921515	0.921509	107740.00000

---

Tuned\_Decision\_Tree\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.927306	0.929096	0.928201	13469.00000
1	0.918362	0.916325	0.917343	11724.00000
accuracy	0.923153	0.923153	0.923153	0.923153
macro avg	0.922834	0.922711	0.922772	25193.00000
weighted avg	0.923144	0.923153	0.923148	25193.00000

---

Tuned\_Decision\_Tree\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.924218	0.930239	0.927219	53870.00000
1	0.919263	0.912382	0.915810	46897.00000
accuracy	0.921929	0.921929	0.921929	0.921929
macro avg	0.921740	0.921311	0.921514	100767.00000
weighted avg	0.921912	0.921929	0.921909	100767.00000

---

Tuned\_KNN\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.993476	0.994951	0.994213	13469.00000
1	0.994190	0.992494	0.993341	11724.00000
accuracy	0.993808	0.993808	0.993808	0.993808
macro avg	0.993833	0.993723	0.993777	25193.00000
weighted avg	0.993808	0.993808	0.993807	25193.00000

---

Tuned\_KNN\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.995792	0.997253	0.996522	53870.00000
1	0.997249	0.995786	0.996517	53870.00000
accuracy	0.996519	0.996519	0.996519	0.996519
macro avg	0.996520	0.996519	0.996519	107740.00000
weighted avg	0.996520	0.996519	0.996519	107740.00000

---

Tuned\_KNN\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.992967	0.995768	0.994365	13469.00000
1	0.995122	0.991897	0.993507	11724.00000
accuracy	0.993967	0.993967	0.993967	0.993967
macro avg	0.994044	0.993833	0.993936	25193.00000
weighted avg	0.993970	0.993967	0.993966	25193.00000

---

Tuned\_KNN\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.995498	0.997513	0.996504	53870.00000
1	0.997136	0.994818	0.995976	46897.00000
accuracy	0.996259	0.996259	0.996259	0.996259
macro avg	0.996317	0.996165	0.996240	100767.00000
weighted avg	0.996260	0.996259	0.996258	100767.00000

---

Tuned\_MLPClassifier\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.977763	0.861831	0.916144	13469.00000
1	0.860296	0.977482	0.915153	11724.00000
accuracy	0.915651	0.915651	0.915651	0.915651
macro avg	0.919029	0.919656	0.915648	25193.00000
weighted avg	0.923097	0.915651	0.915683	25193.00000

---

Tuned\_MLPClassifier\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.961571	0.916893	0.938700	53870.00000
1	0.920583	0.963356	0.941484	53870.00000
accuracy	0.940124	0.940124	0.940124	0.940124
macro avg	0.941077	0.940124	0.940092	107740.00000
weighted avg	0.941077	0.940124	0.940092	107740.00000

---

Tuned\_MLPClassifier\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.968820	0.892791	0.929253	13469.00000
1	0.887020	0.966991	0.925281	11724.00000
accuracy	0.927321	0.927321	0.927321	0.927321
macro avg	0.927920	0.929891	0.927267	25193.00000
weighted avg	0.930753	0.927321	0.927404	25193.00000

---

Tuned\_MLPClassifier\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.945126	0.966215	0.955554	53870.00000
1	0.960171	0.935561	0.947706	46897.00000
accuracy	0.951949	0.951949	0.951949	0.951949
macro avg	0.952649	0.950888	0.951630	100767.00000
weighted avg	0.952128	0.951949	0.951902	100767.00000

---

Tuned\_Logistic\_Regression\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.992865	0.898805	0.943496	13469.00000
1	0.895154	0.992579	0.941353	11724.00000
accuracy	0.942444	0.942444	0.942444	0.942444
macro avg	0.944009	0.945692	0.942424	25193.00000
weighted avg	0.947393	0.942444	0.942499	25193.00000

---

Tuned\_Logistic\_Regression\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.955246	0.973900	0.964483	53870.00000
1	0.973380	0.954372	0.963782	53870.00000
accuracy	0.964136	0.964136	0.964136	0.964136
macro avg	0.964313	0.964136	0.964132	107740.00000
weighted avg	0.964313	0.964136	0.964132	107740.00000

---

Tuned\_Logistic\_Regression\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.993192	0.899027	0.943767	13469.00000
1	0.895393	0.992921	0.941638	11724.00000
accuracy	0.942722	0.942722	0.942722	0.942722
macro avg	0.944292	0.945974	0.942702	25193.00000
weighted avg	0.947680	0.942722	0.942776	25193.00000

---

Tuned\_Logistic\_Regression\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.960179	0.973529	0.966808	53870.00000
1	0.969099	0.953622	0.961298	46897.00000
accuracy	0.964264	0.964264	0.964264	0.964264
macro avg	0.964639	0.963575	0.964053	100767.00000
weighted avg	0.964330	0.964264	0.964244	100767.00000

---

Simple\_Logistic\_Regresssion\_on\_balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.996026	0.725815	0.839718	13469.00000
1	0.759852	0.996673	0.862298	11724.00000
accuracy	0.851864	0.851864	0.851864	0.851864
macro avg	0.877939	0.861244	0.851008	25193.00000
weighted avg	0.886118	0.851864	0.850226	25193.00000

---

Simple\_Logistic\_Regresssion\_on\_balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.972322	0.976889	0.974600	53870.00000
1	0.976780	0.972192	0.974481	53870.00000
accuracy	0.974541	0.974541	0.974541	0.974541
macro avg	0.974551	0.974541	0.974540	107740.00000
weighted avg	0.974551	0.974541	0.974540	107740.00000

---

Simple\_Logistic\_Regression\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.995620	0.742520	0.850642	13469.00000
1	0.771059	0.996247	0.869306	11724.00000
accuracy	0.860596	0.860596	0.860596	0.860596
macro avg	0.883339	0.869383	0.859974	25193.00000
weighted avg	0.891116	0.860596	0.859328	25193.00000

---

Simple\_Logistic\_Regression\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.973266	0.978559	0.975906	53870.000000
1	0.975217	0.969124	0.972161	46897.000000
accuracy	0.974168	0.974168	0.974168	0.974168
macro avg	0.974241	0.973842	0.974033	100767.000000
weighted avg	0.974174	0.974168	0.974163	100767.000000

---

Tuned\_KNN\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.993329	0.995026	0.994177	13469.000000
1	0.994274	0.992323	0.993298	11724.000000
accuracy	0.993768	0.993768	0.993768	0.993768
macro avg	0.993802	0.993675	0.993737	25193.000000
weighted avg	0.993769	0.993768	0.993768	25193.000000

---

Tuned\_KNN\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.995498	0.997531	0.996514	53870.000000
1	0.997157	0.994818	0.995987	46897.000000
accuracy	0.996269	0.996269	0.996269	0.996269
macro avg	0.996328	0.996175	0.996250	100767.000000
weighted avg	0.996270	0.996269	0.996268	100767.000000

---

Tuned\_MLPClassifier\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.949597	0.952558	0.951075	13469.000000
1	0.945300	0.941914	0.943604	11724.000000
accuracy	0.947604	0.947604	0.947604	0.947604
macro avg	0.947449	0.947236	0.947340	25193.000000
weighted avg	0.947597	0.947604	0.947598	25193.000000

---

Tuned\_MLPClassifier\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.939947	0.950399	0.945144	53870.000000
1	0.949841	0.939280	0.944531	53870.000000
accuracy	0.944839	0.944839	0.944839	0.944839
macro avg	0.944894	0.944839	0.944838	107740.000000
weighted avg	0.944894	0.944839	0.944838	107740.000000

---

Tuned\_MLPClassifier\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.948722	0.958794	0.953731	13469.000000
1	0.952077	0.940464	0.946235	11724.000000
accuracy	0.950264	0.950264	0.950264	0.950264
macro avg	0.950399	0.949629	0.949983	25193.000000
weighted avg	0.950283	0.950264	0.950243	25193.000000

---

Tuned\_MLPClassifier\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.947237	0.961129	0.954132	53870.000000

```
1          0.954584  0.938504  0.946475   46897.000000
accuracy    0.950599  0.950599  0.950599      0.950599
macro avg    0.950911  0.949816  0.950304  100767.000000
weighted avg  0.950657  0.950599  0.950569  100767.000000
```

---

Tuned\_Logistic\_Regression\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.963806	0.968743	0.966268	13469.000000
1	0.963878	0.958205	0.961033	11724.000000
accuracy	0.963839	0.963839	0.963839	0.963839
macro avg	0.963842	0.963474	0.963651	25193.000000
weighted avg	0.963839	0.963839	0.963832	25193.000000

---

Tuned\_Logistic\_Regression\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.952349	0.972025	0.962086	53870.000000
1	0.971435	0.951364	0.961295	53870.000000
accuracy	0.961695	0.961695	0.961695	0.961695
macro avg	0.961892	0.961695	0.961691	107740.000000
weighted avg	0.961892	0.961695	0.961691	107740.000000

---

Tuned\_Logistic\_Regression\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.963026	0.968817	0.965913	13469.000000
1	0.963927	0.957267	0.960585	11724.000000
accuracy	0.963442	0.963442	0.963442	0.963442
macro avg	0.963476	0.963042	0.963249	25193.000000
weighted avg	0.963445	0.963442	0.963434	25193.000000

---

Tuned\_Logistic\_Regression\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.957740	0.971802	0.964720	53870.000000
1	0.967054	0.950743	0.958829	46897.000000
accuracy	0.962001	0.962001	0.962001	0.962001
macro avg	0.962397	0.961273	0.961775	100767.000000
weighted avg	0.962075	0.962001	0.961978	100767.000000

---

Simple\_Logistic\_Regresssion\_on\_balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.977286	0.971119	0.974193	13469.000000
1	0.967059	0.974070	0.970552	11724.000000
accuracy	0.972492	0.972492	0.972492	0.972492
macro avg	0.972173	0.972595	0.972372	25193.000000
weighted avg	0.972527	0.972492	0.972499	25193.000000

---

Simple\_Logistic\_Regresssion\_on\_balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	0.968520	0.976053	0.972272	53870.000000
1	0.975866	0.968275	0.972056	53870.000000
accuracy	0.972164	0.972164	0.972164	0.972164

```
macro avg      0.972193  0.972164  0.972164  107740.000000
weighted avg   0.972193  0.972164  0.972164  107740.000000
```

---

```
Simple_Logistic_Regression_on_Imbalanced_Dataset_test:
```

	precision	recall	f1-score	support
0	0.975517	0.973272	0.974393	13469.000000
1	0.969375	0.971938	0.970655	11724.000000
accuracy	0.972651	0.972651	0.972651	0.972651
macro avg	0.972446	0.972605	0.972524	25193.000000
weighted avg	0.972659	0.972651	0.972653	25193.000000

---

```
Simple_Logistic_Regression_on_Imbalanced_Dataset_train:
```

	precision	recall	f1-score	support
0	0.970499	0.978318	0.974393	53870.000000
1	0.974862	0.965840	0.970330	46897.000000
accuracy	0.972511	0.972511	0.972511	0.972511
macro avg	0.972681	0.972079	0.972362	100767.000000
weighted avg	0.972530	0.972511	0.972502	100767.000000

---

### Observation

Here we can observe all the models are providing metrics more than 0.9.  
Boosting models providing metrics more than 0.99.

## CHAPTER 6: MULTI-CLASS CLASSIFICATION (ONLY APPLYING BEST MODEL OF BINARY CLASS)

### Preprocessing for Multi-class classification

```
In [ ]: nadp_multi = nadp_add.copy(deep=True)
nadp_multi = nadp_multi.drop(["attack_or_normal","attack","lastflag","service_catego"])

In [ ]: # Checking null values
sum(nadp_multi.isna().sum())

Out[ ]: 0

In [ ]: # Checking duplicates after removing unwanted features
nadp_multi.duplicated().sum()

Out[ ]: np.int64(9)

In [ ]: # Drop duplicates
nadp_multi.drop_duplicates(keep="first",inplace=True)

In [ ]: # shape of nadp_multi
nadp_multi.shape

Out[ ]: (125964, 58)

In [ ]: # Separate features and target
nadp_X_multi = nadp_multi.drop(["attack_category"], axis=1) # Features
nadp_y_multi = nadp_multi["attack_category"] # Target variable
```

```
# Split the data with stratification
nadp_X_train_multi, nadp_X_test_multi, nadp_y_train_multi, nadp_y_test_multi = train_test_split(X, y, test_size=0.2, stratify=y)

# Verify the value counts in train and test sets
print("Training set value counts:\n", nadp_y_train_multi.value_counts())
print("Test set value counts:\n", nadp_y_test_multi.value_counts())
```

Training set value counts:

attack_category	count
Normal	53874
DOS	36741
Probe	9318
R2L	796
U2R	42
Name: count, dtype: int64	

Test set value counts:

attack_category	count
Normal	13469
DOS	9186
Probe	2329
R2L	199
U2R	10
Name: count, dtype: int64	

In [ ]: # three categorical features  
nadp\_X\_train\_multi.describe(include = "object")

Out[ ]:

	protocoltype	service	flag
count	100771	100771	100771
unique	3	70	11
top	tcp	http	SF
freq	82115	32268	59963

In [ ]: # checking duplicates after train test split  
nadp\_X\_train\_multi[nadp\_X\_train\_multi.duplicated(keep=False)]

Out[ ]:

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urg
30680	0	icmp	tim_i	SF	564	0	0	0	0
89324	0	tcp	finger	S0	0	0	1	0	0
37107	0	tcp	finger	S0	0	0	1	0	0
121116	0	tcp	finger	S0	0	0	1	0	0
13210	0	tcp	finger	S0	0	0	1	0	0
1143	0	icmp	tim_i	SF	564	0	0	0	0
67588	0	tcp	finger	S0	0	0	1	0	0
72491	0	tcp	finger	S0	0	0	1	0	0

In [ ]: # checking duplicates related nadp\_y\_train\_multi  
nadp\_y\_train\_multi[nadp\_X\_train\_multi.duplicated(keep=False)]

Out[ ]:

	attack_category
30680	Normal
89324	DOS
37107	Normal
121116	DOS
13210	Normal
1143	DOS
67588	DOS
72491	Normal

Name: attack\_category, dtype: object

In [ ]: # REMOVING DUPLICATES WHOSE nadp\_y\_train\_multi == 0 (keeping attacked rows)  
# Create a boolean mask for y\_train where the value is 0  
mask\_y\_equals\_normal = nadp\_y\_train\_multi == "Normal"

```

# Identify duplicates in X_train where y_train is 0
duplicates_mask = nadp_X_train_multi.duplicated(keep=False)

# Combine both masks to identify the rows to keep
rows_to_keep = nadp_X_train_multi[~(duplicates_mask & mask_y_equals_normal)]

# Remove duplicates from X_train and corresponding values in y_train
nadp_X_train_multi = nadp_X_train_multi.loc[rows_to_keep.index]
nadp_y_train_multi = nadp_y_train_multi.loc[rows_to_keep.index]

# Optionally, reset the index
nadp_X_train_multi.reset_index(drop=True, inplace=True)
nadp_y_train_multi.reset_index(drop=True, inplace=True)

```

In [ ]: # Final check of duplicates  
nadp\_X\_train\_multi.duplicated().sum()

Out[ ]: np.int64(0)

```

In [ ]: nadp_X_train_multi_encoded = nadp_X_train_multi.copy(deep=True)
nadp_y_train_multi_encoded = nadp_y_train_multi.copy(deep=True)

# Get value counts from the training set and create encoding for 'attack_category'
attack_category_value_counts = nadp_y_train_multi_encoded.value_counts()
attack_category_encoding = {category: rank for rank, category in enumerate(attack_ca
nadp_y_train_multi_encoded = nadp_y_train_multi_encoded.map(attack_category_encoding)

# Initialize OneHotEncoder
ohe_encoder = OneHotEncoder(drop="first", sparse_output=False) # Drop first to avoid

# Fit and transform the selected columns
encoded_data = ohe_encoder.fit_transform(nadp_X_train_multi_encoded[['protcoltype', 

# Convert to DataFrame with proper column names
encoded_nadp_X_train_multi_encoded = pd.DataFrame(encoded_data, columns=ohe_encoder.

# reset index
nadp_X_train_multi_encoded = nadp_X_train_multi_encoded.reset_index(drop=True)
encoded_nadp_X_train_multi_encoded = encoded_nadp_X_train_multi_encoded.reset_index()

# Combine the original DataFrame with the encoded DataFrame
nadp_X_train_multi_encoded = pd.concat([nadp_X_train_multi_encoded.drop(columns=['pr

# Get value counts from the training set and create encoding for 'service'
service_value_counts = nadp_X_train_multi_encoded['service'].value_counts()
service_encoding = {category: rank for rank, category in enumerate(service_value_cou
nadp_X_train_multi_encoded['service'] = nadp_X_train_multi_encoded['service'].map(se

```

In [ ]: sum(nadp\_X\_train\_multi\_encoded.isna().sum())

Out[ ]: 0

In [ ]: nadp\_X\_train\_multi\_encoded.shape

Out[ ]: (100767, 67)

```

In [ ]: # Save OneHotEncoder
with open('ohe_encoder_multi.pkl', 'wb') as f:
    pickle.dump(ohe_encoder, f)

# Save service encoding mapping
with open('service_encoding_multi.pkl', 'wb') as f:
    pickle.dump(service_encoding, f)

# Save attack category encoding mapping
with open('attack_category_encoding_multi.pkl', 'wb') as f:
    pickle.dump(attack_category_encoding, f)

```

In [ ]: nadp\_X\_train\_multi\_encoded.columns

```
Out[ ]: Index(['duration', 'service', 'srcbytes', 'dstbytes', 'land', 'wrongfragment',  
       'urgent', 'hot', 'numfailedlogins', 'loggedin', 'numcompromised',  
       'rootshell', 'suattempted', 'numroot', 'numfilecreations', 'numshells',  
       'numaccessfiles', 'ishostlogin', 'isguestlogin', 'count', 'srvcount',  
       'serrorrate', 'srvserrorrate', 'rerrorrate', 'srvrerrorrate',  
       'samesrvrate', 'diffsrvrate', 'srvdifffhostrate', 'dsthostcount',  
       'dsthostsrvcount', 'dsthostsamesrvrate', 'dsthostdiffsrvrate',  
       'dsthostsamesrcportrate', 'dsthostsrvdifffhostrate', 'dsthostsserrorrate',  
       'dsthostsrvserrorrate', 'dsthostrerrorrate', 'dsthostsrvrerrorrate',  
       'serrors_count', 'rerrors_count', 'samesrv_count', 'diffsrv_count',  
       'serrors_srvcount', 'rerrors_srvcount', 'srvdifffhost_srvcount',  
       'dsthost_serrors_count', 'dsthost_rerrors_count',  
       'dsthost_samesrv_count', 'dsthost_diffsrv_count',  
       'dsthost_serrors_srvcount', 'dsthost_rerrors_srvcount',  
       'dsthost_samesrcport_srvcount', 'dsthost_srvdifffhost_srvcount',  
       'srcbytes/sec', 'dstbytes/sec', 'protocoltype_tcp', 'protocoltype_udp',  
       'flag_REJ', 'flag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0',  
       'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH'],  
      dtype='object')
```

```
In [ ]: # Assuming nadp_X_test_multi is your test dataset  
nadp_X_test_multi_encoded = nadp_X_test_multi.copy(deep=True)  
nadp_y_test_multi_encoded = nadp_y_test_multi.copy(deep=True)  
  
# Apply frequency encoding for 'attack_category' in the test dataset  
nadp_y_test_multi_encoded = nadp_y_test_multi_encoded.map(attack_category_encoding)  
  
# Transform 'protocoltype' and 'flag' columns using the fitted OneHotEncoder  
encoded_test_data = ohe_encoder.transform(nadp_X_test_multi_encoded[['protocoltype',  
encoded_nadp_X_test_multi_encoded = pd.DataFrame(encoded_test_data, columns=ohe_enco  
  
# Reset index for both DataFrames  
encoded_nadp_X_test_multi_encoded = encoded_nadp_X_test_multi_encoded.reset_index(dr  
nadp_X_test_multi_encoded = nadp_X_test_multi_encoded.reset_index(drop=True)  
nadp_y_test_multi_encoded = nadp_y_test_multi_encoded.reset_index(drop=True)  
  
# Combine the original DataFrame with the encoded DataFrame  
nadp_X_test_multi_encoded = pd.concat([nadp_X_test_multi_encoded.drop(columns=['prot  
  
# Apply frequency encoding for 'service' in the test dataset  
nadp_X_test_multi_encoded['service'] = nadp_X_test_multi_encoded['service'].map(serv  
  
# For any new service types in the test dataset that weren't in the training set, as  
max_service_value = nadp_X_train_multi_encoded['service'].max()  
nadp_X_test_multi_encoded['service'].fillna(max_service_value + 1, inplace=True)
```

```
In [ ]: sum(nadp_X_test_multi_encoded.isna().sum())
```

```
Out[ ]: 0
```

```
In [ ]: nadp_X_test_multi_encoded.shape
```

```
Out[ ]: (25193, 67)
```

```
In [ ]: # SCALING  
# Create a StandardScaler object  
nadp_X_train_multi_scaler = StandardScaler()  
  
# Fit the scaler to the training features and transform them  
nadp_X_train_multi_scaled = nadp_X_train_multi_scaler.fit_transform(nadp_X_train_mu  
  
# Convert the scaled training features back to a DataFrame  
nadp_X_train_multi_scaled = pd.DataFrame(nadp_X_train_multi_scaled, columns=nadp_X_t
```

```
In [ ]: # Scale the test features using the same scaler  
nadp_X_test_multi_scaled = nadp_X_train_multi_scaler.transform(nadp_X_test_multi_enc  
  
# Convert the scaled test features back to a DataFrame  
nadp_X_test_multi_scaled = pd.DataFrame(nadp_X_test_multi_scaled, columns=nadp_X_tes
```

```
In [ ]: # Save the scaler to a file  
with open('nadp_X_train_multi_scaler.pkl', 'wb') as file:
```

```
pickle.dump(nadp_X_train_multi_scaler, file)
```

```
In [ ]: nadp_X_train_multi_scaled.columns
```

```
Out[ ]: Index(['duration', 'service', 'srcbytes', 'dstbytes', 'land', 'wrongfragment',  
       'urgent', 'hot', 'numfailedlogins', 'loggedin', 'numcompromised',  
       'rootshell', 'suattempted', 'numroot', 'numfilecreations', 'numshells',  
       'numaccessfiles', 'ishostlogin', 'isguestlogin', 'count', 'srvcount',  
       'serrorrate', 'srvserrorrate', 'rerrorrate', 'srvrerrorrate',  
       'samesrvrate', 'diffsrvrate', 'srvdifffhostrate', 'dsthostcount',  
       'dsthostsrvcount', 'dsthostsamesrvrate', 'dsthostdiffsrvrate',  
       'dsthostsamesrcportrate', 'dsthostsrvdifffhostrate', 'dsthostsserrorrate',  
       'dsthostsrvserrorrate', 'dsthostrerrorrate', 'dsthostsrvrerrorrate',  
       'serrors_count', 'rerrors_count', 'samesrv_count', 'diffsrv_count',  
       'serrors_srvcount', 'rerrors_srvcount', 'srvdifffhost_srvcount',  
       'dsthost_serrors_count', 'dsthost_rerrors_count',  
       'dsthost_samesrv_count', 'dsthost_diffsrv_count',  
       'dsthost_serrorssrvcount', 'dsthost_rerrorssrvcount',  
       'dsthost_samesrcport_srvcount', 'dsthost_srvdifffhost_srvcount',  
       'srcbytes/sec', 'dstbytes/sec', 'protocoltype_tcp', 'protocoltype_udp',  
       'flag_REJ', 'flag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0',  
       'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH'],  
       dtype='object')
```

```
In [ ]: def calculate_vif(X):  
    vif = pd.DataFrame()  
    vif["Features"] = X.columns  
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]  
    return vif  
  
def remove_worst_feature(X):  
    vif = calculate_vif(X)  
    vif["VIF"] = round(vif["VIF"], 2)  
    vif = vif.sort_values(by="VIF", ascending=False)  
  
    # Check if all VIF values are Less than 10  
    if vif["VIF"].max() < 10:  
        return X # Stop if all VIFs are acceptable  
  
    # Remove the feature with the highest VIF  
    worst_feature = vif["Features"].iloc[0]  
    print(f"Removing feature: {worst_feature} with VIF: {vif['VIF'].iloc[0]}")  
  
    # Recursively call the function with the reduced dataset  
    return remove_worst_feature(X.drop(columns=[worst_feature]))  
  
# VIF should be applied only among continuous features  
X_t = nadp_X_train_multi_scaled[['duration', 'srcbytes', 'dstbytes', 'wrongfragment',  
       'urgent', 'hot', 'numfailedlogins', 'numcompromised', 'numroot', 'numfilecrea',  
       'numaccessfiles', 'count', 'srvcount', 'serrorrate', 'srvserrorrate', 'rerror',  
       'samesrvrate', 'diffsrvrate', 'srvdifffhostrate', 'dsthostcount',  
       'dsthostsrvcount', 'dsthostsamesrvrate', 'dsthostdiffsrvrate',  
       'dsthostsamesrcportrate', 'dsthostsrvdifffhostrate', 'dsthostsserrorrate',  
       'dsthostsrvserrorrate', 'dsthostrerrorrate', 'dsthostsrvrerrorrate',  
       'serrors_count', 'rerrors_count', 'samesrv_count', 'diffsrv_count',  
       'serrors_srvcount', 'rerrors_srvcount', 'srvdifffhost_srvcount',  
       'dsthost_serrors_count', 'dsthost_rerrors_count',  
       'dsthost_samesrv_count', 'dsthost_diffsrv_count',  
       'dsthost_serrorssrvcount', 'dsthost_rerrorssrvcount',  
       'dsthost_samesrcport_srvcount', 'dsthost_srvdifffhost_srvcount',  
       'srcbytes/sec', 'dstbytes/sec']]  
VIF_reduced = remove_worst_feature(X_t)  
  
# The reduced dataset will have all VIFs < 10  
print("Final features after VIF removal:", VIF_reduced.columns)
```

```

Removing feature: count with VIF: 297.16
Removing feature: numroot with VIF: 282.89
Removing feature: srvserrorrate with VIF: 126.5
Removing feature: dsthosterrorrate with VIF: 73.0
Removing feature: srverrorrate with VIF: 60.76
Removing feature: dsthostsrvserrorrate with VIF: 47.86
Removing feature: srvcount with VIF: 31.58
Removing feature: dsthostsamesrvrate with VIF: 28.66
Removing feature: dsthost_serrors_count with VIF: 22.91
Removing feature: dsthosterrorrate with VIF: 20.91
Removing feature: dsthostsrvrerrorrate with VIF: 18.44
Removing feature: dsthost_diffsrv_count with VIF: 14.87
Removing feature: samesrvrate with VIF: 13.34
Final features after VIF removal: Index(['duration', 'srcbytes', 'dstbytes', 'wrongfragment', 'urgent', 'hot',
       'numfailedlogins', 'numcompromised', 'numfilecreations', 'numshells',
       'numaccessfiles', 'serrorrate', 'rerrorrate', 'diffsrvrate',
       'srvdifffhostrate', 'dsthostcount', 'dsthostsrvcount',
       'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
       'dsthostsrvdifffhostrate', 'serrors_count', 'rerrors_count',
       'samesrv_count', 'diffsrv_count', 'serrors_srvcount',
       'rerrors_srvcount', 'srvdifffhost_srvcount', 'dsthost_rerrors_count',
       'dsthost_samesrv_count', 'dsthost_serrors_srvcount',
       'dsthost_rerrors_srvcount', 'dsthost_samesrcport_srvcount',
       'dsthost_srvdifffhost_srvcount', 'srcbytes/sec', 'dstbytes/sec'],
      dtype='object')

```

VIF reduced features are same in both multi and binary class classification

```
In [ ]: # combine categorical features to VIF reduced features
VIF_reduced_columns = ['duration', 'srcbytes', 'dstbytes', 'wrongfragment', 'urgent',
                       'numfailedlogins', 'numcompromised', 'numfilecreations', 'numshells',
                       'numaccessfiles', 'serrorrate', 'rerrorrate', 'diffsrvrate',
                       'srvdifffhostrate', 'dsthostcount', 'dsthostsrvcount',
                       'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
                       'dsthostsrvdifffhostrate', 'serrors_count', 'rerrors_count',
                       'samesrv_count', 'diffsrv_count', 'serrors_srvcount',
                       'rerrors_srvcount', 'srvdifffhost_srvcount', 'dsthost_rerrors_count',
                       'dsthost_samesrv_count', 'dsthost_serrors_srvcount',
                       'dsthost_rerrors_srvcount', 'dsthost_samesrcport_srvcount',
                       'dsthost_srvdifffhost_srvcount', 'srcbytes/sec', 'dstbytes/sec']
cat_features = ['flag_REJ', 'flag_RST0', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0',
                'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH', 'isguestlogin',
                'ishostlogin', 'land', 'loggedin', 'protocoltype_tcp', 'protocoltype_rootshell',
                'service', 'suattempted']
final_selected_features = VIF_reduced_columns + cat_features
print(f"Number of final_selected_features : {len(final_selected_features)}")
print(f"Number of features removed by VIF : {nadp_X_train_multi_scaled.shape[1] - len(VIF_reduced_columns)}")

Number of final_selected_features : 54
Number of features removed by VIF : 13
```

```
In [ ]: vif = calculate_vif(nadp_X_train_multi_scaled[VIF_reduced_columns])
vif["VIF"] = round(vif["VIF"], 2)
vif = vif.sort_values(by="VIF", ascending=False)
vif
```

Out[ ]:

	Features	VIF
21	errors_count	9.80
28	dsthost_samesrv_count	8.12
27	dsthost_errors_count	7.89
11	serrorrate	7.83
16	dsthostsrvcount	7.53
23	diffsrv_count	6.53
15	dsthostcount	5.90
12	rerrorrate	5.45
20	serrors_count	4.73
18	dsthostsamesrcportrate	4.02
24	serrors_srvcount	3.25
31	dsthost_samesrcport_srvcount	3.18
25	rerrors_srvcount	3.13
32	dsthost_svrdiffhost_srvcount	2.86
17	dsthostdiffsvrrate	2.86
19	dsthostsrvdiffhostrate	2.51
7	numcompromised	2.16
26	svrdiffhost_srvcount	2.09
13	diffsrvrate	2.03
29	dsthost_serrors_srvcount	2.03
10	numaccessfiles	1.83
30	dsthost_errors_srvcount	1.80
14	svrdiffhostrate	1.72
0	duration	1.39
34	dstbytes/sec	1.37
22	samesrv_count	1.36
5	hot	1.06
3	wrongfragment	1.06
8	numfilecreations	1.04
6	numfailedlogins	1.03
4	urgent	1.02
33	srcbytes/sec	1.01
1	srcbytes	1.01
2	dstbytes	1.00
9	numshells	1.00

In [ ]: `# Filter both the training and test datasets to keep only the selected features  
nadp_X_train_multi_final = nadp_X_train_multi_scaled[final_selected_features]  
nadp_X_test_multi_final = nadp_X_test_multi_scaled[final_selected_features]  
nadp_y_train_multi_final = nadp_y_train_multi_encoded.copy(deep = True)  
nadp_y_test_multi_final = nadp_y_test_multi_encoded.copy(deep = True)`

In [ ]: `# Saving final preprocessed dataframes to csv  
nadp_X_train_multi_final.to_csv("nadp_X_train_multi_final",index=False)`

```
nadp_X_test_multi_final.to_csv("nadp_X_test_multi_final",index=False)
nadp_y_train_multi_final.to_csv("nadp_y_train_multi_final",index=False)
nadp_y_test_multi_final.to_csv("nadp_y_test_multi_final",index=False)
```

```
In [ ]: # Loading the final preprocessed dataframes
nadp_X_train_multi_final = pd.read_csv("./nadp_X_train_multi_final")
nadp_X_test_multi_final = pd.read_csv("./nadp_X_test_multi_final")
nadp_y_train_multi_final = pd.read_csv("./nadp_y_train_multi_final")
nadp_y_test_multi_final = pd.read_csv("./nadp_y_test_multi_final")
```

```
In [ ]: nadp_X_train_multi_final.to_pickle('nadp_X_train_multi_final.pkl', compression='gzip')
```

## Creating Additional Features using Unsupervised Algorithms

```
In [ ]: multi_cmap = ListedColormap(sns.husl_palette(len(np.unique(nadp_y_train_multi_final)))
```

```
In [ ]: multi_train_umap = UMAP(init='random',n_neighbors=10,min_dist=0.1, random_state=42,n_
multi_test_umap = UMAP(init='random',n_neighbors=10,min_dist=0.1, random_state=42,n_
```

```
In [ ]: np.save("multi_train_umap.npy",multi_train_umap)
np.save("multi_test_umap.npy",multi_test_umap)
```

```
In [ ]: multi_train_umap = np.load("multi_train_umap.npy")
multi_test_umap = np.load("multi_test_umap.npy")
```

```
In [ ]: def create_multi_umap(multi_train_umap,nadp_y_train_multi_final,multi_test_umap,nadp
    # Create a figure with 1 row and 2 columns
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    # Plot the training data
    im_train = axs[0].scatter(multi_train_umap[:, 0],
                            multi_train_umap[:, 1],
                            s=25,
                            c=np.array(nadp_y_train_multi_final),
                            cmap=multi_cmap,
                            edgecolor='none')
    axs[0].set_title('Train Data UMAP')
    axs[0].set_xlabel('UMAP Dimension 1')
    axs[0].set_ylabel('UMAP Dimension 2')
```

```
# Plot the test data
im_test = axs[1].scatter(multi_test_umap[:, 0],
                        multi_test_umap[:, 1],
                        s=25,
                        c=np.array(nadp_y_test_multi_final),
                        cmap= multi_cmap,
                        edgecolor='none')
axs[1].set_title('Test Data UMAP')
axs[1].set_xlabel('UMAP Dimension 1')
axs[1].set_ylabel('UMAP Dimension 2')
# Add colorbar for training data
cbar_train = fig.colorbar(im_train, ax=axs[1], label='attack_category')
```

```
# Display the plots
plt.tight_layout()
plt.savefig(f"{run_name}_umap.png")
mlflow.log_artifact(f"{run_name}_umap.png")
plt.show()
```

```
In [ ]: # Logging the multi_ground_truth_umap.png artifact into mlflow
experiment_name = "nadp_multi"
run_name = "multi_ground_truth_umap"
# mlflow.create_experiment("nadp_multi")
mlflow.set_experiment("nadp_multi")

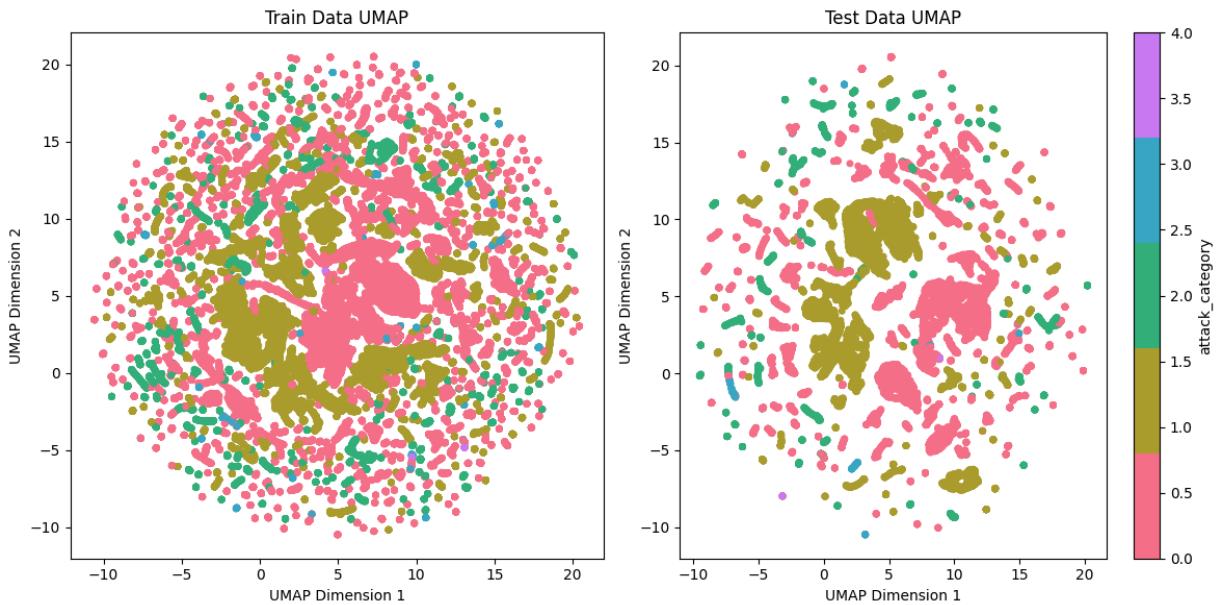
with mlflow.start_run(run_name=run_name):
```

```

try:
    # Log umap
    create_multi_umap(multi_train_umap,nadp_y_train_multi_final,multi_test_umap,
    print("MLFLOW Logging is completed")
except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

```

2025/05/17 07:57:41 INFO mlflow.tracking.fluent: Experiment with name 'nadp\_multi' does not exist. Creating a new experiment.



MLFLOW Logging is completed

In [ ]: `attack_category_encoding`

Out[ ]: `{'Normal': 0, 'DOS': 1, 'Probe': 2, 'R2L': 3, 'U2R': 4}`

In [ ]: `# Creating new dataframes to store the anomaly_scores`  
`nadp_X_train_multi_anomaly = nadp_X_train_multi_final.copy(deep = True)`  
`nadp_X_test_multi_anomaly = nadp_X_test_multi_final.copy(deep = True)`

In [ ]: `# Logging the multi_lof_umap.png artifact into mlflow`  
`experiment_name = "nadp_multi"`  
`run_name = "multi_lof"`  
`#mlflow.create_experiment("nadp_multi")`  
`mlflow.set_experiment("nadp_multi")`

`with mlflow.start_run(run_name=run_name):`  
`try:`  
 `# Log params`  
 `params = {"n_neighbors":20,"contamination":"auto","n_jobs": -1}`  
 `mlflow.log_params(params)`

`# Train dataset`  
 `train_multi_lof = LocalOutlierFactor(**params)`  
 `X_train_multi_lof_labels = train_multi_lof.fit_predict(nadp_X_train_multi_final)`  
 `X_train_multi_lof_labels = np.where(X_train_multi_lof_labels == -1,1,0)`  
 `nadp_X_train_multi_anomaly["multi_lof_nof"] = train_multi_lof.negative_outlier_`  
 `train_metrics = {"actual_n_neighbors":train_multi_lof.n_neighbors_,"offset":train_multi_lof.offset_}`  
 `mlflow.log_metrics(train_metrics)`

`# Test dataset`  
 `test_multi_lof = LocalOutlierFactor(**params)`  
 `X_test_multi_lof_labels = test_multi_lof.fit_predict(nadp_X_test_multi_final)`  
 `X_test_multi_lof_labels = np.where(X_test_multi_lof_labels == -1,1,0)`  
 `nadp_X_test_multi_anomaly["multi_lof_nof"] = test_multi_lof.negative_outlier_`  
 `test_metrics = {"actual_n_neighbors":test_multi_lof.n_neighbors_,"offset":test_multi_lof.offset_}`  
 `mlflow.log_metrics(test_metrics)`

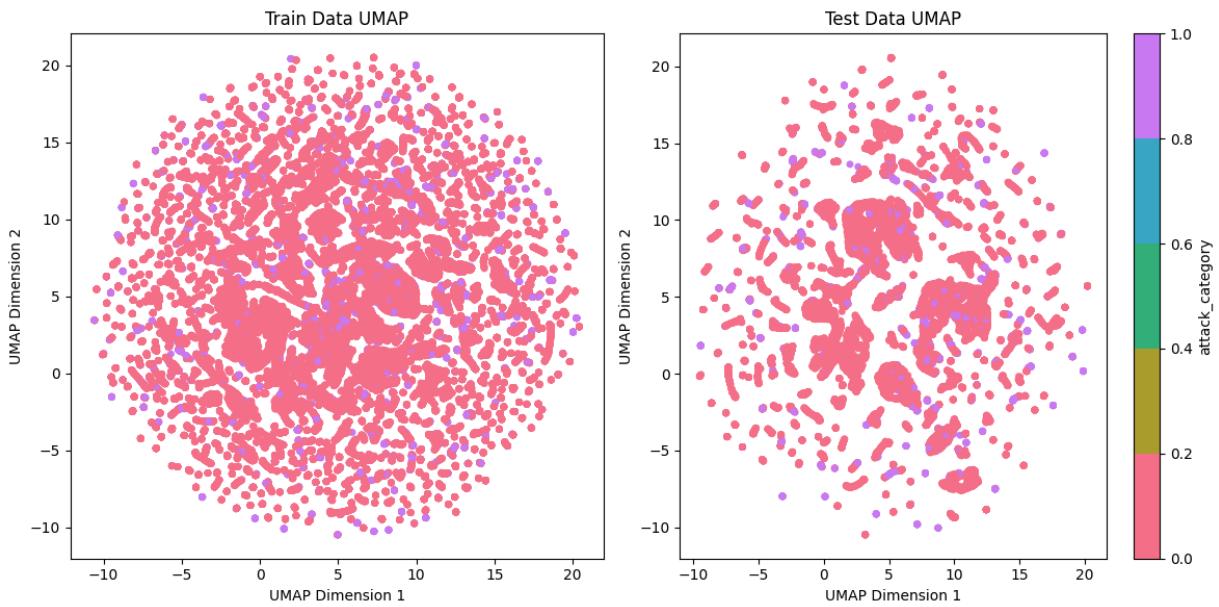
`# Log umap`  
 `create_multi_umap(multi_train_umap,X_train_multi_lof_labels,multi_test_umap,`

`# Log the model`  
 `mlflow.sklearn.log_model(train_multi_lof, f"{run_name}_train_model")`  
 `mlflow.sklearn.log_model(test_multi_lof, f"{run_name}_test_model")`  
 `print("MLFLOW Logging is completed")`

```

except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

```



```

2025/05/17 07:59:45 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/17 08:00:03 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto infer
the model signature.
2025/05/17 08:00:03 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/17 08:00:14 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto infer
the model signature.

```

MLFLOW Logging is completed

```

In [ ]: # logging the multi_iforest_umap.png artifact into mlflow
experiment_name = "nadp_multi"
run_name = "multi_iforest"
#mlflow.create_experiment("nadp_multi")
mlflow.set_experiment("nadp_multi")

with mlflow.start_run(run_name=run_name):
    try:
        # log params
        params = {"n_estimators":100,"contamination":"auto","n_jobs": -1,"random_state":42}
        mlflow.log_params(params)

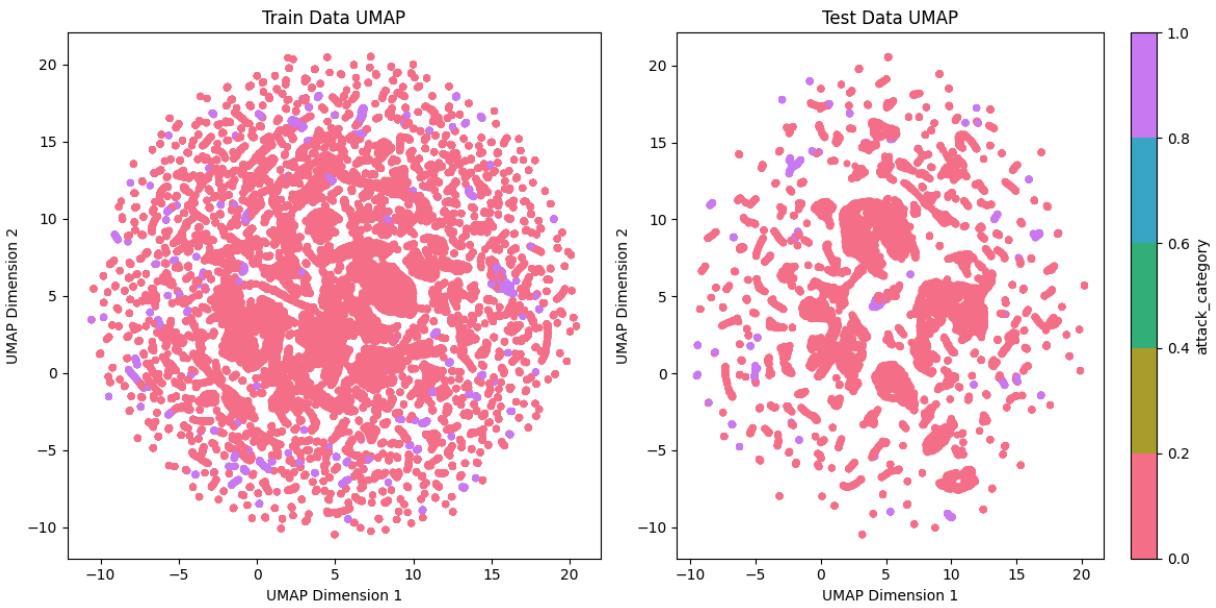
        # Train dataset
        train_multi_iforest = IsolationForest(**params)
        X_train_multi_iforest_labels = train_multi_iforest.fit_predict(nadp_X_train)
        X_train_multi_iforest_labels = np.where(X_train_multi_iforest_labels == -1,1,0)
        nadp_X_train_anomaly["multi_iforest_df"] = train_multi_iforest.decision_function(X_train)
        train_metrics = {"n_estimators":len(train_multi_iforest.estimators_),"offset":0}
        mlflow.log_metrics(train_metrics)

        # Test dataset
        X_test_multi_iforest_labels = train_multi_iforest.predict(nadp_X_test_multi)
        X_test_multi_iforest_labels = np.where(X_test_multi_iforest_labels == -1,1,0)
        nadp_X_test_anomaly["multi_iforest_df"] = train_multi_iforest.decision_function(X_test)

        # Log umap
        create_multi_umap(multi_train_umap,X_train_multi_iforest_labels,multi_test_u

        # Log the model
        mlflow.sklearn.log_model(train_multi_iforest, f"{run_name}_train_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")

```



```
2025/05/17 08:00:56 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

```
In [ ]: params = {"n_estimators":100,"contamination":"auto","n_jobs": -1,"random_state":42,"train_multi_iforest = IsolationForest(**params)
train_multi_iforest.fit(nadp_X_train_multi_final)
```

```
Out[ ]: ▾ IsolationForest ⓘ ? IsolationForest(n_jobs=-1, random_state=42)
```

```
In [ ]: with open("train_multi_iforest.pkl", "wb") as f:
pickle.dump(train_multi_iforest,f)
```

```
In [ ]: # Logging the multi_robust_cov_umap.png artifact into mlflow
experiment_name = "nadp_multi"
run_name = "multi_robust_cov"
#mlflow.create_experiment("nadp_multi")
mlflow.set_experiment("nadp_multi")

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

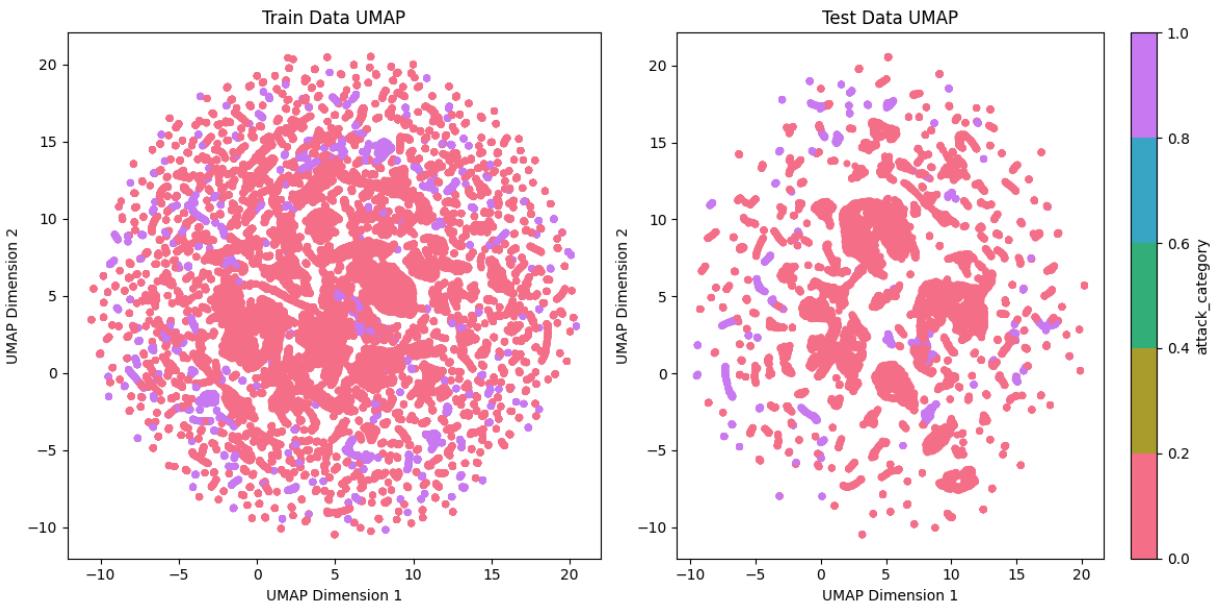
        # Log params
        params = {"contamination":0.1,"random_state":42}
        mlflow.log_params(params)

        # Train dataset
        train_multi_robust_cov = EllipticEnvelope(**params)
        X_train_multi_robust_cov_labels = train_multi_robust_cov.fit_predict(nadp_X_
X_train_multi_robust_cov_labels = np.where(X_train_multi_robust_cov_labels ==
nadp_X_train_multi_anomaly["multi_robust_cov_df"] = train_multi_robust_cov.de
train_metrics = {"offset":train_multi_robust_cov.offset_}
mlflow.log_metrics(train_metrics)

        # Test dataset
        X_test_multi_robust_cov_labels = train_multi_robust_cov.predict(nadp_X_test_
X_test_multi_robust_cov_labels = np.where(X_test_multi_robust_cov_labels ==
nadp_X_test_multi_anomaly["multi_robust_cov_df"] = train_multi_robust_cov.dec

        # Log umap
        create_multi_umap(multi_train_umap,X_train_multi_robust_cov_labels,multi_tes

        # Log the model
        mlflow.sklearn.log_model(train_multi_robust_cov, f"{run_name}_train_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")
```



```
2025/05/17 08:02:15 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

```
In [ ]: params = {"contamination":0.1,"random_state":42}
train_multi_robust_cov = EllipticEnvelope(**params)
train_multi_robust_cov = train_multi_robust_cov.fit(nadp_X_train_multi_final)

In [ ]: with open("train_multi_robust_cov.pkl","wb") as f:
pickle.dump(train_multi_robust_cov,f)

In [ ]: # Logging the multi_one_class_svm_umap.png artifact into mlflow
experiment_name = "nadp_multi"
run_name = "multi_one_class_svm"
#mlflow.create_experiment("nadp_multi")
mlflow.set_experiment("nadp_multi")

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

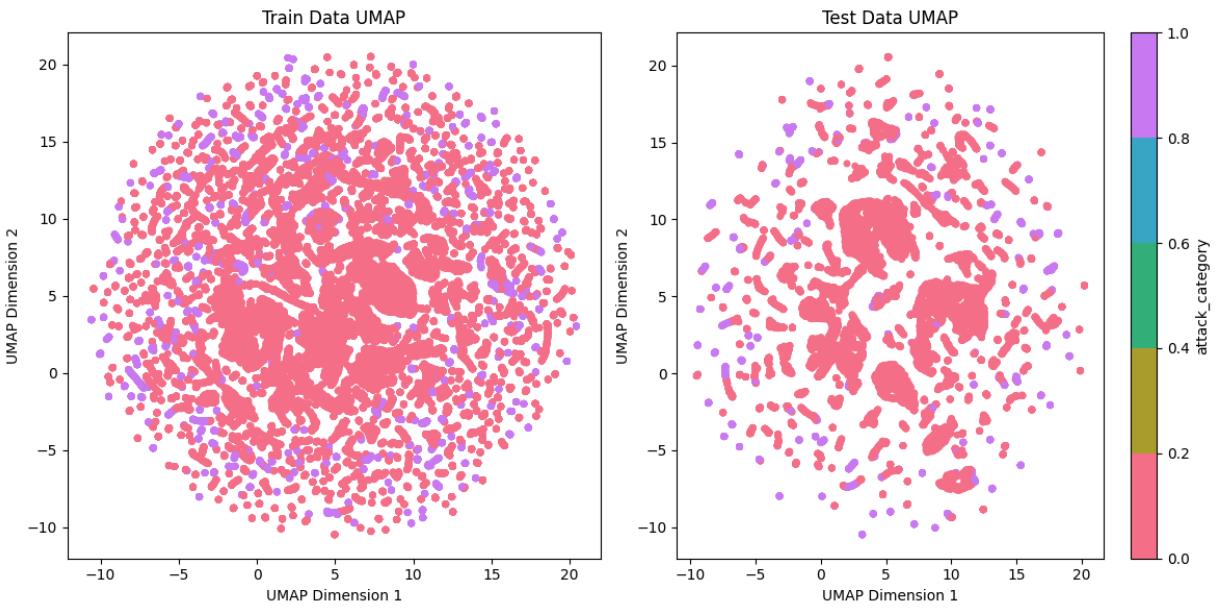
        # Log params
        params = {"nu":0.1, "verbose":0}
        mlflow.log_params(params)

        # Train dataset
        train_multi_one_class_svm = OneClassSVM(**params)
        X_train_multi_one_class_svm_labels = train_multi_one_class_svm.fit_predict(nadp_X_train_multi_anomaly["multi_one_class_svm_df"])
        np.where(X_train_multi_one_class_svm_labels == -1, 1, 0)
        train_metrics = {"offset":train_multi_one_class_svm.offset_,"n_support_vectors":train_multi_one_class_svm.n_support_}
        mlflow.log_metrics(train_metrics)

        # Test dataset
        X_test_multi_one_class_svm_labels = train_multi_one_class_svm.predict(nadp_X_test_multi_anomaly["multi_one_class_svm_df"])
        np.where(X_test_multi_one_class_svm_labels == -1, 1, 0)
        test_metrics = {"offset":train_multi_one_class_svm.offset_,"n_support_vectors":train_multi_one_class_svm.n_support_}
        mlflow.log_metrics(test_metrics)

        # Log umap
        create_multi_umap(multi_train_umap,X_train_multi_one_class_svm_labels,multi_test_umap,X_test_multi_one_class_svm_labels)

        # Log the model
        mlflow.sklearn.log_model(train_multi_one_class_svm, f"{run_name}_train_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")
```



```
2025/05/17 08:21:23 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto infer
the model signature.
```

MLFLOW Logging is completed

```
In [ ]: params = {"nu":0.1, "verbose":0}
train_multi_one_class_svm = OneClassSVM(**params)
train_multi_one_class_svm = train_multi_one_class_svm.fit(nadp_X_train_multi_final)

In [ ]: with open("train_multi_one_class_svm.pkl","wb") as f:
pickle.dump(train_multi_one_class_svm,f)

In [ ]: # Logging the multi_dbscan_umap.png artifact into mlflow
experiment_name = "nadp_multi"
run_name = "multi_dbscan"
#mlflow.create_experiment("nadp_multi")
mlflow.set_experiment("nadp_multi")

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

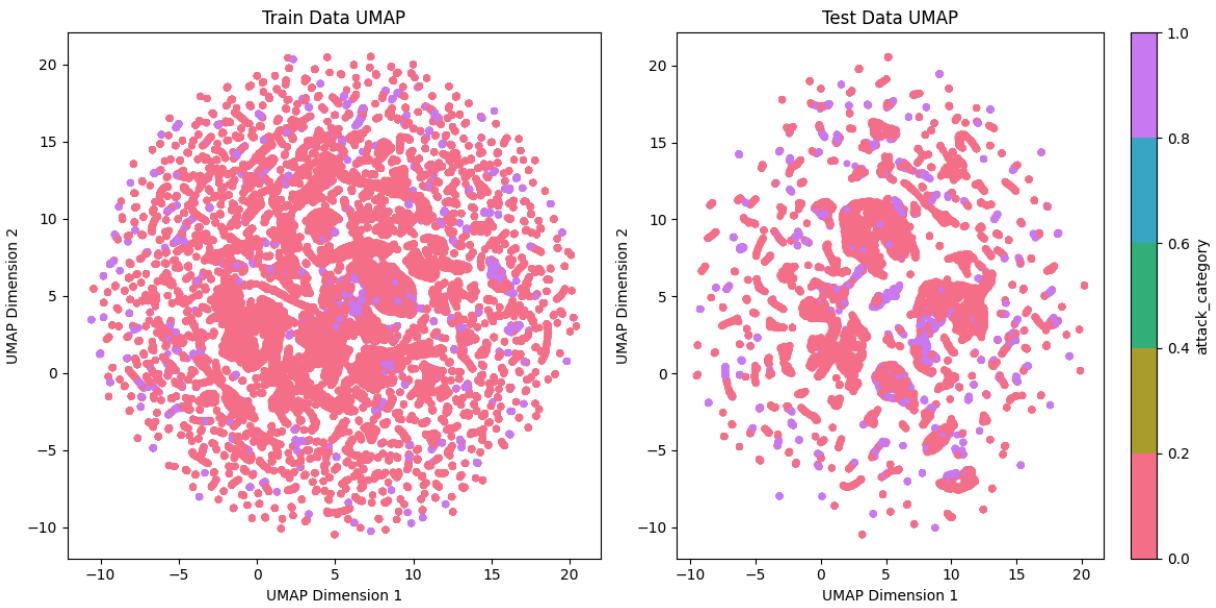
        # Log params
        params = {"eps":0.5, "min_samples":5, "n_jobs":-1}
        mlflow.log_params(params)

        # Train dataset
        train_multi_dbscan = DBSCAN(**params)
        X_train_multi_dbscan_labels = train_multi_dbscan.fit_predict(nadp_X_train_mu
X_train_multi_dbscan_labels = np.where(X_train_multi_dbscan_labels == -1,1,0)
nadp_X_train_multi_anomaly["multi_dbscan_labels"] = train_multi_dbscan.labels_
train_metrics = {"n_unique_labels":len(np.unique(train_multi_dbscan.labels_))}
mlflow.log_metrics(train_metrics)

        # Test dataset
        test_multi_dbscan = DBSCAN(**params)
        X_test_multi_dbscan_labels = test_multi_dbscan.fit_predict(nadp_X_test_multi
X_test_multi_dbscan_labels = np.where(X_test_multi_dbscan_labels == -1,1,0)
nadp_X_test_multi_anomaly["multi_dbscan_labels"] = test_multi_dbscan.labels_
test_metrics = {"n_unique_labels":len(np.unique(test_multi_dbscan.labels_))}
mlflow.log_metrics(test_metrics)

        # Log umap
        create_multi_umap(multi_train_umap,X_train_multi_dbscan_labels,multi_test_um

        # Log the model
        mlflow.sklearn.log_model(train_multi_dbscan, f"{run_name}_train_model")
        mlflow.sklearn.log_model(test_multi_dbscan, f"{run_name}_test_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")
```



```
2025/05/17 08:29:19 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/17 08:29:32 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto in
fer the model signature.
2025/05/17 08:29:32 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/17 08:29:38 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto in
fer the model signature.
```

MLFLOW Logging is completed

```
In [ ]: # Logging the multi_knn_umap.png artifact into mlflow
experiment_name = "nadp_multi"
run_name = "multi_knn"
#mlflow.create_experiment("nadp_multi")
mlflow.set_experiment("nadp_multi")

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

        # Log params
        params = {"n_neighbors":5,"n_jobs":-1}
        mlflow.log_params(params)

        # Train dataset
        train_multi_knn = NearestNeighbors(**params)
        train_multi_knn.fit(nadp_X_train_multi_final)
        train_distances, train_indices = train_multi_knn.kneighbors(nadp_X_train_multi_
nadp_X_train_multi_anomaly["multi_knn_kth_distance"] = train_distances[:, -1]

        # Test dataset
        # test_multi_knn = NearestNeighbors(**params)
        # X_train_multi_knn_Labels = test_multi_knn.fit(nadp_X_train_multi_final)
        test_distances, test_indices = train_multi_knn.kneighbors(nadp_X_test_multi_
nadp_X_test_multi_anomaly["multi_knn_kth_distance"] = test_distances[:, -1]

        # Log umap (No umap in knn because we cannot calculate labels in unsupervised
        # create_multi_umap(multi_train_umap,X_train_multi_knn_Labels,multi_test_uma

        # Log the model
        mlflow.sklearn.log_model(train_multi_knn, f"{run_name}_train_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")
```

```
2025/05/17 08:30:44 WARNING mlflow.sklearn: Model was missing function: predict. Not
logging python_function flavor!
2025/05/17 08:30:57 WARNING mlflow.models.model: Model logged without a signature and
input example. Please set `input_example` parameter when logging the model to auto in
fer the model signature.
```

MLFLOW Logging is completed

```
In [ ]: params = {"n_neighbors":5,"n_jobs":-1}
train_multi_knn = NearestNeighbors(**params)
train_multi_knn = train_multi_knn.fit(nadp_X_train_multi_final)

In [ ]: with open("train_multi_knn.pkl","wb") as f:
pickle.dump(train_multi_knn,f)

In [ ]: # Logging the multi_gmm_umap.png artifact into mlflow
experiment_name = "nadp_multi"
run_name = "multi_gmm"
#mlflow.create_experiment("nadp_multi")
mlflow.set_experiment("nadp_multi")

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

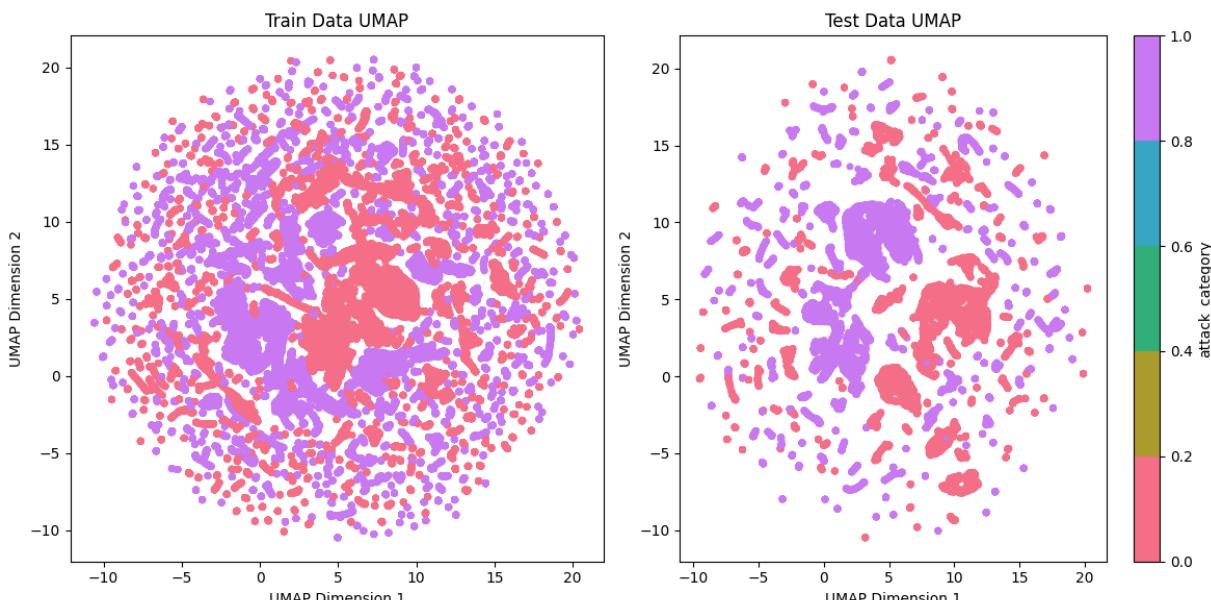
        # Log params
        params = {"n_components":2, "random_state":42,"verbose":0}
        mlflow.log_params(params)

        # Train dataset
        train_multi_gmm = GaussianMixture(**params)
        X_train_multi_gmm_labels = train_multi_gmm.fit_predict(nadp_X_train_multi_final)
        # X_train_multi_gmm_labels = np.where(X_train_multi_gmm_labels == -1,1,0)
        nadp_X_train_multi_anomaly["multi_gmm_score"] = train_multi_gmm.score_samples
        train_metrics = {"AIC":train_multi_gmm.aic(nadp_X_train_multi_final),"BIC":train_multi_gmm.bic(nadp_X_train_multi_final)}
        mlflow.log_metrics(train_metrics)

        # Test dataset
        X_test_multi_gmm_labels = train_multi_gmm.predict(nadp_X_test_multi_final)
        # X_test_multi_gmm_labels = np.where(X_test_multi_gmm_labels == -1,1,0)
        nadp_X_test_multi_anomaly["multi_gmm_score"] = train_multi_gmm.score_samples

        # Log umap
        create_multi_umap(multi_train_umap,X_train_multi_gmm_labels,multi_test_umap,run_name)

        # Log the model
        mlflow.sklearn.log_model(train_multi_gmm, f"{run_name}_train_model")
        print("MLFLOW Logging is completed")
    except Exception as e:
        print(f"Error in mlflow_logging_and_metric_printing: {e}")
```



2025/05/17 08:31:31 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging is completed

```
In [ ]: params = {"n_components":2, "random_state":42,"verbose":0}
train_multi_gmm = GaussianMixture(**params)
train_multi_gmm = train_multi_gmm.fit(nadp_X_train_multi_final)
```

```

In [ ]: with open("train_multi_gmm.pkl","wb") as f:
    pickle.dump(train_multi_gmm,f)

In [ ]: nadp_X_train_multi_anomaly.to_csv("nadp_X_train_multi_anomaly",index = False)
nadp_X_test_multi_anomaly.to_csv("nadp_X_test_multi_anomaly",index = False)

In [ ]: # Loading the anomaly preprocessed dataframes
nadp_X_train_multi_anomaly = pd.read_csv("./nadp_X_train_multi_anomaly")
nadp_X_test_multi_anomaly = pd.read_csv("./nadp_X_test_multi_anomaly")

In [ ]: # Standardize the data
k_means_scaler_multi = StandardScaler()
data = k_means_scaler_multi.fit_transform(nadp_X_train_multi_anomaly)

# Save the scaler for future use
with open('k_means_scaler_multi.pkl', 'wb') as f:
    pickle.dump(k_means_scaler_multi, f)

In [ ]: true_labels = np.array(nadp_y_train_multi_final)
kmeans = KMeans(n_clusters=5, random_state=42).fit(data)
predicted_labels = kmeans.labels_
ari = adjusted_rand_score(true_labels.reshape(-1), predicted_labels.reshape(-1))
print(f"K={5}, Adjusted Rand Index: {ari}")

K=5, Adjusted Rand Index: 0.6200087978604398

In [ ]: with open('kmeans_best_multi.pkl', 'wb') as f:
    pickle.dump(kmeans, f)

In [ ]: # Applying the best k_means_scaler and parameters on both train and test data
data_train = k_means_scaler_multi.transform(nadp_X_train_multi_anomaly)
data_test = k_means_scaler_multi.transform(nadp_X_test_multi_anomaly)

# Re-run the best model using KMeans with the best parameters
kmeans_best = KMeans(n_clusters=5, random_state=42)
kmeans_best.fit(data_train)
train_labels = kmeans_best.labels_
test_labels = kmeans_best.predict(data_test)

# Add the labels as a new feature
nadp_X_train_multi_anomaly["multi_kmeans_adv"] = train_labels
nadp_X_test_multi_anomaly["multi_kmeans_adv"] = test_labels

In [ ]: # Log metrics and model using MLflow
experiment_name = "nadp_multi"
run_name = "multi_kmeans_adv"
mlflow.set_experiment(experiment_name)

with mlflow.start_run(run_name=run_name):
    try:
        # Suppress warnings
        warnings.filterwarnings('ignore')

        # Log params
        params = {"n_cluster":5,"random_state":42}
        mlflow.log_params(params)

        # Define true labels for accuracy calculation
        train_true_labels = np.array(nadp_y_train_multi_final)
        test_true_labels = np.array(nadp_y_test_multi_final)

        # Flatten labels for comparison
        train_labels_flat = train_labels.flatten()
        test_labels_flat = test_labels.flatten()
        train_true_labels_flat = train_true_labels.flatten()
        test_true_labels_flat = test_true_labels.flatten()

        # Check for consistent lengths
        if len(train_true_labels_flat) != len(train_labels_flat):
            print(f"Mismatch between train_true_labels and train_labels: {len(train_")
            raise ValueError("Labels have inconsistent lengths.")
    
```

```

if len(test_true_labels_flat) != len(test_labels_flat):
    print(f"Mismatch between test_true_labels and test_labels: {len(test_true_labels_flat)} vs {len(test_labels_flat)}")
    raise ValueError("Labels have inconsistent lengths.")

# Calculate metrics
train_accuracy = accuracy_score(train_true_labels_flat, train_labels_flat)
mlflow.log_metric("train_accuracy", train_accuracy)

test_accuracy = accuracy_score(test_true_labels_flat, test_labels_flat)
mlflow.log_metric("test_accuracy", test_accuracy)

mlflow.log_metric("silhouette_score_train", silhouette_score(data_train, kmeans_best))
mlflow.log_metric("davies_bouldin_index_train", davies_bouldin_score(data_train, kmeans_best))

# Supervised metrics comparing predictions with true labels
mlflow.log_metric("fowlkes_mallows_index", fowlkes_mallows_score(train_true_labels_flat, train_labels_flat))
mlflow.log_metric("adjusted_mutual_info", adjusted_mutual_info_score(train_true_labels_flat, train_labels_flat))
mlflow.log_metric("adjusted_rand_score", adjusted_rand_score(train_true_labels_flat, train_labels_flat))
mlflow.log_metric("normalized_mutual_info", normalized_mutual_info_score(train_true_labels_flat, train_labels_flat))
mlflow.log_metric("homogeneity", homogeneity_score(train_true_labels_flat, train_labels_flat))
mlflow.log_metric("completeness", completeness_score(train_true_labels_flat, train_labels_flat))
mlflow.log_metric("v_measure", v_measure_score(train_true_labels_flat, train_labels_flat))

# Create Pairwise Confusion Matrix
pairwise_cm = pair_confusion_matrix(train_true_labels_flat, train_labels_flat)
pairwise_cm_disp = ConfusionMatrixDisplay(pairwise_cm)
pairwise_cm_path = f"{run_name}_pairwise_cm.png"
pairwise_cm_disp.plot(cmap=plt.cm.Blues)
plt.savefig(pairwise_cm_path)
plt.show()
plt.close()
mlflow.log_artifact(pairwise_cm_path)

# Create Contingency Matrix
cont_matrix = contingency_matrix(train_true_labels_flat, train_labels_flat)
cont_matrix_disp = ConfusionMatrixDisplay(cont_matrix)
contingency_cm_path = f"{run_name}_contingency_cm.png"
cont_matrix_disp.plot(cmap=plt.cm.Blues)
plt.savefig(contingency_cm_path)
plt.show()
plt.close()
mlflow.log_artifact(contingency_cm_path)

# Compute Learning curve data
train_sizes, train_scores, test_scores = learning_curve(kmeans_best, data_train, data_test)
train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)

# Plot the Learning curve
plt.figure()
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training Score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation Score")
plt.title(f"Learning Curve - {run_name}")
plt.xlabel("Training Examples")
plt.ylabel("Accuracy")
plt.legend(loc="best")
plt.grid()

# Save the plot
learning_curve_path = f"{run_name}_learning_curve.png"
plt.savefig(learning_curve_path)
plt.show()
plt.close()

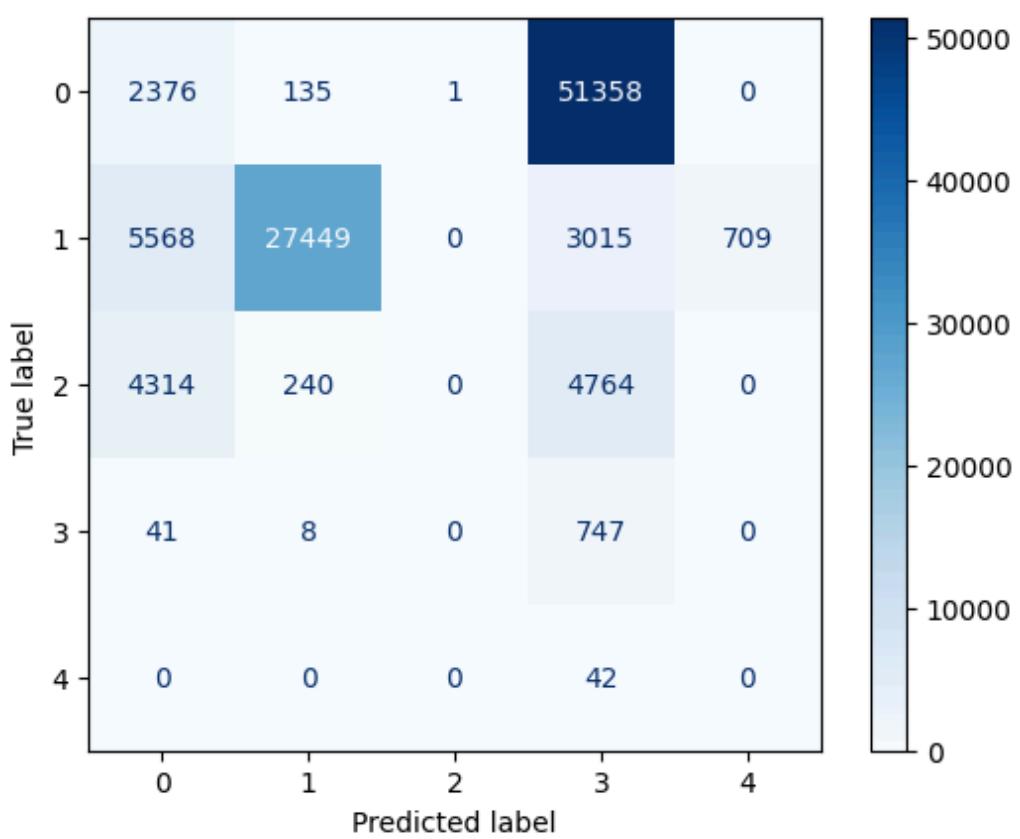
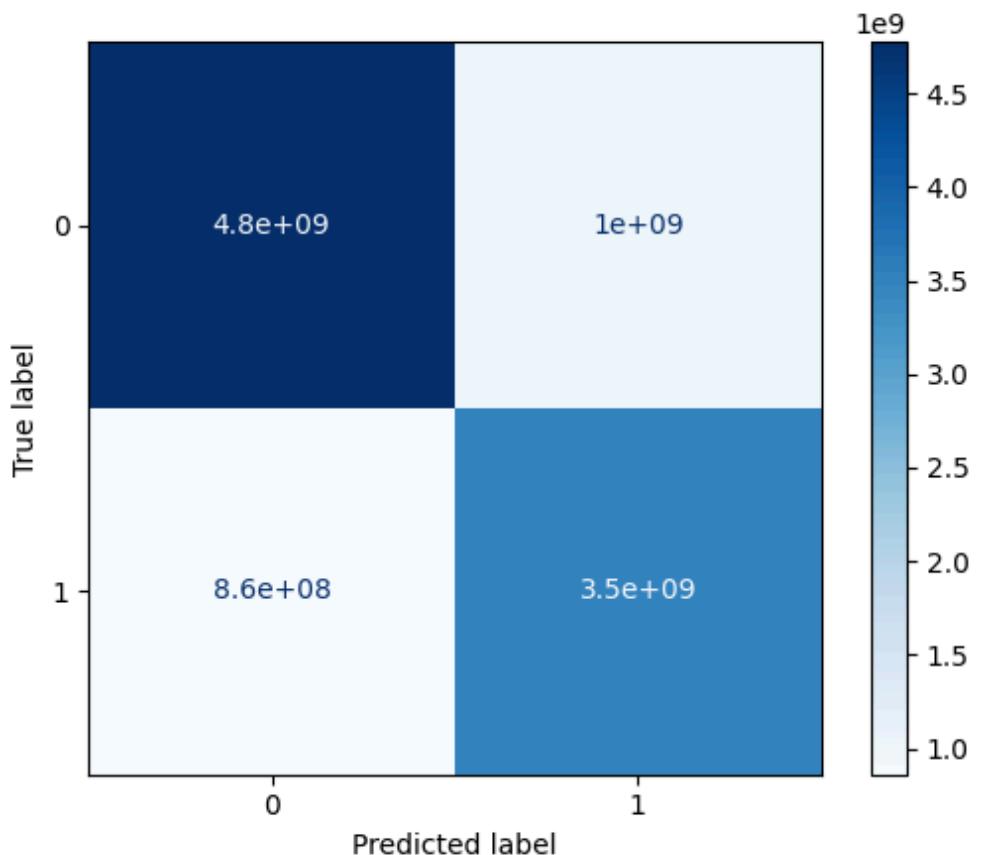
# Log the plot as an artifact in MLflow
mlflow.log_artifact(learning_curve_path)

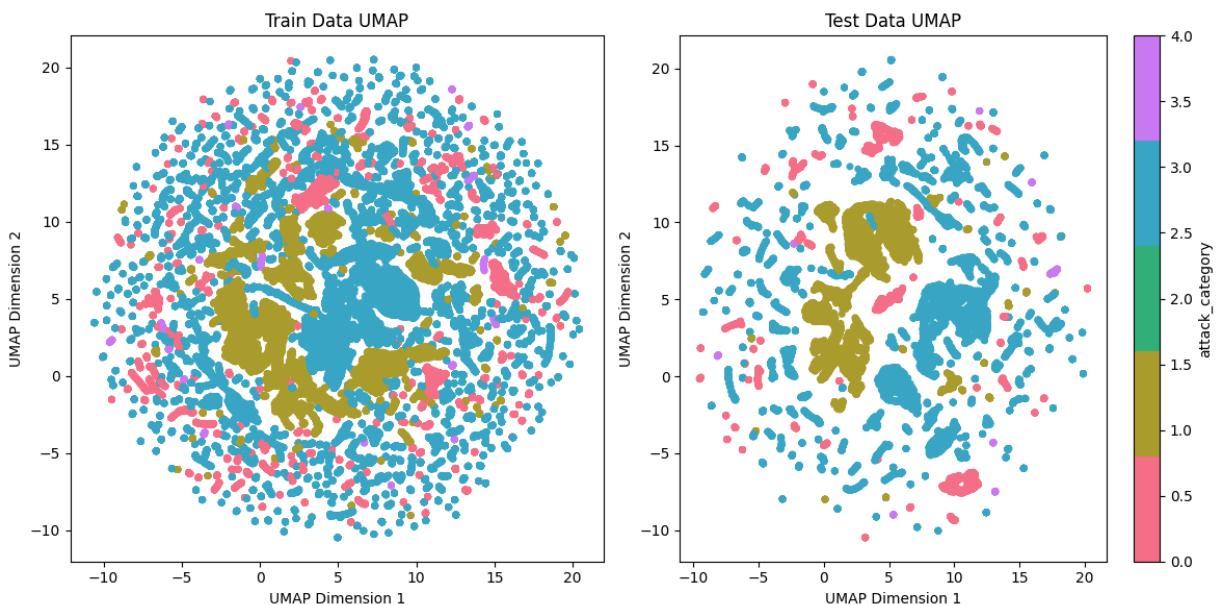
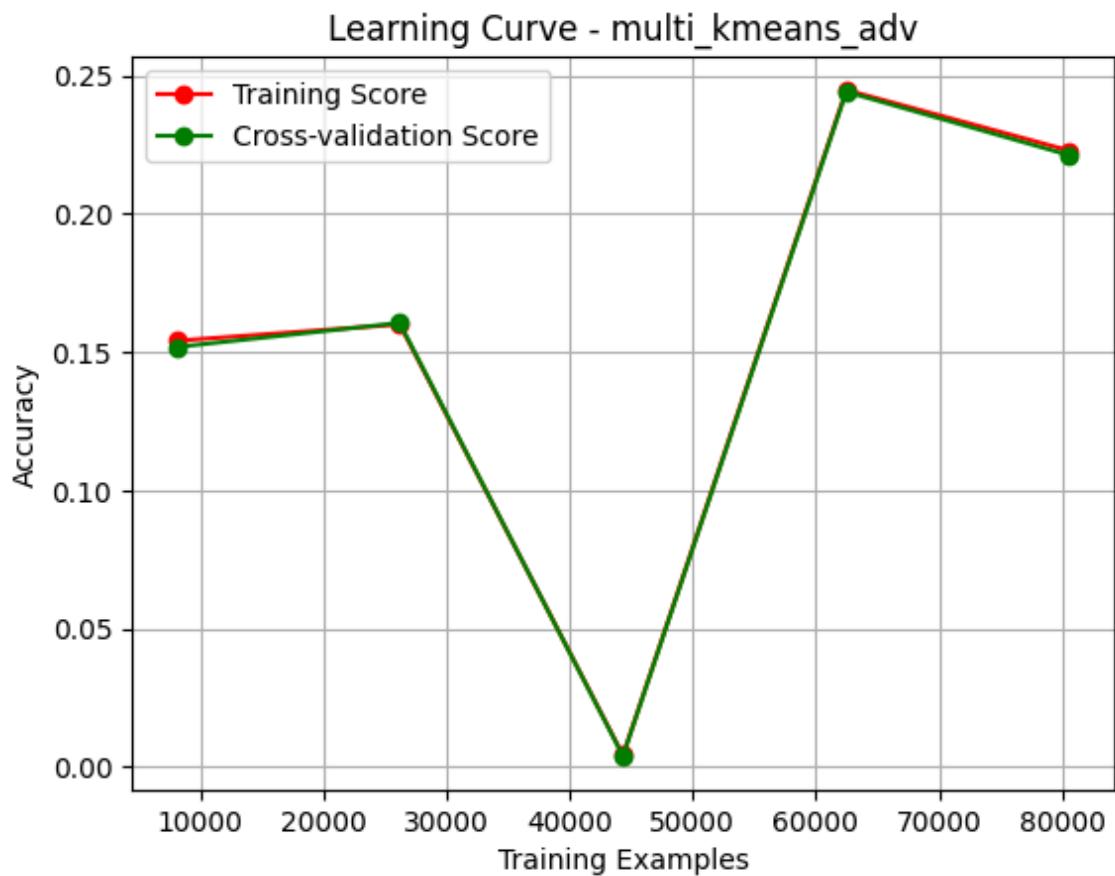
# Log umap
create_multi_umap(multi_train_umap, train_labels_flat, multi_test_umap, test_labels_flat)

# Log the trained model
mlflow.sklearn.log_model(kmeans_best, f"{run_name}_train_model")
print("MLFLOW Logging is completed")

```

```
except Exception as e:  
    print(f"Error in mlflow_logging_and_metric_printing: {e}")
```





```
2025/05/17 08:36:16 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```

MLFLOW Logging is completed

In [ ]: `display_all(nadp_X_train_multi_anomaly.head())`

	duration	srcbytes	dstbytes	wrongfragment	urgent	hot	numfailedlogins	num
0	-0.11051	-0.008279	-0.004662	-0.089697	-0.007717	-0.094196	-0.027186	
1	-0.11051	-0.008283	-0.004918	-0.089697	-0.007717	-0.094196	-0.027186	
2	-0.11051	-0.008345	-0.005329	-0.089697	-0.007717	-0.094196	-0.027186	
3	-0.11051	-0.008345	-0.005329	-0.089697	-0.007717	-0.094196	-0.027186	
4	-0.11051	-0.008345	-0.005329	-0.089697	-0.007717	-0.094196	-0.027186	

In [ ]: `kmeans_adv_scaler_multi = StandardScaler()  
nadp_X_train_multi_anomaly_final = kmeans_adv_scaler_multi.fit_transform(nadp_X_train_multi_anomaly)  
nadp_X_test_multi_anomaly_final = kmeans_adv_scaler_multi.transform(nadp_X_test_multi_anomaly)`

```
# Save the scaler for future use
with open('kmeans_adv_scaler_multi.pkl', 'wb') as f:
    pickle.dump(kmeans_adv_scaler_multi, f)
```

```
In [ ]: nadp_X_train_multi_anomaly_final = pd.DataFrame(nadp_X_train_multi_anomaly_final,columns=nadp_X_train_multi_anomaly_final.columns)
nadp_X_test_multi_anomaly_final = pd.DataFrame(nadp_X_test_multi_anomaly_final,columns=nadp_X_test_multi_anomaly_final.columns)

In [ ]: nadp_X_train_multi_anomaly_final.to_csv("nadp_X_train_multi_anomaly_final",index = False)
nadp_X_test_multi_anomaly_final.to_csv("nadp_X_test_multi_anomaly_final",index = False)

In [ ]: X_train_imb = pd.read_csv("./nadp_X_train_multi_anomaly_final")
X_test_imb = pd.read_csv("./nadp_X_test_multi_anomaly_final")

In [ ]: y_train_imb = pd.read_csv("./nadp_y_train_multi_final").squeeze()
y_test_imb = pd.read_csv("./nadp_y_test_multi_final").squeeze()

In [ ]: # SMOTE balancing
smt = SMOTE(random_state=42)
X_train_bal, y_train_bal = smt.fit_resample(X_train_imb,y_train_imb)
X_test_bal = X_test_imb.copy(deep = True)
y_test_bal = y_test_imb.copy(deep = True)

print("X_train_bal shape",X_train_bal.shape)
print("X_test_bal shape",X_test_bal.shape)
print("y_train_bal shape",y_train_bal.shape)
print("y_test_bal shape",y_test_bal.shape)
print("y_train_bal value_counts", y_train_bal.value_counts())
print("y_test_bal value_counts", y_test_bal.value_counts())

X_train_bal shape (269350, 62)
X_test_bal shape (25193, 62)
y_train_bal shape (269350,)
y_test_bal shape (25193,)
y_train_bal value_counts attack_category
0    53870
1    53870
2    53870
3    53870
4    53870
Name: count, dtype: int64
y_test_bal value_counts attack_category
0    13469
1    9186
2    2329
3    199
4     10
Name: count, dtype: int64

In [ ]: # Save the scaler for future use
with open('multi_smote.pkl', 'wb') as f:
    pickle.dump(smt, f)
```

## Multi-Class Classification Utilizing Optimal Models

```
In [ ]: # Define the base estimator (Decision Stump)
base_stump = DecisionTreeClassifier(max_depth=5)

# Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'n_estimators': stats.randint(100, 200), # Number of boosting stages
    'learning_rate': stats.uniform(0.01, 1.0), # Step size for boosting
    'algorithm': ['SAMME', 'SAMME.R'], # Algorithm to use for boosting
}

# Initialize the AdaBoost model with the decision stump as base estimator
ada_boost = AdaBoostClassifier(estimator=base_stump, random_state=42)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=ada_boost,
    param_distributions=param_dist,
    n_iter=2, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
```

```

        random_state=42,
        n_jobs=-1 # Use all available cores
    )

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)

```

Fitting 5 folds for each of 2 candidates, totalling 10 fits  
Best parameters found: {'algorithm': 'SAMME', 'learning\_rate': np.float64(0.7896910002727693), 'n\_estimators': 120}  
Best score: 0.998729740310875  
Tuning\_time: 795.1099212169647

```
In [ ]: nadp_multi_features = nadp_X_train_multi_anomaly_final.columns
```

```
In [ ]: # Initialize DataFrames
multi_time_df = pd.DataFrame(columns=["Model", "bal_type", "Training_Time", "Testing_Time"])
multi_feature_importance_df = pd.DataFrame()
```

```
In [ ]: def mlflow_logging_and_metric_printing(model, run_name, bal_type, X_train, y_train, mlflow):
    mlflow.set_experiment("nadp_multi")

    with mlflow.start_run(run_name=run_name):
        try:
            # Log parameters
            if params:
                mlflow.log_params(params)
            mlflow.log_param("bal_type", bal_type)

            # Calculate metrics
            train_metrics = {
                "Accuracy_train": accuracy_score(y_train, y_pred_train),
                "Precision_train_macro": precision_score(y_train, y_pred_train, average='macro'),
                "Recall_train_macro": recall_score(y_train, y_pred_train, average='macro'),
                "F1_score_train_macro": f1_score(y_train, y_pred_train, average='macro'),
                "F2_score_train_macro": fbeta_score(y_train, y_pred_train, beta=2, average='macro')
            }

            test_metrics = {
                "Accuracy_test": accuracy_score(y_test, y_pred_test),
                "Precision_test_macro": precision_score(y_test, y_pred_test, average='macro'),
                "Recall_test_macro": recall_score(y_test, y_pred_test, average='macro'),
                "F1_score_test_macro": f1_score(y_test, y_pred_test, average='macro'),
                "F2_score_test_macro": fbeta_score(y_test, y_pred_test, beta=2, average='macro')
            }

            tuning_metrics = {"hyper_parameter_tuning_best_est_score": hyper_tuning}

            # Compute AUC metrics for multi-class (macro-average)
            train_fpr, train_tpr, train_pr, train_re, train_roc_auc, train_pr_auc = \
                roc_curve(y_train, model.predict_proba(X_train)[:, 1])
            test_fpr, test_tpr, test_pr, test_re, test_roc_auc, test_pr_auc = \
                roc_curve(y_test, model.predict_proba(X_test)[:, 1])

            # Log AUC metrics (Macro-average AUC)
            if train_roc_auc is not None:
                train_metrics["Roc_auc_train_macro"] = np.mean(list(train_roc_auc.values))
                mlflow.log_metric("Roc_auc_train_macro", train_metrics["Roc_auc_train_macro"])
            if train_pr_auc is not None:
                train_metrics["Pr_auc_train_macro"] = np.mean(list(train_pr_auc.values))
                mlflow.log_metric("Pr_auc_train_macro", train_metrics["Pr_auc_train_macro"])
            if test_roc_auc is not None:
                test_metrics["Roc_auc_test_macro"] = np.mean(list(test_roc_auc.values))
                mlflow.log_metric("Roc_auc_test_macro", test_metrics["Roc_auc_test_macro"])
            if test_pr_auc is not None:
                test_metrics["Pr_auc_test_macro"] = np.mean(list(test_pr_auc.values))
                mlflow.log_metric("Pr_auc_test_macro", test_metrics["Pr_auc_test_macro"])

            # Log other metrics
            mlflow.log_metric("Accuracy_train", train_metrics["Accuracy_train"])
            mlflow.log_metric("Precision_train_macro", train_metrics["Precision_train_macro"])
            mlflow.log_metric("Recall_train_macro", train_metrics["Recall_train_macro"])
            mlflow.log_metric("F1_score_train_macro", train_metrics["F1_score_train_macro"])
            mlflow.log_metric("F2_score_train_macro", train_metrics["F2_score_train_macro"])

            mlflow.log_metric("Accuracy_test", test_metrics["Accuracy_test"])
            mlflow.log_metric("Precision_test_macro", test_metrics["Precision_test_macro"])
            mlflow.log_metric("Recall_test_macro", test_metrics["Recall_test_macro"])
            mlflow.log_metric("F1_score_test_macro", test_metrics["F1_score_test_macro"])
            mlflow.log_metric("F2_score_test_macro", test_metrics["F2_score_test_macro"])

            mlflow.log_metric("Roc_auc_train_macro", train_metrics["Roc_auc_train_macro"])
            mlflow.log_metric("Pr_auc_train_macro", train_metrics["Pr_auc_train_macro"])
            mlflow.log_metric("Roc_auc_test_macro", test_metrics["Roc_auc_test_macro"])
            mlflow.log_metric("Pr_auc_test_macro", test_metrics["Pr_auc_test_macro"])

            mlflow.log_metric("Hyperparameter_tuning", hyper_tuning)
            mlflow.log_metric("Tuning_time", tuning_time)

        except Exception as e:
            print(f"An error occurred: {e}")
            mlflow.log_error(str(e))

    mlflow.end_run()

mlflow_logging_and_metric_printing(model, run_name, bal_type, X_train, y_train, mlflow)
```

```

# Print metrics
print("Train Metrics:")
for key, value in train_metrics.items():
    print(f"{key}: {value:.4f}")
print("\nTest Metrics:")
for key, value in test_metrics.items():
    print(f"{key}: {value:.4f}")
print("\nTuning Metrics:")
for key, value in tuning_metrics.items():
    print(f"{key}: {value:.4f}")

# Classification Reports
train_clf_report = classification_report(y_train, y_pred_train)
test_clf_report = classification_report(y_test, y_pred_test)

# Print classification reports
print("\nTrain Classification Report:")
print(train_clf_report)
print("\nTest Classification Report:")
print(test_clf_report)

# Log metrics
mlflow.log_metrics(train_metrics)
mlflow.log_metrics(test_metrics)
mlflow.log_metrics(tuning_metrics)

# Convert classification reports to DataFrames
train_clf_report_dict = classification_report(y_train, y_pred_train, output_dict=True)
train_clf_report_df = pd.DataFrame(train_clf_report_dict).transpose()
test_clf_report_dict = classification_report(y_test, y_pred_test, output_dict=True)
test_clf_report_df = pd.DataFrame(test_clf_report_dict).transpose()

# Save classification reports and Log as artifacts
train_clf_report_df.to_csv(f"{run_name}_train_classification_report.csv")
mlflow.log_artifact(f"{run_name}_train_classification_report.csv")

test_clf_report_df.to_csv(f"{run_name}_test_classification_report.csv")
mlflow.log_artifact(f"{run_name}_test_classification_report.csv")

# Plot confusion matrices
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_train, y_pred_train)).plot(ax=axes[0])
axes[0].set_title('Train Confusion Matrix')

ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_test)).plot(ax=axes[1])
axes[1].set_title('Test Confusion Matrix')

plt.tight_layout()
plt.savefig(f'{run_name}_confusion_matrix.png')
mlflow.log_artifact(f'{run_name}_confusion_matrix.png')

# Log model
mlflow.sklearn.log_model(model, f'{run_name}_model')

except Exception as e:
    print(f"Error in mlflow_logging_and_metric_printing: {e}")

```

```

In [ ]: # Model details
name = "Tuned_Adaboost_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time

```

```

start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
multi_time_df,multi_feature_importance_df = log_time_and_feature_importances_df(mult
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes

```

Model: Tuned\_Adaboost\_on\_Imbalanced\_Dataset  
 params: {'algorithm': 'SAMME', 'learning\_rate': np.float64(0.7896910002727693), 'n\_estimators': 120}  
 Training Time: 190.1937 seconds  
 Testing Time: 7.5605 seconds  
 Tuning Time: 795.1099 seconds  
 Error in auc\_plots: multiclass format is not supported  
 Error in auc\_plots: multiclass format is not supported  
 Train Metrics:  
 Accuracy\_train: 1.0000  
 Precision\_train\_macro: 0.9997  
 Recall\_train\_macro: 1.0000  
 F1\_score\_train\_macro: 0.9999  
 F2\_score\_train\_macro: 0.9999  
 Test Metrics:  
 Accuracy\_test: 0.9988  
 Precision\_test\_macro: 0.9540  
 Recall\_test\_macro: 0.9320  
 F1\_score\_test\_macro: 0.9425  
 F2\_score\_test\_macro: 0.9361  
 Tuning Metrics:  
 hyper\_parameter\_tuning\_best\_est\_score: 0.9987

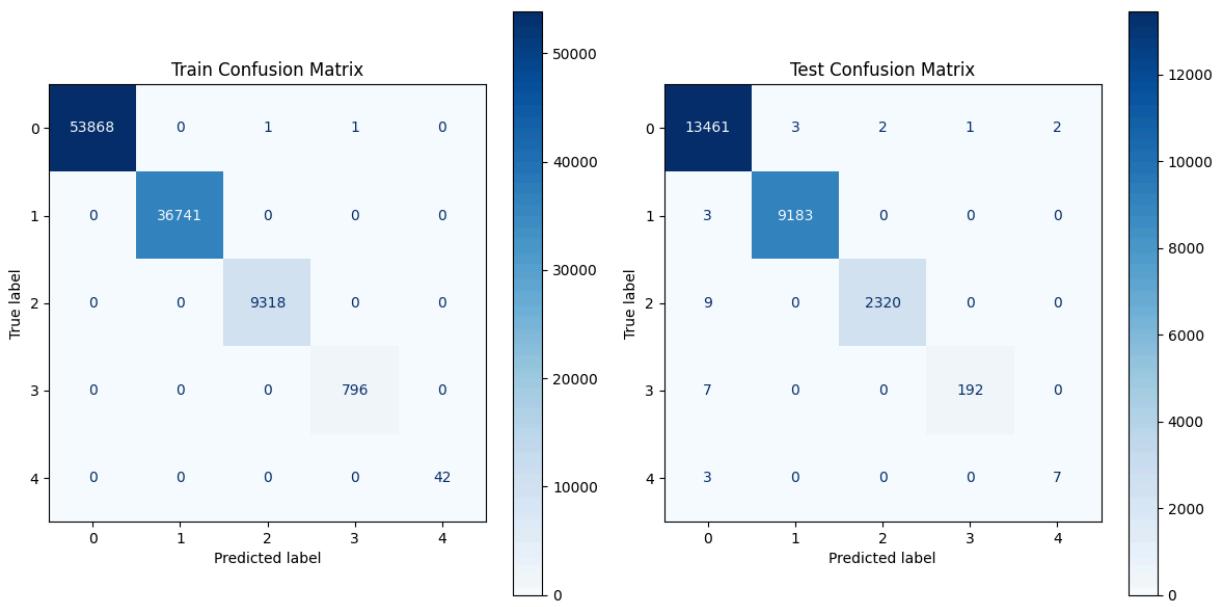
Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	36741
2	1.00	1.00	1.00	9318
3	1.00	1.00	1.00	796
4	1.00	1.00	1.00	42
accuracy			1.00	100767
macro avg	1.00	1.00	1.00	100767
weighted avg	1.00	1.00	1.00	100767

Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	9186
2	1.00	1.00	1.00	2329
3	0.99	0.96	0.98	199
4	0.78	0.70	0.74	10
accuracy			1.00	25193
macro avg	0.95	0.93	0.94	25193
weighted avg	1.00	1.00	1.00	25193

2025/05/17 08:54:20 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.



```
In [ ]: # Define the base estimator (Decision Stump)
base_stump = DecisionTreeClassifier(max_depth=5)

# Define the parameter grid
start_tune_time = time.time()
param_dist = {
    'n_estimators': stats.randint(100, 200), # Number of boosting stages
    'learning_rate': stats.uniform(0.01, 1.0), # Step size for boosting
    'algorithm': ['SAMME', 'SAMME.R'], # Algorithm to use for boosting
}

# Initialize the AdaBoost model with the decision stump as base estimator
ada_boost = AdaBoostClassifier(estimator=base_stump, random_state=42)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=ada_boost,
    param_distributions=param_dist,
    n_iter=2, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=1,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)
```

Fitting 5 folds for each of 2 candidates, totalling 10 fits  
 Best parameters found: {'algorithm': 'SAMME', 'learning\_rate': np.float64(0.7896910002727693), 'n\_estimators': 120}  
 Best score: 0.9995841841470205  
 Tuning\_time: 2821.958587169647

```
In [ ]: # Model details
name = "Tuned_Adaboost_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time
```

```

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
multi_time_df,multi_feature_importance_df = log_time_and_feature_importances_df(mult
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_tes

```

Model: Tuned\_Adaboost\_on\_Balanced\_Dataset  
 params: {'algorithm': 'SAMME', 'learning\_rate': np.float64(0.7896910002727693), 'n\_estimators': 120}  
 Training Time: 779.4589 seconds  
 Testing Time: 11.9286 seconds  
 Tuning Time: 2821.9586 seconds  
 Error in auc\_plots: multiclass format is not supported  
 Error in auc\_plots: multiclass format is not supported  
 Train Metrics:  
 Accuracy\_train: 1.0000  
 Precision\_train\_macro: 1.0000  
 Recall\_train\_macro: 1.0000  
 F1\_score\_train\_macro: 1.0000  
 F2\_score\_train\_macro: 1.0000  
  
 Test Metrics:  
 Accuracy\_test: 0.9988  
 Precision\_test\_macro: 0.9325  
 Recall\_test\_macro: 0.9352  
 F1\_score\_test\_macro: 0.9338  
 F2\_score\_test\_macro: 0.9346

Tuning Metrics:  
 hyper\_parameter\_tuning\_best\_est\_score: 0.9996

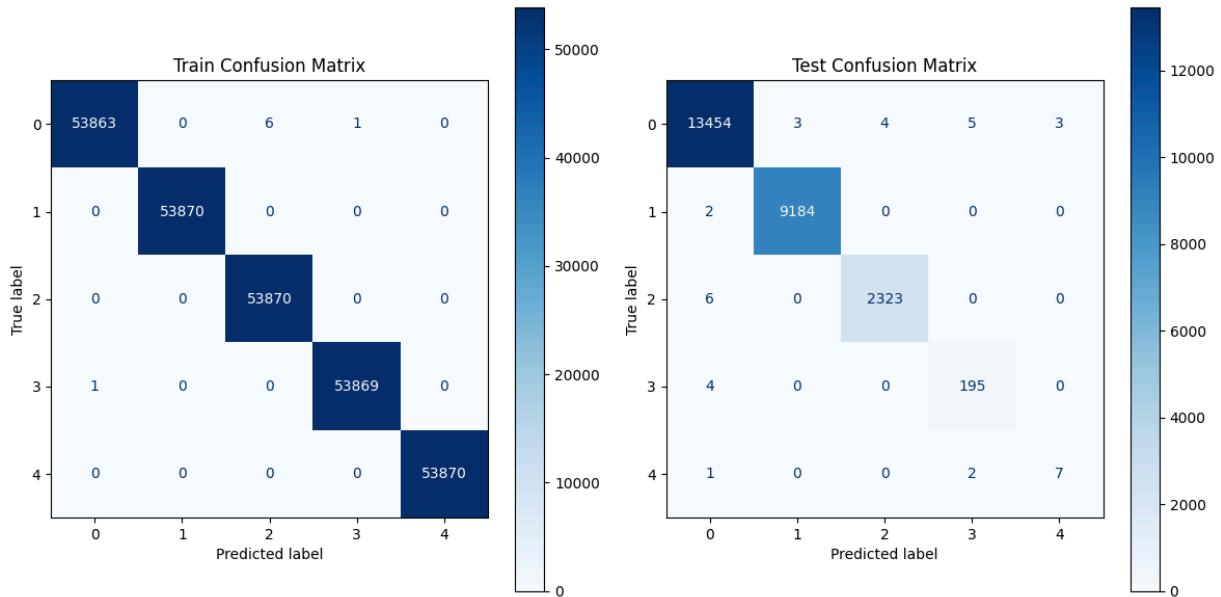
Train Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	53870
2	1.00	1.00	1.00	53870
3	1.00	1.00	1.00	53870
4	1.00	1.00	1.00	53870
accuracy			1.00	269350
macro avg	1.00	1.00	1.00	269350
weighted avg	1.00	1.00	1.00	269350

Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	9186
2	1.00	1.00	1.00	2329
3	0.97	0.98	0.97	199
4	0.70	0.70	0.70	10
accuracy			1.00	25193
macro avg	0.93	0.94	0.93	25193
weighted avg	1.00	1.00	1.00	25193

2025/05/17 09:55:11 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.



```
In [ ]: # Start timing the tuning process
start_tune_time = time.time()

# Define the parameter grid
param_dist = {
    'num_leaves': stats.randint(20, 150),
    'max_depth': stats.randint(3, 15),
    'learning_rate': stats.uniform(0.01, 0.3),
    'n_estimators': stats.randint(100, 200),
    'min_child_samples': stats.randint(10, 100),
    'min_child_weight': stats.uniform(1e-3, 1e-1),
    'subsample': stats.uniform(0.5, 1.0),
    'colsample_bytree': stats.uniform(0.5, 1.0),
    'reg_alpha': stats.uniform(0, 0.5),
    'reg_lambda': stats.uniform(0.5, 1.5),
    'scale_pos_weight': stats.uniform(0.5, 2),
    'boosting_type': ['gbdt', 'dart', 'goss'],
    'objective': ['binary', 'multiclass'],
    'bagging_fraction': stats.uniform(0.5, 1.0),
    'bagging_freq': stats.randint(1, 10),
    'feature_fraction': stats.uniform(0.5, 1.0),
    'min_split_gain': stats.uniform(0, 0.1),
    'min_data_in_leaf': stats.randint(20, 100),
    'random_state': [42],
}

# Initialize the LGBMClassifier
lgbm_clf = LGBMClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=lgbm_clf,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=False,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_imb, y_train_imb)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_

# End timing and print tuning time
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
```

```
print("Tuning_time:", tuning_time)
warnings.filterwarnings('ignore')
```

























```

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
Best parameters found: {'bagging_fraction': np.float64(0.8042422429595377), 'bagging_
freq': 6, 'boosting_type': 'gbdt', 'colsample_bytree': np.float64(0.931945018642115
8), 'feature_fraction': np.float64(0.7912291401980419), 'learning_rate': np.float64
(0.19355586841671385), 'max_depth': 12, 'min_child_samples': 69, 'min_child_weight':
np.float64(0.037636184329369174), 'min_data_in_leaf': 81, 'min_split_gain': np.float6
4(0.00906064345328208), 'n_estimators': 161, 'num_leaves': 70, 'objective': 'multicla
ss', 'random_state': 42, 'reg_alpha': np.float64(0.2571172192068058), 'reg_lambda': n
p.float64(1.3886218532930636), 'scale_pos_weight': np.float64(0.5929008254399954), 's
ubsample': np.float64(1.1075448519014384)}
Best score: 0.9987297467122371
Tuning_time: 230.33011984825134

```

```

In [ ]: # Model details
name = "Tuned_Light_GBM_on_Imbalanced_Dataset"
bal_type = "Imbalanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_imb, y_train_imb)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_imb_train = model.predict(X_train_imb)
y_pred_imb_test = model.predict(X_test_imb)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
multi_time_df,multi_feature_importance_df = log_time_and_feature_importances_df(mult
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_imb,y_train_imb,X_tes

```

























```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the
split requirements
[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6
[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6
Model: Tuned_Light_GBM_on_Imbalanced_Dataset
params: {'bagging_fraction': np.float64(0.8042422429595377), 'bagging_freq': 6, 'boos
ting_type': 'gbdt', 'colsample_bytree': np.float64(0.9319450186421158), 'feature_fra
ction': np.float64(0.7912291401980419), 'learning_rate': np.float64(0.1935558684167138
5), 'max_depth': 12, 'min_child_samples': 69, 'min_child_weight': np.float64(0.037636
184329369174), 'min_data_in_leaf': 81, 'min_split_gain': np.float64(0.009060643453282
08), 'n_estimators': 161, 'num_leaves': 70, 'objective': 'multiclass', 'random_stat
e': 42, 'reg_alpha': np.float64(0.2571172192068058), 'reg_lambda': np.float64(1.38862
18532930636), 'scale_pos_weight': np.float64(0.5929008254399954), 'subsample': np.flo
at64(1.1075448519014384)}
Training Time: 9.7409 seconds
Testing Time: 3.1568 seconds
Tuning Time: 230.3301 seconds
[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6
Error in auc_plots: multiclass format is not supported
[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6
Error in auc_plots: multiclass format is not supported
Train Metrics:
Accuracy_train: 1.0000
Precision_train_macro: 1.0000
```

```
Recall_train_macro: 1.0000
F1_score_train_macro: 1.0000
F2_score_train_macro: 1.0000
```

```
Test Metrics:
Accuracy_test: 0.9989
Precision_test_macro: 0.9447
Recall_test_macro: 0.9170
F1_score_test_macro: 0.9291
F2_score_test_macro: 0.9215
```

```
Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9987
```

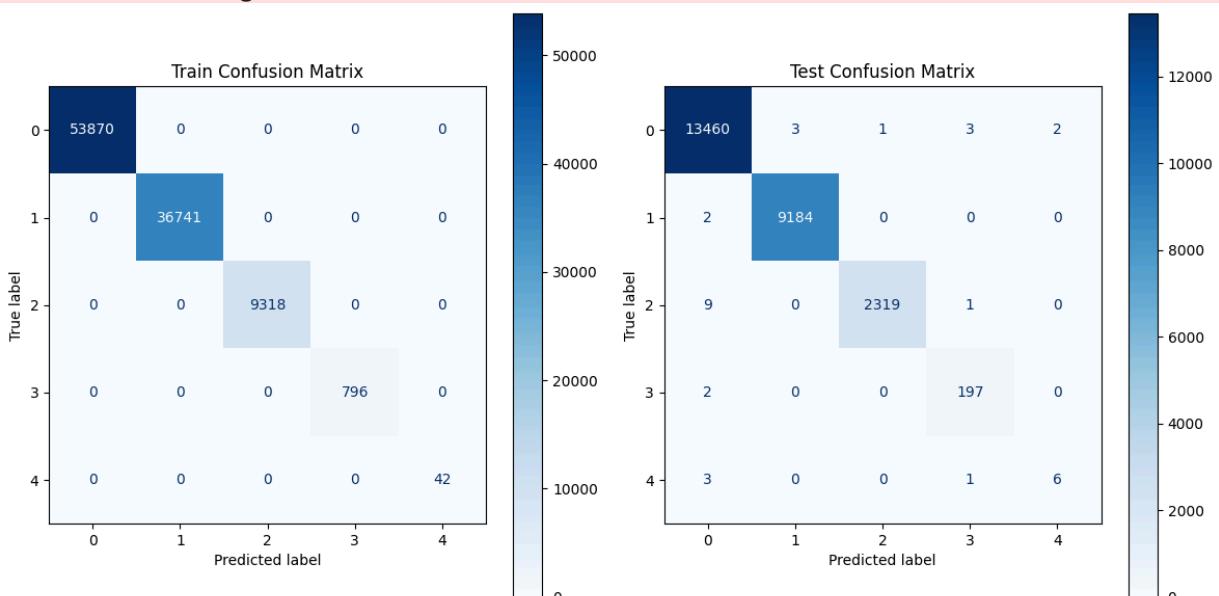
```
Train Classification Report:
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	53870
1	1.00	1.00	1.00	36741
2	1.00	1.00	1.00	9318
3	1.00	1.00	1.00	796
4	1.00	1.00	1.00	42
accuracy			1.00	100767
macro avg	1.00	1.00	1.00	100767
weighted avg	1.00	1.00	1.00	100767

```
Test Classification Report:
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	9186
2	1.00	1.00	1.00	2329
3	0.98	0.99	0.98	199
4	0.75	0.60	0.67	10
accuracy			1.00	25193
macro avg	0.94	0.92	0.93	25193
weighted avg	1.00	1.00	1.00	25193

```
2025/05/17 09:59:36 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.
```



```
In [ ]: # Start timing the tuning process
start_tune_time = time.time()

# Define the parameter grid
param_dist = {
    'num_leaves': stats.randint(20, 150),          # Number of Leaves in one tree
    'max_depth': stats.randint(3, 15),              # Maximum tree depth for base
    'learning_rate': stats.uniform(0.01, 0.3),       # Boosting Learning rate
    'n_estimators': stats.randint(100, 200),         # Number of boosting rounds
    'min_child_samples': stats.randint(10, 100),      # Minimum number of data need
```

```

'min_child_weight': stats.uniform(1e-3, 1e-1),
'subsample': stats.uniform(0.5, 1.0),
'colsample_bytree': stats.uniform(0.5, 1.0),
'reg_alpha': stats.uniform(0, 0.5),
'reg_lambda': stats.uniform(0.5, 1.5),
'scale_pos_weight': stats.uniform(0.5, 2),
'boosting_type': ['gbdt', 'dart', 'goss'],
'objective': ['binary', 'multiclass'],
'bagging_fraction': stats.uniform(0.5, 1.0),
'bagging_freq': stats.randint(1, 10),
'feature_fraction': stats.uniform(0.5, 1.0),
'min_split_gain': stats.uniform(0, 0.1),
'min_data_in_leaf': stats.randint(20, 100),
'random_state': [42],
}

# Initialize the LGBMClassifier
lgbm_clf = LGBMClassifier()

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=lgbm_clf,
    param_distributions=param_dist,
    n_iter=10, # Number of parameter settings to try
    cv=5, # Number of folds in cross-validation
    verbose=False,
    random_state=42,
    n_jobs=-1 # Use all available cores
)

# Fit RandomizedSearchCV
random_search.fit(X_train_bal, y_train_bal)

# Best model and hyperparameters
print("Best parameters found:", random_search.best_params_)
print("Best score:", random_search.best_score_)
tuning_score = random_search.best_score_

# End timing and print tuning time
end_tune_time = time.time()
tuning_time = end_tune_time - start_tune_time
print("Tuning_time:", tuning_time)
warnings.filterwarnings('ignore')

```























```
split requirements
Best parameters found: {'bagging_fraction': np.float64(0.8042422429595377), 'bagging_
freq': 6, 'boosting_type': 'gbdt', 'colsample_bytree': np.float64(0.931945018642115
8), 'feature_fraction': np.float64(0.7912291401980419), 'learning_rate': np.float64(
0.19355586841671385), 'max_depth': 12, 'min_child_samples': 69, 'min_child_weight': np.
float64(0.037636184329369174), 'min_data_in_leaf': 81, 'min_split_gain': np.float64(
0.00906064345328208), 'n_estimators': 161, 'num_leaves': 70, 'objective': 'multicla
ss', 'random_state': 42, 'reg_alpha': np.float64(0.2571172192068058), 'reg_lambda': n
p.float64(1.3886218532930636), 'scale_pos_weight': np.float64(0.5929008254399954), 's
ubsample': np.float64(1.1075448519014384)}
Best score: 0.9997141266010766
Tuning_time: 894.554042339325
```

```
In [ ]: # Model details
name = "Tuned_Light_GBM_on_Balanced_Dataset"
bal_type = "Balanced"
model = random_search.best_estimator_
params = random_search.best_params_

# Record training time
start_train_time = time.time()
model.fit(X_train_bal, y_train_bal)
end_train_time = time.time()

# Calculate training time
training_time = end_train_time - start_train_time

# Record testing time
start_test_time = time.time()
y_pred_bal_train = model.predict(X_train_bal)
y_pred_bal_test = model.predict(X_test_bal)
end_test_time = time.time()

# Calculate testing time
testing_time = end_test_time - start_test_time

# Print model name and times
print(f"Model: {name}")
print(f"params: {params}")
print(f"Training Time: {training_time:.4f} seconds")
print(f"Testing Time: {testing_time:.4f} seconds")
print(f"Tuning Time: {tuning_time:.4f} seconds")

# Logging time_df, feature_importance_df, mlflow_logging_and_metric_printing
multi_time_df,multi_feature_importance_df = log_time_and_feature_importances_df(mult
mlflow_logging_and_metric_printing(model,name,bal_type,X_train_bal,y_train_bal,X_tes
```























```

split requirements
[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6
[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6
Model: Tuned_Light_GBM_on_Balanced_Dataset
params: {'bagging_fraction': np.float64(0.8042422429595377), 'bagging_freq': 6, 'boos
ting_type': 'gbdt', 'colsample_bytree': np.float64(0.9319450186421158), 'feature_fra
ction': np.float64(0.7912291401980419), 'learning_rate': np.float64(0.1935558684167138
5), 'max_depth': 12, 'min_child_samples': 69, 'min_child_weight': np.float64(0.037636
184329369174), 'min_data_in_leaf': 81, 'min_split_gain': np.float64(0.009060643453282
08), 'n_estimators': 161, 'num_leaves': 70, 'objective': 'multiclass', 'random_stat
e': 42, 'reg_alpha': np.float64(0.2571172192068058), 'reg_lambda': np.float64(1.38862
18532930636), 'scale_pos_weight': np.float64(0.5929008254399954), 'subsample': np.flo
at64(1.1075448519014384)}
Training Time: 29.3865 seconds
Testing Time: 9.4865 seconds
Tuning Time: 894.5540 seconds
[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6
Error in auc_plots: multiclass format is not supported
[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6
Error in auc_plots: multiclass format is not supported
Train Metrics:
Accuracy_train: 1.0000
Precision_train_macro: 1.0000
Recall_train_macro: 1.0000
F1_score_train_macro: 1.0000
F2_score_train_macro: 1.0000

Test Metrics:
Accuracy_test: 0.9991
Precision_test_macro: 0.9466
Recall_test_macro: 0.9762
F1_score_test_macro: 0.9600
F2_score_test_macro: 0.9694

Tuning Metrics:
hyper_parameter_tuning_best_est_score: 0.9997

Train Classification Report:
      precision    recall   f1-score   support
          0         1.00     1.00     1.00      53870
          1         1.00     1.00     1.00      53870
          2         1.00     1.00     1.00      53870
          3         1.00     1.00     1.00      53870
          4         1.00     1.00     1.00      53870

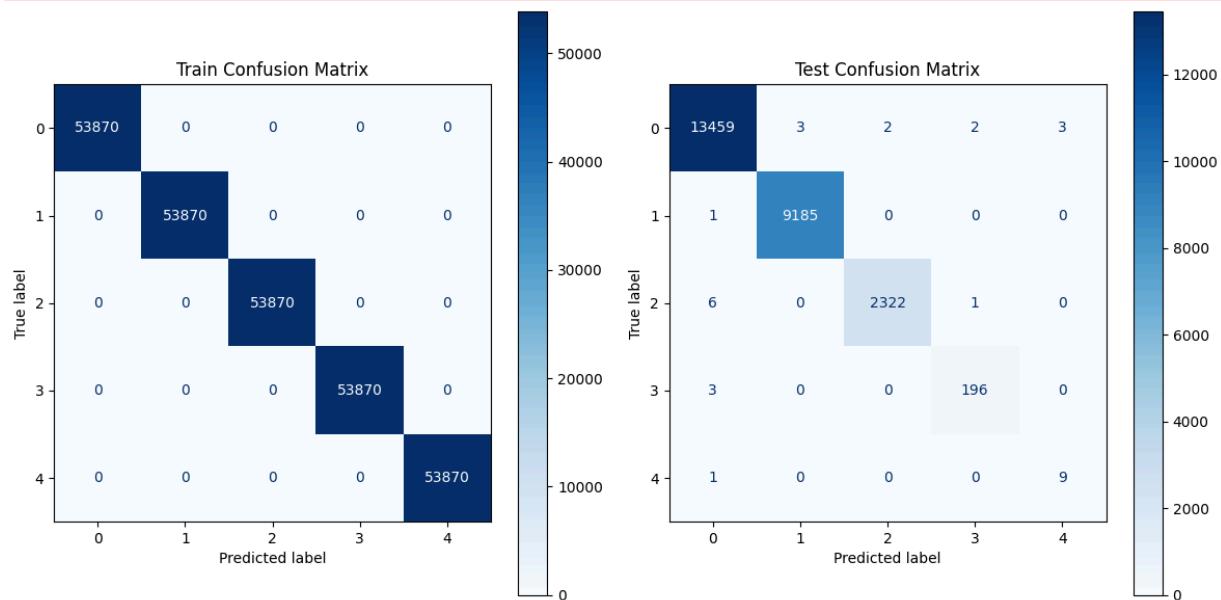
```

accuracy		1.00	1.00	1.00	269350
macro avg	1.00	1.00	1.00	1.00	269350
weighted avg	1.00	1.00	1.00	1.00	269350

Test Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13469
1	1.00	1.00	1.00	9186
2	1.00	1.00	1.00	2329
3	0.98	0.98	0.98	199
4	0.75	0.90	0.82	10
accuracy			1.00	25193
macro avg	0.95	0.98	0.96	25193
weighted avg	1.00	1.00	1.00	25193

2025/05/17 10:40:39 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.



## Multi-class Results Evaluation

In [ ]: multi\_time\_df

Out[ ]:

	Model	bal_type	Training_Time	Testing_Time	Tuning_Time
0	Tuned_Adaboost_on_Imbalanced_Dataset	Imbalanced	190.193719	7.560516	795.10
1	Tuned_Adaboost_on_Balanced_Dataset	Balanced	779.458915	11.928602	2821.95
2	Tuned_Light_GBM_on_Imbalanced_Dataset	Imbalanced	9.740856	3.156761	230.33
3	Tuned_Light_GBM_on_Balanced_Dataset	Balanced	29.386531	9.486482	894.55

In [ ]: multi\_feature\_importance\_df

Out[ ]:

	Tuned_Adaboost_on_Imbalanced_Dataset	Tuned_Adaboost_on_Balanced
duration	0.024551	
srcbytes	0.091851	
dstbytes	0.067384	
wrongfragment	0.022135	
urgent	0.000655	
...	...	
multi_one_class_svm_df	0.029372	
multi_dbSCAN_labels	0.007941	
multi_knn_kth_distance	0.029638	
multi_gmm_score	0.033016	
multi_kmeans_adv	0.004649	

62 rows × 4 columns

```
In [ ]: multi_time_df.to_csv("multi_time_df", index = False)
multi_feature_importance_df.to_csv("multi_feature_importance_df")
```

```
In [ ]: multi_time_df = pd.read_csv("multi_time_df")
multi_feature_importance_df = pd.read_csv("multi_feature_importance_df").set_index("")
multi_feature_importance_df.index.name = None
```

```
In [ ]: def all_logged_metrics():
    # Set the experiment name
    experiment_name = "nadp_multi"

    # Get the experiment details
    experiment = mlflow.get_experiment_by_name(experiment_name)
    experiment_id = experiment.experiment_id

    # Retrieve all runs from the experiment
    runs_df = mlflow.search_runs(experiment_ids=[experiment_id])

    # Extract metrics columns
    metrics_columns = [col for col in runs_df.columns if col.startswith("metrics.")]
    metrics_df = runs_df[metrics_columns]

    # Add run_name as a column
    metrics_df['run_name'] = runs_df['tags.mlflow.runName']
    metrics_df["bal_type"] = runs_df["params.bal_type"]

    # Combine all params into a dictionary
    params_columns = [col for col in runs_df.columns if col.startswith("params.")]
    metrics_df["params_dict"] = runs_df[params_columns].apply(lambda row: row.dropna)

    # Sort remaining columns alphabetically
    sorted_columns = sorted(metrics_columns)

    # Rearrange columns: first column is 'run_name', followed by 'bal_type', 'params
    ordered_columns = ['run_name', "bal_type", "params_dict"] + sorted_columns
    metrics_df = metrics_df[ordered_columns]

    # If you want to view it in a more readable format
    return metrics_df
```

```
In [ ]: all_logged_metrics_df = all_logged_metrics()
all_logged_metrics_df
```

Out[ ]:

	run_name	bal_type	params_dict	metrics.AIC
0	Tuned_Light_GBM_on_Balanced_Dataset	Balanced	{'params.subsample': '1.1075448519014384', 'pa...}	NaN
1	Tuned_Light_GBM_on_Imbalanced_Dataset	Imbalanced	{'params.subsample': '1.1075448519014384', 'pa...}	NaN
2	Tuned_Adaboost_on_Balanced_Dataset	Balanced	{'params.n_estimators': '120', 'params.bal_t...}	NaN
3	Tuned_Adaboost_on_Imbalanced_Dataset	Imbalanced	{'params.n_estimators': '120', 'params.bal_t...}	NaN
4	multi_kmeans_adv	None	{'params.random_state': '42', 'params.n_cluste...}	NaN
5	multi_gmm	None	{'params.random_state': '42', 'params.n_compon...}	-6.280841e+06
6	multi_knn	None	{'params.n_jobs': '-1', 'params.n_neighbors': ...}	NaN
7	multi_dbSCAN	None	{'params.n_jobs': '-1', 'params.eps': '0.5', '...}	NaN
8	multi_one_class_svm	None	{'params.verbose': '0', 'params.nu': '0.1'}	NaN
9	multi_robust_cov	None	{'params.random_state': '42', 'params.contamin...}	NaN
10	multi_iforest	None	{'params.n_estimators': '100', 'params.random_...}	NaN
11	multi_lof	None	{'params.n_jobs': '-1', 'params.n_neighbors': ...}	NaN
12	multi_ground_truth_umap	None	{}	NaN



In [ ]: all\_logged\_metrics\_df.fillna(value = 0, inplace=True)

In [ ]: all\_logged\_metrics\_df.head()

Out[ ]:

	run_name	bal_type	params_dict	metrics.AIC	m
0	Tuned_Light_GBM_on_Balanced_Dataset	Balanced	{'params.subsample': '1.1075448519014384', 'pa...}	0.0	
1	Tuned_Light_GBM_on_Imbalanced_Dataset	Imbalanced	{'params.subsample': '1.1075448519014384', 'pa...}	0.0	
2	Tuned_Adaboost_on_Balanced_Dataset	Balanced	{'params.n_estimators': '120', 'params.bal_t...}	0.0	
3	Tuned_Adaboost_on_Imbalanced_Dataset	Imbalanced	{'params.n_estimators': '120', 'params.bal_t...}	0.0	
4	multi_kmeans_adv	0	{'params.random_state': '42', 'params.n_cluste...}	0.0	



In [ ]: df = all\_logged\_metrics\_df.iloc[0:5][['run\_name', 'bal\_type', 'params\_dict', 'metrics.Accuracy\_test', 'metrics.Accuracy\_train', 'metrics.F1\_score\_test\_macro', 'metrics.F1\_score\_train\_macro',

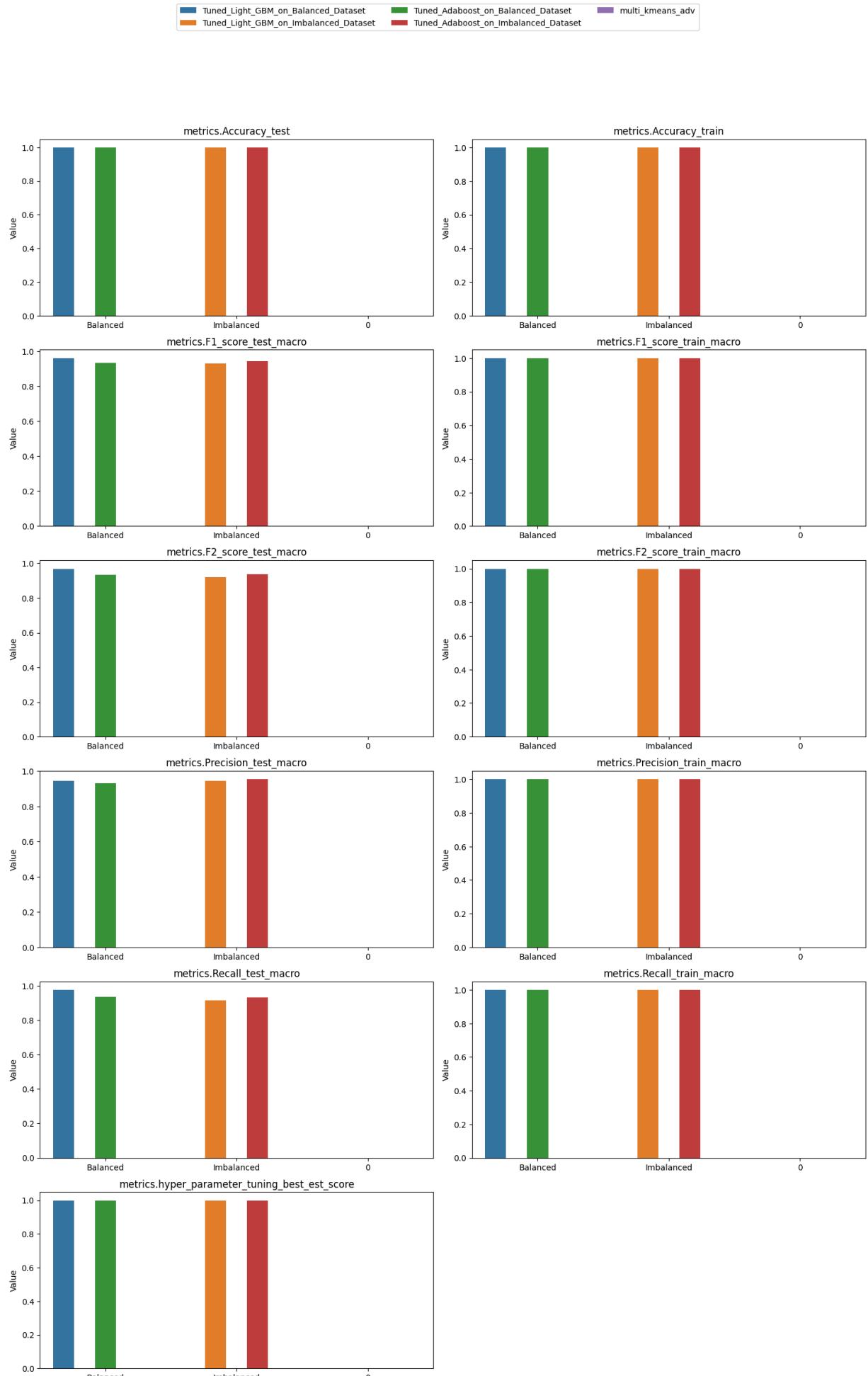
```
'metrics.F2_score_test_macro', 'metrics.F2_score_train_macro',
'metrics.Precision_test_macro', 'metrics.Precision_train_macro', 'metrics.Recall_train_macro',
'metrics.hyper_parameter_tuning_best_est_score']]
```

```
In [ ]: # Considering only classification metrics to plot
df.head()
```

```
Out[ ]:
```

	run_name	bal_type	params_dict	metrics.Accuracy
0	Tuned_Light_GBM_on_Balanced_Dataset	Balanced	{'params.subsample': '1.1075448519014384', 'pa...}	0.5
1	Tuned_Light_GBM_on_Imbalanced_Dataset	Imbalanced	{'params.subsample': '1.1075448519014384', 'pa...}	0.5
2	Tuned_Adaboost_on_Balanced_Dataset	Balanced	{'params.n_estimators': '120', 'params.bal_typ...}	0.5
3	Tuned_Adaboost_on_Imbalanced_Dataset	Imbalanced	{'params.n_estimators': '120', 'params.bal_typ...}	0.5
4	multi_kmeans_adv	0	{'params.random_state': '42', 'params.n_cluste...}	0.0

```
In [ ]: all_logged_metrics_df_plots(df)
```



```
In [ ]: # Assuming your DataFrame is named 'df'
metrics_columns = df.columns[df.columns.str.startswith('metrics.')]

for metric in metrics_columns:
    best_model = df.loc[df[metric].idxmax(), 'run_name']
    best_params = df.loc[df[metric].idxmax(), 'params_dict']
    print(f'{metric} -> best_model -> \'{best_model}\', best_params -> {best_params}')
```

```
metrics.Accuracy_test -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.791291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
```

```
metrics.Accuracy_train -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
```

```
metrics.F1_score_test_macro -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
```

```
metrics.F1_score_train_macro -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
```

```
metrics.F2_score_test_macro -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
```

```
metrics.F2_score_train_macro -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
```

```
metrics.Precision_test_macro -> best_model -> "Tuned_Adaboost_on_Imbalanced_Dataset", best_params -> {'params.n_estimators': '120', 'params.bal_type': 'Imbalanced', 'params.learning_rate': '0.7896910002727693', 'params.algorithm': 'SAMME'}
```

```
metrics.Precision_train_macro -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
```

```

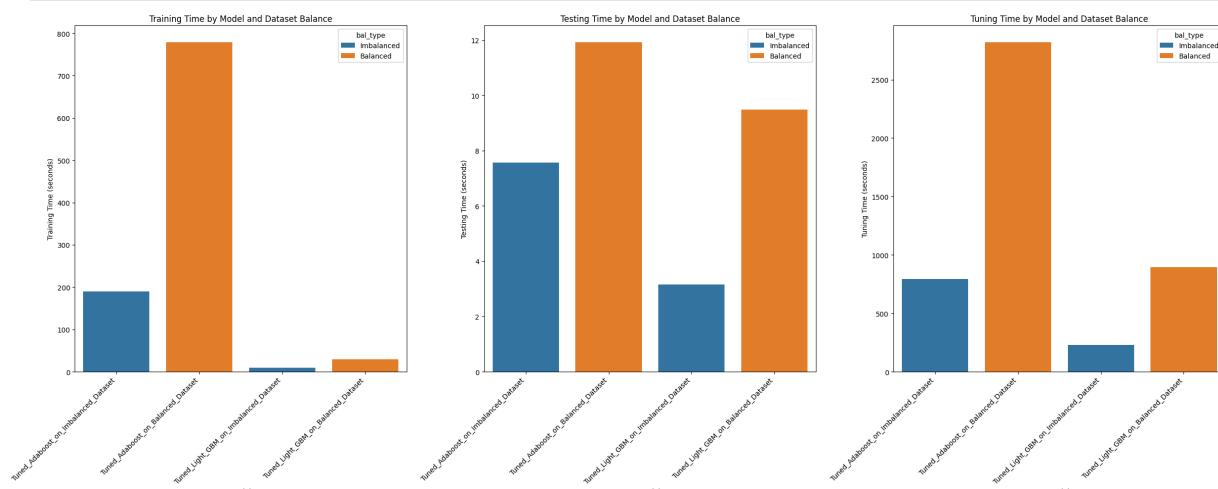
'6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
metrics.Recall_test_macro -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
metrics.Recall_train_macro -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}
metrics.hyper_parameter_tuning_best_est_score -> best_model -> "Tuned_Light_GBM_on_Balanced_Dataset", best_params -> {'params.subsample': '1.1075448519014384', 'params.feature_fraction': '0.7912291401980419', 'params.min_child_samples': '69', 'params.reg_lambda': '1.3886218532930636', 'params.objective': 'multiclass', 'params.min_child_weight': '0.037636184329369174', 'params.min_split_gain': '0.00906064345328208', 'params.bagging_freq': '6', 'params.min_data_in_leaf': '81', 'params.n_estimators': '161', 'params.num_leaves': '70', 'params.max_depth': '12', 'params.bal_type': 'Balanced', 'params.random_state': '42', 'params.scale_pos_weight': '0.5929008254399954', 'params.bagging_fraction': '0.8042422429595377', 'params.reg_alpha': '0.2571172192068058', 'params.colsample_bytree': '0.9319450186421158', 'params.boosting_type': 'gbdt', 'params.learning_rate': '0.19355586841671385'}

```

### Observation

Tuned\_Light\_GBM\_on\_Balanced\_Dataset model has best metrics . So we will use this model for deployment

In [ ]: time\_df\_plots(multi\_time\_df)



### Observation

Lightgbm is best in both accuary and latency

In [ ]: multi\_feature\_importance\_df

Out[ ]:

Tuned\_Adaboost\_on\_Imbalanced\_Dataset Tuned\_Adaboost\_on\_Balanced

<b>duration</b>	0.024551
<b>srcbytes</b>	0.091851
<b>dstbytes</b>	0.067384
<b>wrongfragment</b>	0.022135
<b>urgent</b>	0.000655
...	...
<b>multi_one_class_svm_df</b>	0.029372
<b>multi_dbSCAN_labels</b>	0.007941
<b>multi_knn_kth_distance</b>	0.029638
<b>multi_gmm_score</b>	0.033016
<b>multi_kmeans_adv</b>	0.004649

62 rows × 4 columns



In [ ]:

```
# normalizing the values
scaler = StandardScaler()
multi_feature_importance_scaled = pd.DataFrame(scaler.fit_transform(multi_feature_im-
                                                               index=multi_feature_importance_df.index,
                                                               columns=multi_feature_importance_df.columns)
multi_feature_importance_scaled
```

Out[ ]:

Tuned\_Adaboost\_on\_Imbalanced\_Dataset Tuned\_Adaboost\_on\_Balanced

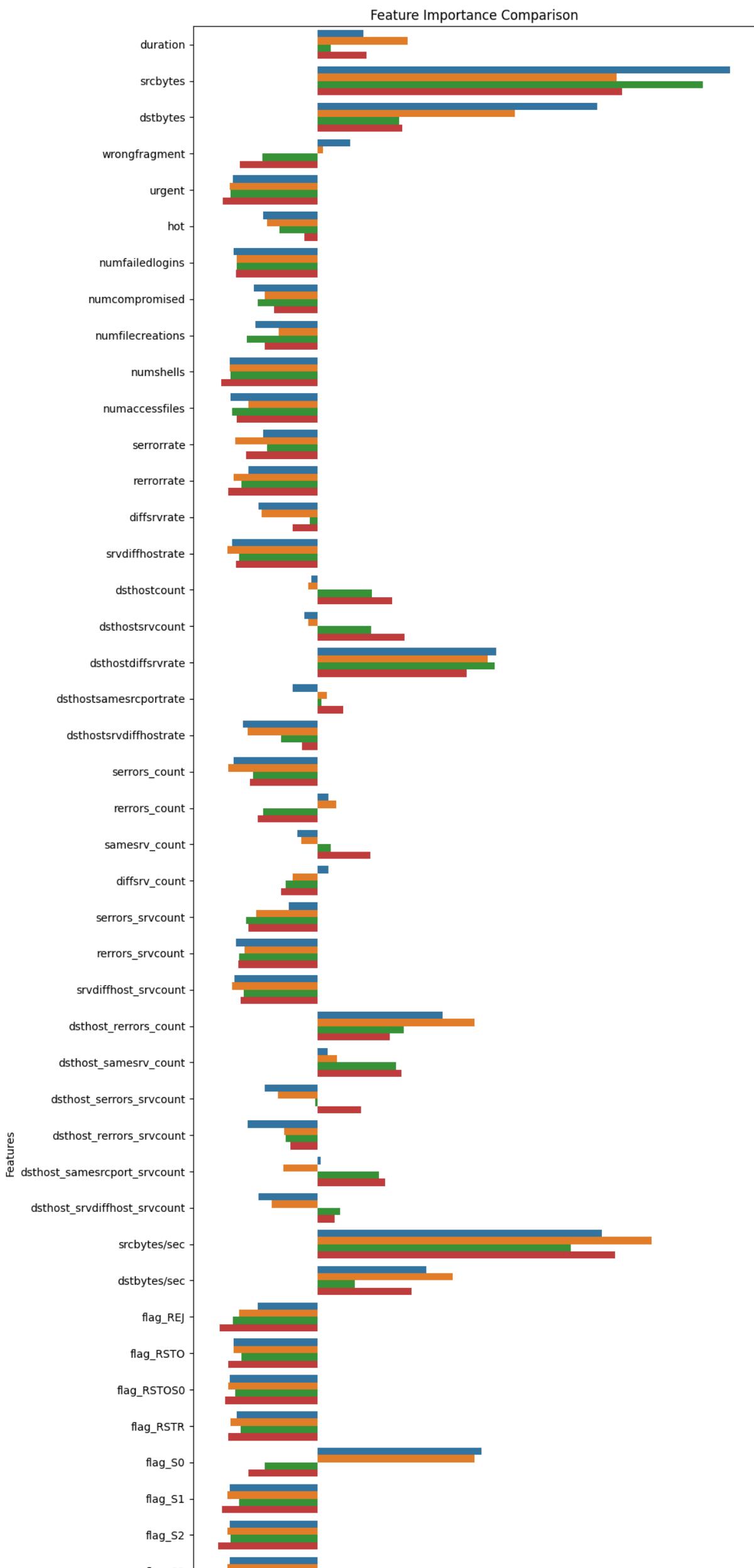
<b>duration</b>	0.419266
<b>srcbytes</b>	3.769738
<b>dstbytes</b>	2.551686
<b>wrongfragment</b>	0.298983
<b>urgent</b>	-0.770362
...	...
<b>multi_one_class_svm_df</b>	0.659302
<b>multi_dbSCAN_labels</b>	-0.407652
<b>multi_knn_kth_distance</b>	0.672536
<b>multi_gmm_score</b>	0.840717
<b>multi_kmeans_adv</b>	-0.571543

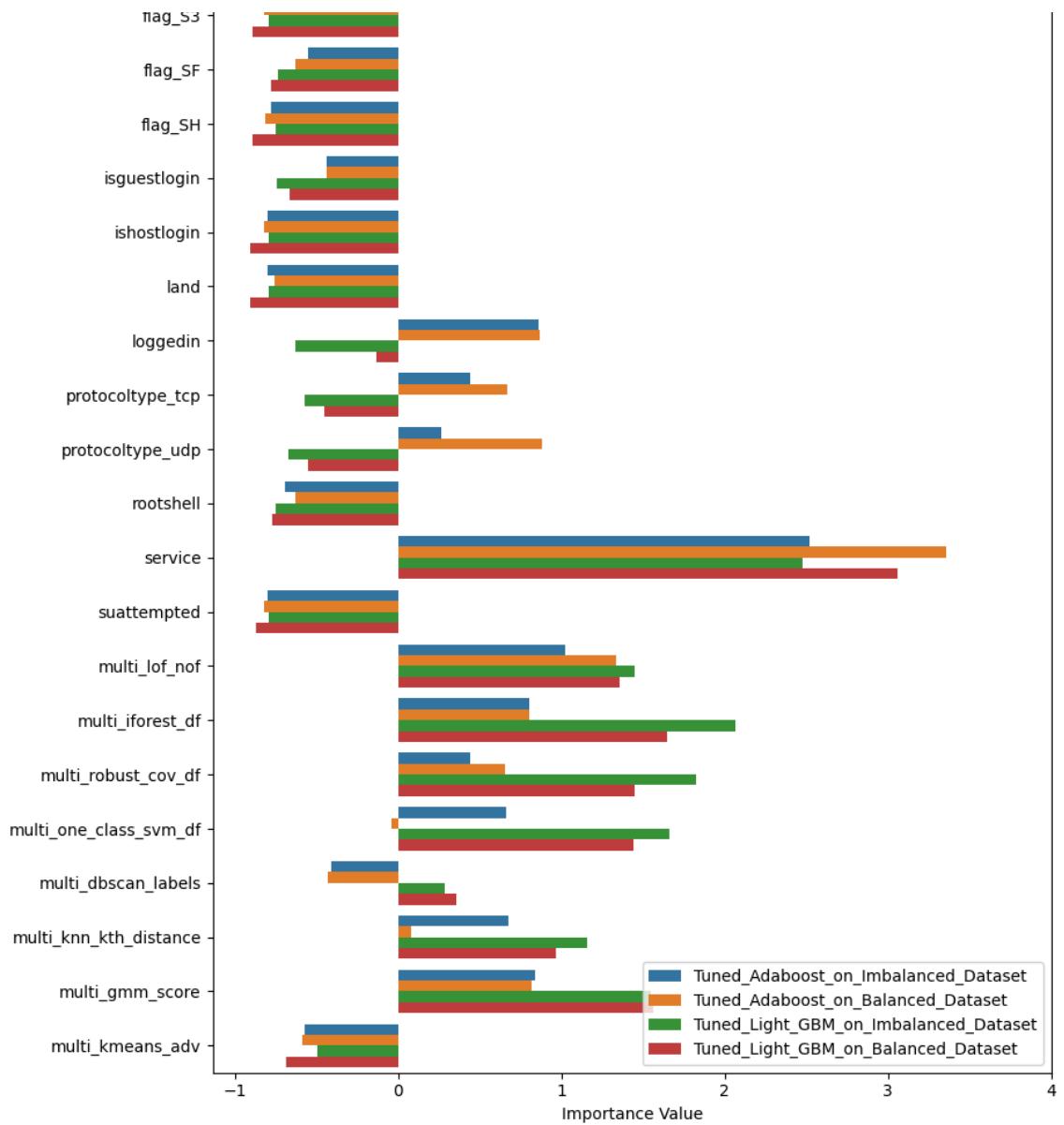
62 rows × 4 columns



In [ ]:

```
feature_importance_plots(multi_feature_importance_scaled)
```





```
In [ ]: combined_multi_feature_importance_scaled = multi_feature_importance_scaled.sum(axis=1)
In [ ]: combined_multi_feature_importance_scaled = pd.DataFrame(combined_multi_feature_importance_scaled)
In [ ]: combined_multi_feature_importance_scaled
```

```
Out[ ]: combined_multi_feature_importances
```

duration	1.809901
srcbytes	12.800782
dstbytes	5.871747
wrongfragment	-0.859871
urgent	-3.229798
...	...
multi_one_class_svm_df	3.721592
multi_dbSCAN_labels	-0.196303
multi_knn_kth_distance	2.875464
multi_gmm_score	4.766877
multi_kmeans_adv	-2.343216

62 rows × 1 columns

```
In [ ]: scaler = StandardScaler()
combined_multi_feature_importance_scaled = pd.DataFrame(scaler.fit_transform(combined_multi_feature_importance_scaled),
index=combined_multi_feature_importance_scaled.index,
columns = combined_multi_feature_importance_scaled.columns)
```

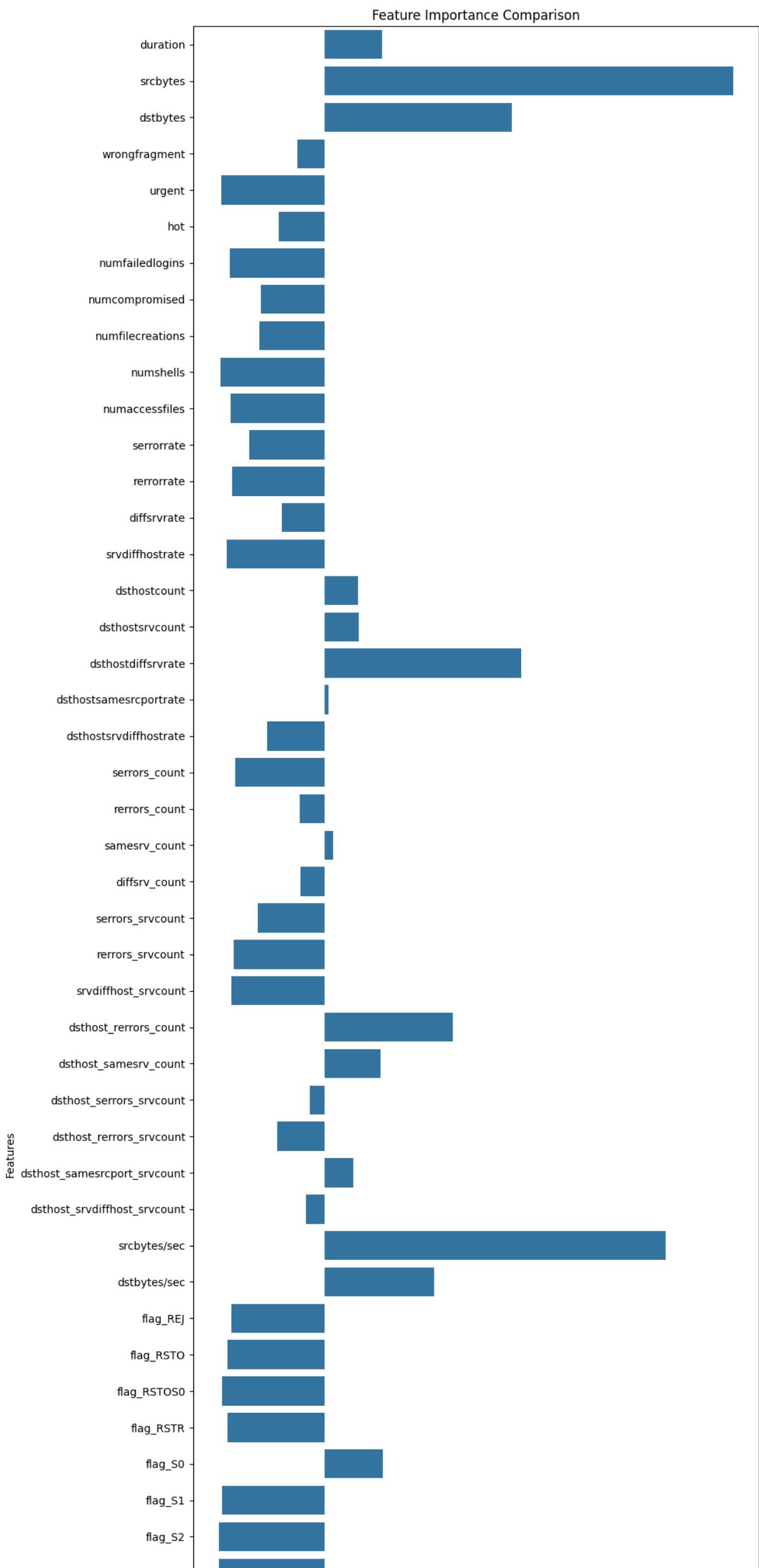
Out[ ]:

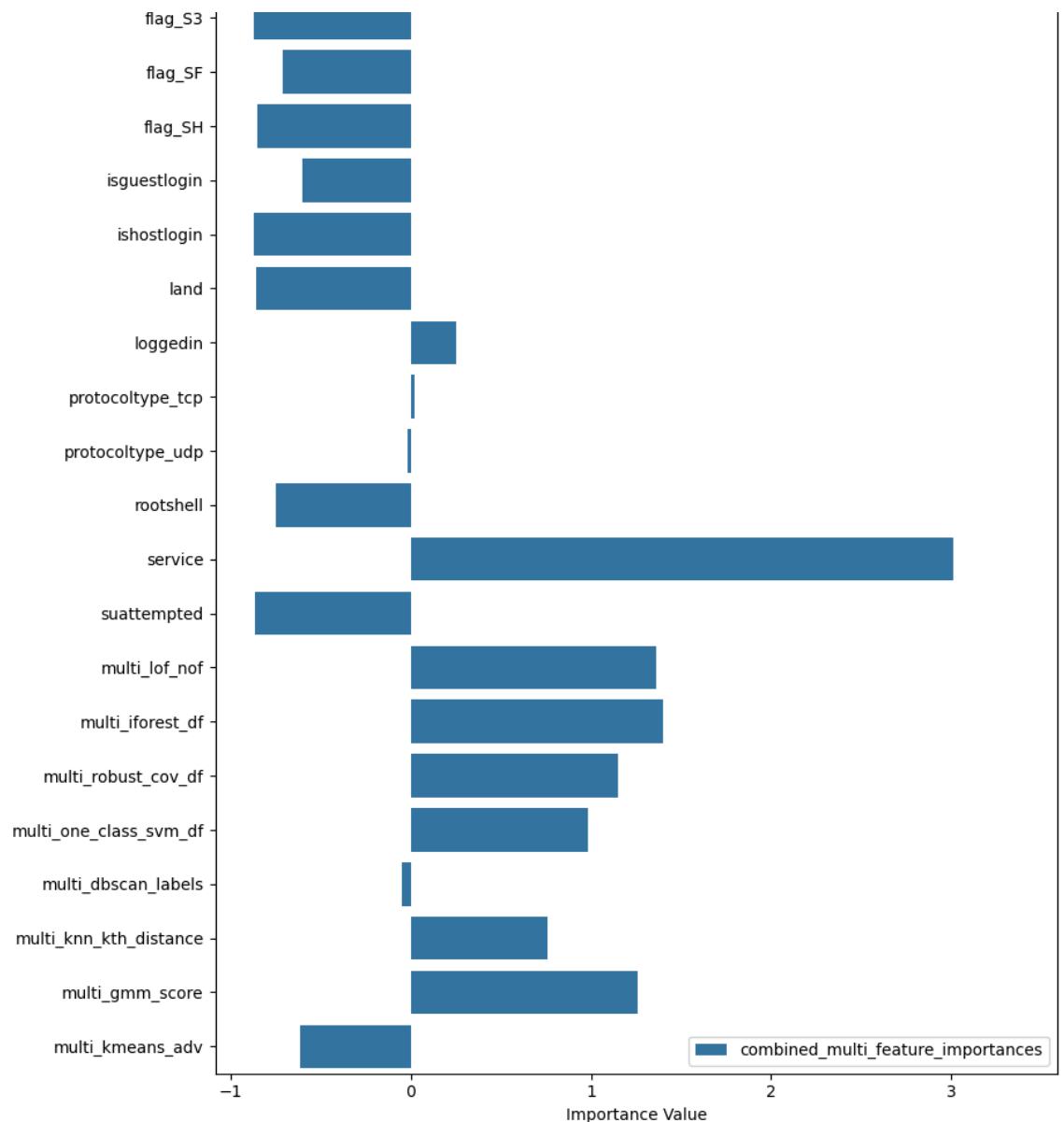
**combined\_multi\_feature\_importances**

<b>duration</b>	0.477609
<b>srcbytes</b>	3.377956
<b>dstbytes</b>	1.549476
<b>wrongfragment</b>	-0.226909
<b>urgent</b>	-0.852301
...	...
<b>multi_one_class_svm_df</b>	0.982079
<b>multi_dbSCAN_labels</b>	-0.051802
<b>multi_knn_kth_distance</b>	0.758797
<b>multi_gmm_score</b>	1.257916
<b>multi_kmeans_adv</b>	-0.618344

62 rows × 1 columns

In [ ]: `feature_importance_plots(combined_multi_feature_importance_scaled)`





```
In [ ]: combined_multi_feature_importance_scaled[np.abs(combined_multi_feature_importance_sc
```

Out[ ]:

combined_multi_feature_importances	
srcbytes	3.377956
service	3.011774
srcbytes/sec	2.817476
dsthosstdiffsrvrate	1.628276
dstbytes	1.549476
multi_iforest_df	1.402727
multi_lof_nof	1.362990
multi_gmm_score	1.257916
multi_robust_cov_df	1.152577
dsthosst_errors_count	1.062212
multi_one_class_svm_df	0.982079
dstbytes/sec	0.904081
flag_RSTR	-0.803400
srvdiffhostrate	-0.806441
flag_S1	-0.846989
flag_RSTOS0	-0.847164
urgent	-0.852301
flag_SH	-0.855680
land	-0.858555
numshells	-0.862957
suattempted	-0.866068
flag_S3	-0.871940
flag_S2	-0.874947
ishostlogin	-0.875257

## Observation

Above dataframe consists of Top 10 features according to combined normalised feature importances.

```
In [ ]: def get_experiment_id(experiment_name):
    experiment = mlflow.get_experiment_by_name(experiment_name)
    if experiment:
        return experiment.experiment_id
    else:
        print(f"Experiment '{experiment_name}' not found.")
        return None

def get_run_ids(experiment_id):
    client = mlflow.tracking.MlflowClient()
    runs = client.search_runs(experiment_id)
    return [run.info.run_id for run in runs]

# Example usage
experiment_id = get_experiment_id("nadb_multi")
run_ids = get_run_ids(experiment_id)
```

```
In [ ]: def print_all_classification_reports(experiment_id, run_ids):
    for run_id in run_ids:
        run_path = f"mlruns/{experiment_id}/{run_id}/artifacts/"
        for file in os.listdir(run_path):
```

```
if file.endswith("_classification_report.csv"):
    report_df = pd.read_csv(os.path.join(run_path, file), index_col=0)
    print(f"\n{file.replace('_classification_report.csv', '')}:\n")
    print(report_df)
    print("\n" + "-"*80 + "\n")

# Example usage
print_all_classification_reports(experiment_id, run_ids)
```

Tuned\_Light\_GBM\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.999183	0.999258	0.999220	13469.000000
1	0.999673	0.999891	0.999782	9186.000000
2	0.999139	0.996994	0.998066	2329.000000
3	0.984925	0.984925	0.984925	199.000000
4	0.750000	0.900000	0.818182	10.000000
accuracy	0.999127	0.999127	0.999127	0.999127
macro avg	0.946584	0.976214	0.960035	25193.000000
weighted avg	0.999146	0.999127	0.999134	25193.000000

---

Tuned\_Light\_GBM\_on\_Balanced\_Dataset\_train:

	precision	recall	f1-score	support
0	1.0	1.0	1.0	53870.0
1	1.0	1.0	1.0	53870.0
2	1.0	1.0	1.0	53870.0
3	1.0	1.0	1.0	53870.0
4	1.0	1.0	1.0	53870.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	269350.0
weighted avg	1.0	1.0	1.0	269350.0

---

Tuned\_Light\_GBM\_on\_Imbalanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.998813	0.999332	0.999072	13469.000000
1	0.999673	0.999782	0.999728	9186.000000
2	0.999569	0.995706	0.997634	2329.000000
3	0.975248	0.989950	0.982544	199.000000
4	0.750000	0.600000	0.666667	10.000000
accuracy	0.998928	0.998928	0.998928	0.998928
macro avg	0.944661	0.916954	0.929129	25193.000000
weighted avg	0.998912	0.998928	0.998916	25193.000000

---

Tuned\_Light\_GBM\_on\_Imbalanced\_Dataset\_train:

	precision	recall	f1-score	support
0	1.0	1.0	1.0	53870.0
1	1.0	1.0	1.0	36741.0
2	1.0	1.0	1.0	9318.0
3	1.0	1.0	1.0	796.0
4	1.0	1.0	1.0	42.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	100767.0
weighted avg	1.0	1.0	1.0	100767.0

---

Tuned\_Adaboost\_on\_Balanced\_Dataset\_test:

	precision	recall	f1-score	support
0	0.999035	0.998886	0.998960	13469.000000
1	0.999673	0.999782	0.999728	9186.000000
2	0.998281	0.997424	0.997852	2329.000000
3	0.965347	0.979899	0.972569	199.000000
4	0.700000	0.700000	0.700000	10.000000
accuracy	0.998809	0.998809	0.998809	0.998809
macro avg	0.932467	0.935198	0.933822	25193.000000
weighted avg	0.998813	0.998809	0.998811	25193.000000

---

```
Tuned_Adaboost_on_Balanced_Dataset_train:
```

	precision	recall	f1-score	support
0	0.999981	0.999870	0.999926	53870.00000
1	1.000000	1.000000	1.000000	53870.00000
2	0.999889	1.000000	0.999944	53870.00000
3	0.999981	0.999981	0.999981	53870.00000
4	1.000000	1.000000	1.000000	53870.00000
accuracy	0.999970	0.999970	0.999970	0.99997
macro avg	0.999970	0.999970	0.999970	269350.00000
weighted avg	0.999970	0.999970	0.999970	269350.00000

```
Tuned_Adaboost_on_Imbalanced_Dataset_test:
```

	precision	recall	f1-score	support
0	0.998368	0.999406	0.998887	13469.00000
1	0.999673	0.999673	0.999673	9186.00000
2	0.999139	0.996136	0.997635	2329.00000
3	0.994819	0.964824	0.979592	199.00000
4	0.777778	0.700000	0.736842	10.00000
accuracy	0.998809	0.998809	0.998809	0.998809
macro avg	0.953955	0.932008	0.942526	25193.00000
weighted avg	0.998800	0.998809	0.998802	25193.00000

```
Tuned_Adaboost_on_Imbalanced_Dataset_train:
```

	precision	recall	f1-score	support
0	1.000000	0.999963	0.999981	53870.00000
1	1.000000	1.000000	1.000000	36741.00000
2	0.999893	1.000000	0.999946	9318.00000
3	0.998745	1.000000	0.999372	796.00000
4	1.000000	1.000000	1.000000	42.00000
accuracy	0.999980	0.999980	0.999980	0.99998
macro avg	0.999728	0.999993	0.999860	100767.00000
weighted avg	0.999980	0.999980	0.999980	100767.00000

## Observation

Lightgbm performs well in detecting all the attack acategories except R2L.

# CHAPTER 7: DEPLOYMENT OF THE OPTIMAL MODEL USING STREAMLIT

## Selection of best models from MLFLOW

### best\_binary\_classification\_model

```
In [ ]: # Set the experiment name
experiment_name = "nadb_binary"
experiment = mlflow.get_experiment_by_name(experiment_name)

# Search for the run with the given run_name
run_name = "Tuned_Adaboost_on_Balanced_Dataset"
runs = mlflow.search_runs(experiment_ids=[experiment.experiment_id], filter_string=f"run_name == '{run_name}'")

# Ensure a run was found
if not runs.empty:
    run_id = runs.iloc[0]["run_id"] # Get the first matched run ID
```

```

# Load the model from the run
model_uri = f"runs:{run_id}/{run_name}_model"
best_binary_classification_model = mlflow.sklearn.load_model(model_uri)

# Print confirmation
print(f"Model loaded from run ID: {run_id}")
else:
    print("No matching run found for the given run_name.")

```

Model loaded from run ID: aa28fe0b38a4453fbefc98b663a578d9

```
In [ ]: with open("best_binary_classification_model.pkl","wb") as f:
    pickle.dump(best_binary_classification_model,f)
```

## best\_multi\_classification\_model

```

In [ ]: # Set the experiment name
experiment_name = "nadp_multi"
experiment = mlflow.get_experiment_by_name(experiment_name)

# Search for the run with the given run_name
run_name = "Tuned_Light_GBM_on_Balanced_Dataset"
runs = mlflow.search_runs(experiment_ids=[experiment.experiment_id], filter_string=f

# Ensure a run was found
if not runs.empty:
    run_id = runs.iloc[0]["run_id"] # Get the first matched run ID

    # Load the model from the run
    model_uri = f"runs:{run_id}/{run_name}_model"
    best_multi_classification_model = mlflow.sklearn.load_model(model_uri)

    # Print confirmation
    print(f"Model loaded from run ID: {run_id}")
else:
    print("No matching run found for the given run_name.")

```

Model loaded from run ID: 2c2b1671176f46b7a019fa99470c2bea

```
In [ ]: with open("best_multi_classification_model.pkl","wb") as f:
    pickle.dump(best_multi_classification_model,f)
```

## Binary\_Classification\_Prediction Deployment function

```

In [ ]: import numpy as np
from sklearn.neighbors import LocalOutlierFactor
import pickle,gzip,warnings
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import LocalOutlierFactor
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor, NearestNeighbors
from sklearn.covariance import EllipticEnvelope
from sklearn.svm import OneClassSVM
from sklearn.mixture import GaussianMixture
from sklearn.cluster import DBSCAN,KMeans

# Load globally
with open('ohe_encoder.pkl', 'rb') as f:
    ohe_encoder = pickle.load(f)

with open('service_encoding.pkl', 'rb') as f:
    service_encoding = pickle.load(f)

with open('nadp_X_train_binary_scaler.pkl', 'rb') as f:
    nadp_X_train_binary_scaler = pickle.load(f)

with gzip.open('nadp_X_train_binary_final.pkl', 'rb') as f:

```

```

nadp_X_train_binary_final = pickle.load(f)

with open("train_binary_iforest.pkl","rb") as f:
    train_binary_iforest = pickle.load(f)

with open("train_binary_robust_cov.pkl","rb") as f:
    train_binary_robust_cov = pickle.load(f)

with open("train_binary_one_class_svm.pkl","rb") as f:
    train_binary_one_class_svm = pickle.load(f)

with open("train_binary_knn.pkl","rb") as f:
    train_binary_knn = pickle.load(f)

with open("train_binary_gmm.pkl","rb") as f:
    train_binary_gmm = pickle.load(f)

with open("k_means_scaler.pkl","rb") as f:
    k_means_scaler = pickle.load(f)

with open("kmeans_best.pkl","rb") as f:
    kmeans_best = pickle.load(f)

with open("kmeans_adv_scaler.pkl","rb") as f:
    kmeans_adv_scaler = pickle.load(f)

with open("best_binary_classification_model.pkl","rb") as f:
    best_binary_classification_model = pickle.load(f)

def label_clusters(labels, centroids, points, alpha, beta, gamma):
    """
    Labels clusters as normal or anomaly based on size, density, and extreme density
    """
    unique_labels = np.unique(labels[labels != -1])
    n_clusters = len(unique_labels)
    cluster_sizes = np.array([np.sum(labels == i) for i in unique_labels])
    N = len(points)
    anomaly_labels = np.full(labels.shape, 'normal')

    # Calculate within-cluster sum of squares
    within_cluster_sums = []
    for i in unique_labels:
        cluster_points = points[labels == i]
        centroid = centroids[i]
        sum_of_squares = np.sum(np.linalg.norm(cluster_points - centroid, axis=1)**2)
        within_cluster_sums.append(sum_of_squares / len(cluster_points) if len(cluster_points) > 0 else 0)

    median_within_sum = np.median(within_cluster_sums)

    # Label clusters based on the given conditions
    for i, label in enumerate(unique_labels):
        size = cluster_sizes[i]
        average_within_sum = within_cluster_sums[i]

        if size < alpha * (N / n_clusters):
            anomaly_labels[labels == label] = 'anomalous'
        elif average_within_sum > beta * median_within_sum:
            anomaly_labels[labels == label] = 'anomalous'
        elif average_within_sum < gamma * median_within_sum:
            anomaly_labels[labels == label] = 'anomalous'

    return anomaly_labels

# The existing function definition
def binary_classification_prediction(df):

    warnings.filterwarnings('ignore')

    # feature engineering
    # Time and host related features
    df['serrors_count'] = df['serrorrate']*df['count']
    df['rerrors_count'] = df['rerrorrate']*df['count']
    df['samesrv_count'] = df['samesrvrate']*df['count']
    df['diffsrv_count'] = df['diffsrvrate']*df['count']

```

```

df['servers_srvcount'] = df['srverrorrate']*df['srvcount']
df['errors_srvcount'] = df['srverrorrate']*df['srvcount']
df['srvdifffhost_srvcount'] = df['srvdifffhostrate']*df['srvcount']
df['dsthost_serrors_count'] = df['dsthosterrorrate']*df['dsthostcount']
df['dsthost_rrrors_count'] = df['dsthostrrrrorrate']*df['dsthostcount']
df['dsthost_samesrv_count'] = df['dsthostsamesrvrate']*df['dsthostcount']
df['dsthost_diffsrv_count'] = df['dsthostdiffsrvrate']*df['dsthostcount']
df['dsthost_serrors_srvcount'] = df['dsthostsrvserrorrate']*df['dsthostsrvcount']
df['dsthost_rrrors_srvcount'] = df['dsthostsrvrerrorrate']*df['dsthostsrvcount']
df['dsthost_samesrcport_srvcount'] = df['dsthostsamesrcportrate']*df['dsthostsrvc']
df['dsthost_srvdifffhost_srvcount'] = df['dsthostsrvdifffhostrate']*df['dsthostsrvc']

# drop numoutboundcmd
df = df.drop(["numoutboundcmds"], axis=1)

# Data speed features
df['srcbytes/sec'] = df.apply(lambda row: row['srcbytes'] / row['duration'] if row['duration'] != 0 else 0, axis=1)
df['dstbytes/sec'] = df.apply(lambda row: row['dstbytes'] / row['duration'] if row['duration'] != 0 else 0, axis=1)

# modify suattempted to binary
df["suattempted"] = df["suattempted"].apply(lambda x: 0 if x == 0 else 1)

# encoding categorical features using ohe_encoder and service_encoder
encoded_test_data = ohe_encoder.transform(df[['protocoltype', 'flag']])
encoded_df = pd.DataFrame(encoded_test_data, columns=ohe_encoder.get_feature_names())
df = pd.concat([df.drop(columns=['protocoltype', 'flag']), encoded_df], axis=1)
df['service'] = df['service'].map(service_encoding)

# For any new service types in the test dataset that weren't in the training set,
df["service"] = df['service'].fillna(70)

# Scale the test features using the same scaler
df_scaled = nadp_X_train_binary_scaler.transform(df)

# Convert the scaled test features back to a DataFrame
df = pd.DataFrame(df_scaled, columns=df.columns)

# Removing VIF features
VIF_reduced_columns = ['duration', 'srcbytes', 'dstbytes', 'wrongfragment', 'urgeness',
    'numfailedlogins', 'numcompromised', 'numfilecreations', 'numshells',
    'numaccessfiles', 'serrorrate', 'rerrorrate', 'diffsrvrate',
    'srvdifffhostrate', 'dsthostcount', 'dsthostsrvcount',
    'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
    'dsthostsrvdifffhostrate', 'serrors_count', 'rrrors_count',
    'samesrv_count', 'diffsrv_count', 'serrors_srvcount',
    'rrrors_srvcount', 'srvdifffhost_srvcount', 'dsthost_rrrors_count',
    'dsthost_samesrv_count', 'dsthost_serrors_srvcount',
    'dsthost_rrrors_srvcount', 'dsthost_samesrcport_srvcount',
    'dsthost_srvdifffhost_srvcount', 'srcbytes/sec', 'dstbytes/sec']

cat_features = ['flag_REJ', 'flag_RST0', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0',
    'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH', 'isguestlog',
    'ishostlogin', 'land', 'loggedin', 'protocoltype_tcp', 'protocoltype',
    'rootshell', 'service', 'suattempted']

final_selected_features = VIF_reduced_columns + cat_features
df = df[final_selected_features]

# Now, add the query point as an extended dataset for LOF calculation
query_point_df = df.copy(deep=True)
query_point = df.values # The query point from the DataFrame `df`
extended_data = np.vstack([nadb_X_train_binary_final, query_point]) # Extend the data

# Fit the LOF model on the extended data (training data + query point)
train_binary_lof = LocalOutlierFactor(n_neighbors=20, contamination="auto", n_jobs=-1)
extended_data_lof_labels = train_binary_lof.fit_predict(extended_data) # Fit on the extended data

# Extract the Negative Outlier Factor (NOF) for the last point (query point)
query_point_lof_nof = train_binary_lof.negative_outlier_factor_[-1] # Get the NOF for the query point

# Add the NOF as a new feature for the query point
df['binary_lof_nof'] = query_point_lof_nof # Only add the NOF for the query point

# Add the decision_function of train_binary_iforest
df["binary_iforest_df"] = train_binary_iforest.decision_function(query_point_df)

```

```

# Add the decision_function of train_binary_robust_cov
df["binary_robust_cov_df"] = train_binary_robust_cov.decision_function(query_point)

# Add the decision_function of train_binary_one_class_svm
df["binary_one_class_svm_df"] = train_binary_one_class_svm.decision_function(query_point)

# Fit the dbscan model on the extended data (training data + query point)
train_binary_dbscan = DBSCAN(eps = 0.5, min_samples=5,n_jobs=-1)
df["binary_dbscan_labels"] = train_binary_dbscan.fit_predict(extended_data)[-1]

# Add the kth neighbor distance using train_binary_knn
test_distances, test_indices = train_binary_knn.kneighbors(query_point_df)
df["binary_knn_kth_distance"] = test_distances[:, -1]

# Add the decision_function of train_binary_gmm
df["binary_gmm_score"] = train_binary_gmm.score_samples(query_point_df)

# Apply k_means_scaler transform
data_train = k_means_scaler.transform(df)
train_labels = label_clusters(kmeans_best.predict(df), kmeans_best.cluster_center)
df["binary_kmeans_adv"] = np.where(train_labels == "anomal", 1, 0)

# Apply Final scaling before classification algorithms
df_array = kmeans_adv_scaler.transform(df)
df = pd.DataFrame(df_array, columns=df.columns)

# Classification prediction
prediction = best_binary_classification_model.predict(df)

if prediction[0] == 0:
    return "NORMAL"
elif prediction[0] == 1:
    return "ATTACK"
return None

```

In [ ]: query\_point = nadp.drop(["attack", "lastflag"], axis = 1).iloc[[0]].reset\_index(drop=True)

In [ ]: print(binary\_classification\_prediction(query\_point))

NORMAL

## Multi-Class Classification Prediction Deployment Function

```

In [ ]: import numpy as np
from sklearn.neighbors import LocalOutlierFactor
import pickle,gzip,warnings
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import LocalOutlierFactor
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor, NearestNeighbors
from sklearn.covariance import EllipticEnvelope
from sklearn.svm import OneClassSVM
from sklearn.mixture import GaussianMixture
from sklearn.cluster import DBSCAN,KMeans
import lightgbm as lgb
from lightgbm import LGBMClassifier

# Load globally
with open('ohe_encoder_multi.pkl', 'rb') as f:
    ohe_encoder_multi = pickle.load(f)

with open('service_encoding_multi.pkl', 'rb') as f:
    service_encoding_multi = pickle.load(f)

with open('attack_category_encoding_multi.pkl','rb') as f:
    attack_category_encoding_multi = pickle.load(f)

with open('nadp_X_train_multi_scaler.pkl', 'rb') as f:
    nadp_X_train_multi_scaler = pickle.load(f)

```

```

with gzip.open('nadp_X_train_multi_final.pkl', 'rb') as f:
    nadp_X_train_multi_final = pickle.load(f)

with open("train_multi_iforest.pkl","rb") as f:
    train_multi_iforest = pickle.load(f)

with open("train_multi_robust_cov.pkl","rb") as f:
    train_multi_robust_cov = pickle.load(f)

with open("train_multi_one_class_svm.pkl","rb") as f:
    train_multi_one_class_svm = pickle.load(f)

with open("train_multi_knn.pkl","rb") as f:
    train_multi_knn = pickle.load(f)

with open("train_multi_gmm.pkl","rb") as f:
    train_multi_gmm = pickle.load(f)

with open("k_means_scaler_multi.pkl","rb") as f:
    k_means_scaler_multi = pickle.load(f)

with open("kmeans_best_multi.pkl","rb") as f:
    kmeans_best_multi = pickle.load(f)

with open("kmeans_adv_scaler_multi.pkl","rb") as f:
    kmeans_adv_scaler_multi = pickle.load(f)

with open("multi_smote.pkl","rb") as f:
    multi_smote = pickle.load(f)

with open("best_multi_classification_model.pkl","rb") as f:
    best_multi_classification_model = pickle.load(f)

# The existing function definition
def multi_classification_prediction(df):

    warnings.filterwarnings('ignore')

    # feature engineering
    # Time and host related features
    df['serrors_count'] = df['serrorrate']*df['count']
    df['rerrors_count'] = df['rerrorrate']*df['count']
    df['samesrv_count'] = df['samesrvrate']*df['count']
    df['diffsrv_count'] = df['diffsrvrate']*df['count']
    df['serrors_srvcount'] = df['srvserrorrate']*df['srvcount']
    df['rerrors_srvcount'] = df['srvrerrorrate']*df['srvcount']
    df['srvidffhost_srvcount'] = df['srvidffhostrate']*df['srvcount']
    df['dsthost_serrors_count'] = df['dsthosterrorrate']*df['dsthostcount']
    df['dsthost_rerrors_count'] = df['dsthostrrorrate']*df['dsthostcount']
    df['dsthost_samesrv_count'] = df['dsthostsamesrvrate']*df['dsthostcount']
    df['dsthost_diffsrv_count'] = df['dsthostdiffsrvrate']*df['dsthostcount']
    df['dsthost_serrors_srvcount'] = df['dsthostsrvserrorrate']*df['dsthostsrvcount']
    df['dsthost_rerrors_srvcount'] = df['dsthostsrvrrorrate']*df['dsthostsrvcount']
    df['dsthost_samesrcport_srvcount'] = df['dsthostsamesrcportrate']*df['dsthostsrc']
    df['dsthost_srvidffhost_srvcount'] = df['dsthostsrvdifffhostrate']*df['dsthostsrv']

    # drop numoutboundcmd
    df = df.drop(["numoutboundcmds"], axis=1)

    # Data speed features
    df['srcbytes/sec'] = df.apply(lambda row: row['srcbytes'] / row['duration'] if row['duration'] != 0 else 0, axis=1)
    df['dstbytes/sec'] = df.apply(lambda row: row['dstbytes'] / row['duration'] if row['duration'] != 0 else 0, axis=1)

    # modify suattempted to multi
    df["suattempted"] = df["suattempted"].apply(lambda x: 0 if x == 0 else 1)

    # encoding categorical features using ohe_encoder_multi and service_encoder
    encoded_test_data = ohe_encoder_multi.transform(df[['protcoltype', 'flag']])
    encoded_df = pd.DataFrame(encoded_test_data, columns=ohe_encoder_multi.get_features())
    df = pd.concat([df.drop(columns=['protcoltype', 'flag']), encoded_df], axis=1)
    df['service'] = df['service'].map(service_encoding_multi)

    # For any new service types in the test dataset that weren't in the training set

```

```

df["service"] = df['service'].fillna(70)

# Scale the test features using the same scaler
df_scaled = nadp_X_train_multi_scaler.transform(df)

# Convert the scaled test features back to a DataFrame
df = pd.DataFrame(df_scaled, columns=df.columns)

# Removing VIF features
VIF_reduced_columns = ['duration', 'srcbytes', 'dstbytes', 'wrongfragment', 'urg
    'numfailedlogins', 'numcompromised', 'numfilecreations', 'numshells',
    'numaccessfiles', 'serrorrate', 'rerrorrate', 'diffsrvrate',
    'srvdifffhostrate', 'dsthostcount', 'dsthostsrvcount',
    'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
    'dsthostsrvdiffhostrate', 'serrors_count', 'rerrors_count',
    'samesrv_count', 'diffsrv_count', 'serrors_srvcount',
    'rerrors_srvcount', 'srvdifffhost_srvcount', 'dsthost_rerrors_count',
    'dsthost_samesrv_count', 'dsthost_serrors_srvcount',
    'dsthost_rerrors_srvcount', 'dsthost_samesrcport_srvcount',
    'dsthost_srvdiffhost_srvcount', 'srcbytes/sec', 'dstbytes/sec']

cat_features = ['flag_REJ', 'flag_RST0', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0',
    'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH', 'isguestst
    'ishostlogin', 'land', 'loggedin', 'protocoltype_tcp', 'protocolt
    'rootshell', 'service', 'suattempted']

final_selected_features = VIF_reduced_columns + cat_features
df = df[final_selected_features]

# Now, add the query point as an extended dataset for LOF calculation
query_point_df = df.copy(deep = True)
query_point = df.values # The query point from the DataFrame `df`
extended_data = np.vstack([nadp_X_train_multi_final, query_point]) # Extend the

# Fit the LOF model on the extended data (training data + query point)
train_multi_lof = LocalOutlierFactor(n_neighbors=20, contamination="auto", n_jobs=-1)
extended_data_lof_labels = train_multi_lof.fit_predict(extended_data) # Fit on

# Extract the Negative Outlier Factor (NOF) for the last point (query point)
query_point_lof_nof = train_multi_lof.negative_outlier_factor_[-1] # Get the NOF

# Add the NOF as a new feature for the query point
df['multi_lof_nof'] = query_point_lof_nof # Only add the NOF for the query point

# Add the decision_function of train_multi_iforest
df["multi_iforest_df"] = train_multi_iforest.decision_function(query_point_df)

# Add the decision_function of train_multi_robust_cov
df["multi_robust_cov_df"] = train_multi_robust_cov.decision_function(query_point_df)

# Add the decision_function of train_multi_one_class_svm
df["multi_one_class_svm_df"] = train_multi_svm.decision_function(query_point_df)

# Fit the dbscan model on the extended data (training data + query point)
train_multi_dbscan = DBSCAN(eps = 0.5, min_samples=5, n_jobs=-1)
df["multi_dbscan_labels"] = train_multi_dbscan.fit_predict(extended_data)[-1] # Get the labels

# Add the kth neighbor distance using train_multi_knn
test_distances, test_indices = train_multi_knn.kneighbors(query_point_df)
df["multi_knn_kth_distance"] = test_distances[:, -1]

# Add the decision_function of train_multi_gmm
df["multi_gmm_score"] = train_multi_gmm.score_samples(query_point_df)

# Apply k_means_scaler_multi transform
data_train = k_means_scaler_multi.transform(df)
df["multi_kmeans_adv"] = kmeans_best_multi.predict(data_train)

# Apply Final scaling before classification algorithms
df_array = kmeans_adv_scaler_multi.transform(df)
df = pd.DataFrame(df_array, columns=df.columns)

# Classification prediction
prediction = best_multi_classification_model.predict(df)

# Reverse the dictionary for decoding

```

```

    decoding_dict = {v: k for k, v in attack_category_encoding_multi.items()}

    # Decode the prediction
    decoded_prediction = decoding_dict[prediction[0]]

    return decoded_prediction

```

In [ ]: a = multi\_classification\_prediction(query\_point)

```

[LightGBM] [Warning] min_data_in_leaf is set=81, min_child_samples=69 will be ignore
d. Current value: min_data_in_leaf=81
[LightGBM] [Warning] feature_fraction is set=0.7912291401980419, colsample_bytree=0.9
319450186421158 will be ignored. Current value: feature_fraction=0.7912291401980419
[LightGBM] [Warning] bagging_fraction is set=0.8042422429595377, subsample=1.10754485
19014384 will be ignored. Current value: bagging_fraction=0.8042422429595377
[LightGBM] [Warning] bagging_freq is set=6, subsample_freq=0 will be ignored. Current
value: bagging_freq=6

```

In [ ]: print(a)

```
Normal
```

## STREAM LIT CODE

In [ ]: # Import necessary libraries

```

import streamlit as st
import pandas as pd
import numpy as np
import pickle
import gzip

import streamlit as st
import numpy as np

# Define StreamLit app
st.title("Binary Classification Prediction for Network Intrusion")

# Create user input fields for each feature
st.subheader("Input the following network connection features:")

features = {
    'Duration': st.number_input(
        'Duration (seconds)', min_value=0.0, step=1.0,
        help='Duration of the connection in seconds'
    ),
    'Protocol Type': st.selectbox(
        'Protocol Type', ['tcp', 'udp', 'icmp'],
        help='Type of protocol used in the connection'
    ),
    'Service': st.text_input(
        'Service (e.g., http, smtp, ftp-data)',
        help='Service requested by the connection, e.g., HTTP or FTP'
    ),
    'Flag': st.selectbox(
        'Flag', ['SF', 'S0', 'REJ', 'RST0', 'RSTR', 'SH'],
        help='State of the connection (e.g., successful, reset)'
    ),
    'Source Bytes': st.number_input(
        'Source Bytes', min_value=0, step=1,
        help='Number of data bytes sent from source to destination'
    ),
    'Destination Bytes': st.number_input(
        'Destination Bytes', min_value=0, step=1,
        help='Number of data bytes sent from destination to source'
    ),
    'Land': st.selectbox(
        'Land (1 if equal, 0 otherwise)', [0, 1],
        help='Indicates if source and destination addresses are the same'
    ),
    'Wrong Fragment': st.number_input(
        'Wrong Fragment', min_value=0, step=1,
        help='Number of wrong fragments in the connection'
    ),
}

```

```

'Urgent Packets': st.number_input(
    'Urgent Packets', min_value=0, step=1,
    help='Number of urgent packets in the connection'
),
'HOT Indicators': st.number_input(
    'HOT Indicators', min_value=0, step=1,
    help='Number of hot indicators (e.g., failed login attempts)'
),
'Number of Failed Logins': st.number_input(
    'Number of Failed Logins', min_value=0, step=1,
    help='Number of failed login attempts during the connection'
),
'Logged In': st.selectbox(
    'Logged In (1 if yes, 0 otherwise)', [0, 1],
    help='Indicates if the user logged in successfully'
),
'Number of Compromised Conditions': st.number_input(
    'Number of Compromised Conditions', min_value=0, step=1,
    help='Number of compromised conditions during the connection'
),
'Root Shell': st.selectbox(
    'Root Shell (1 if obtained, 0 otherwise)', [0, 1],
    help='Indicates if a root shell was obtained'
),
'SU Attempted': st.selectbox(
    'SU Attempted (1 if yes, 0 otherwise)', [0, 1],
    help='Indicates if a superuser (SU) command was attempted'
),
'Number of Root Operations': st.number_input(
    'Number of Root Operations', min_value=0, step=1,
    help='Number of root accesses or operations performed'
),
'Number of File Creations': st.number_input(
    'Number of File Creations', min_value=0, step=1,
    help='Number of file creation operations during the connection'
),
'Number of Shells': st.number_input(
    'Number of Shells', min_value=0, step=1,
    help='Number of shell prompts invoked'
),
'Number of Access Files': st.number_input(
    'Number of Access Files', min_value=0, step=1,
    help='Number of times access to sensitive files was attempted'
),
'Number of Outbound Commands': st.number_input(
    'Number of Outbound Commands', min_value=0, step=1,
    help='Number of outbound commands in an FTP session'
),
'Is Host Login': st.selectbox(
    'Is Host Login (1 if yes, 0 otherwise)', [0, 1],
    help='Indicates if the login is for the host (1) or not (0)'
),
'Is Guest Login': st.selectbox(
    'Is Guest Login (1 if yes, 0 otherwise)', [0, 1],
    help='Indicates if the login was performed as a guest'
),
'Count': st.number_input(
    'Count (Number of connections to the same host)', min_value=0, step=1,
    help='Number of connections made to the same host in a given period'
),
'Srv Count': st.number_input(
    'Srv Count (Number of connections to the same service)', min_value=0, step=1,
    help='Number of connections made to the same service in a given period'
),
'S Error Rate': st.number_input(
    'S Error Rate', min_value=0.0, step=0.01,
    help='Percentage of connections that had SYN errors'
),
'Srv S Error Rate': st.number_input(
    'Srv S Error Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to the same service with SYN errors'
),
'R Error Rate': st.number_input(
    'R Error Rate', min_value=0.0, step=0.01,
    help='Percentage of connections that had RST errors'
)

```

```

        help='Percentage of connections that had REJ errors'
),
'Srv R Error Rate': st.number_input(
    'Srv R Error Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to the same service with REJ errors'
),
'Same Srv Rate': st.number_input(
    'Same Srv Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to the same service'
),
'Diff Srv Rate': st.number_input(
    'Diff Srv Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to different services'
),
'Srv Diff Host Rate': st.number_input(
    'Srv Diff Host Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to different hosts'
),
'Dst Host Count': st.number_input(
    'Dst Host Count', min_value=0, step=1,
    help='Number of connections made to the same destination host'
),
'Dst Host Srv Count': st.number_input(
    'Dst Host Srv Count', min_value=0, step=1,
    help='Number of connections made to the same service at the destination host'
),
'Dst Host Same Srv Rate': st.number_input(
    'Dst Host Same Srv Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to the same service at the destination host'
),
'Dst Host Diff Srv Rate': st.number_input(
    'Dst Host Diff Srv Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to different services at the destination host'
),
'Dst Host Same Src Port Rate': st.number_input(
    'Dst Host Same Src Port Rate', min_value=0.0, step=0.01,
    help='Percentage of connections from the same source port to the destination host'
),
'Dst Host Srv Diff Host Rate': st.number_input(
    'Dst Host Srv Diff Host Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to different hosts using the same service'
),
'Dst Host S Error Rate': st.number_input(
    'Dst Host S Error Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to the destination host with SYN errors'
),
'Dst Host Srv S Error Rate': st.number_input(
    'Dst Host Srv S Error Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to the same service at the destination host with SYN errors'
),
'Dst Host R Error Rate': st.number_input(
    'Dst Host R Error Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to the destination host with REJ errors'
),
'Dst Host Srv R Error Rate': st.number_input(
    'Dst Host Srv R Error Rate', min_value=0.0, step=0.01,
    help='Percentage of connections to the same service at the destination host with REJ errors'
)
}

# Create a dictionary to map the input labels to required column names
column_map = {
    'Duration': 'duration',
    'Protocol Type': 'protocoltype',
    'Service': 'service',
    'Flag': 'flag',
    'Source Bytes': 'srcbytes',
    'Destination Bytes': 'dstbytes',
    'Land': 'land',
    'Wrong Fragment': 'wrongfragment',
    'Urgent Packets': 'urgent',
    'Hot Indicators': 'hot',
    'Number of Failed Logins': 'numfailedlogins',
    'Logged In': 'loggedin',
}

```

```

'Number of Compromised Conditions': 'numcompromised',
'Root Shell': 'rootshell',
'SU Attempted': 'suattempted',
'Number of Root Operations': 'numroot',
'Number of File Creations': 'numfilecreations',
'Number of Shells': 'numshells',
'Number of Access Files': 'numaccessfiles',
'Number of Outbound Commands': 'numoutboundcmds',
'Is Host Login': 'ishostlogin',
'Is Guest Login': 'isguestlogin',
'Count': 'count',
'Srv Count': 'srvcount',
'S Error Rate': 'serrorrate',
'Srv S Error Rate': 'srvserrorrate',
'R Error Rate': 'rerrorrate',
'Srv R Error Rate': 'svrerrorrate',
'Same Srv Rate': 'samesrvrate',
'Diff Srv Rate': 'diffsrvrate',
'Srv Diff Host Rate': 'srvdifffhostrate',
'Dst Host Count': 'dsthostcount',
'Dst Host Srv Count': 'dsthostsrvcount',
'Dst Host Same Srv Rate': 'dsthostsamesrvrate',
'Dst Host Diff Srv Rate': 'dsthostdiffsrvrate',
'Dst Host Same Src Port Rate': 'dsthostsamesrcportrate',
'Dst Host Srv Diff Host Rate': 'dsthostsrvdifffhostrate',
'Dst Host S Error Rate': 'dsthosterrorrate',
'Dst Host Srv S Error Rate': 'dsthostsrvserrorrate',
'Dst Host R Error Rate': 'dsthosterrorrate',
'Dst Host Srv R Error Rate': 'dsthostsrverrorrate'
}

# Convert user input into a DataFrame
input_data = pd.DataFrame([features])

# Rename the columns
input_data.rename(columns=column_map, inplace=True)

# Display input data if needed
st.write("User Input:", input_data)

# Prediction button
if st.button("Predict Attack or Normal"):
    # Call the prediction function
    result = binary_classification_prediction(input_data)
    st.subheader(f"Prediction: {result}")

# Prediction button
if st.button("Predict Attack Category"):
    # Call the prediction function
    result = multi_classification_prediction(input_data)
    st.subheader(f"Attack Category: {result}")

```

```
2025-05-17 11:02:19.162 WARNING streamlit.runtime.scriptrunner_utils.script_run_content: Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:19.984
Warning: to view this Streamlit app on a browser, run it with the following command:

    streamlit run c:\Users\amaan\AppData\Local\Programs\Python\Python313\Lib\site-packages\ipykernel_launcher.py [ARGUMENTS]

2025-05-17 11:02:19.987 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:19.990 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:19.991 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:19.993 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:19.994 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:19.996 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.003 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.005 Session state does not function when running a script without `streamlit run`
2025-05-17 11:02:20.006 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.008 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.010 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.011 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.012 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.014 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.015 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.017 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.019 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.020 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.021 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.022 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.023 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.024 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.026 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.027 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.028 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.029 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.032 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.034 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.035 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.036 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.038 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-17 11:02:20.039 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
```













```
2025-05-17 11:02:20.515 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.518 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.519 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.520 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.522 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.523 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.524 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.526 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.527 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.528 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.529 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.  
2025-05-17 11:02:20.531 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
```

#### NOTE

- Combine the Binary\_Classification\_Prediction Deployment function , Multi-Class Classification Prediction Deployment Function and stream lit code and make it as 'stream\_lit\_app.py'
- And use streamlit run stream\_lit\_app.py in Command terminal
- Donot forget to pip freeze requirements.txt or add the required libraries manually incase of deployment errors.
- Make sure that all the required pickle files are place in main branch outside folder of github before deploying for smooth deployment.

## CHAPTER 8: CONCLUSION, ACTIONABLE INSIGHTS, RECOMMENDATIONS & FUTURE SCOPE

### CONCLUSION

In this project, we successfully developed a model for network anomaly detection, utilizing a variety of machine learning techniques to classify network traffic as either normal or anomalous. After extensive feature engineering, we employed unsupervised algorithms to enhance feature extraction and used binary and multi-class classification models to predict network anomalies. Among the models tested, Adaboost for binary classification and LightGBM for multi-class classification demonstrated the best performance in terms of accuracy and latency. We deployed the final models using Streamlit, allowing for real-time anomaly detection and prediction. Our findings can assist in improving network security by quickly identifying and mitigating malicious activities.

### ACTIONABLE INSIGHTS

1. **Anomaly Detection:** The Adaboost model for binary classification and LightGBM for multi-class classification can be used to detect network anomalies efficiently, helping network administrators identify potential attacks or intrusions with minimal latency. We have achieved 99% accuracy on the provided dataset.
2. **Time Complexity:** We have used clustering for feature engineering which increased the test time complexity. Particularly LOF, DBSCAN feature engineering takes around 60 seconds to get the related additional features. We can remove them to reduce the latency.
3. **Feature Importance:** Features such as error\_flag\_or\_not, urgent\_or\_not, and service have the most significant impact on anomaly detection. Srcbytes, diffsrvcount, dstbytes, Flag\_SF, service have high feature importance . Prioritizing these features in the data collection and monitoring process can improve detection accuracy. binary\_kmeans\_adv has better feature importance than multi\_kmeans\_adv. This says that sparse, dense cluster approach(alpha, beta, gamma approach) worked well for improving the classification performance.
4. **Real-Time Monitoring:** The deployment of the model via Streamlit can enable real-time monitoring of network traffic and alert administrators immediately when anomalies are detected, minimizing the response time to potential threats.

## RECOMMENDATIONS

1. **Model Optimization:** Further fine-tuning of hyperparameters for the Adaboost and LightGBM models, as well as exploring other algorithms like Random Forest or XGBoost, could improve model performance and robustness.
2. **Scalability:** As the dataset grows, consider implementing distributed computing solutions for training models on larger datasets and improving inference speed.
3. **Integration:** The model can be integrated into existing network monitoring systems or intrusion detection systems (IDS) to automate threat detection, improving network security posture.
4. **Data Quality:** Ensure continuous data collection with updated and diverse features, especially for categorical features such as service and protocoltype, which significantly affect model performance.

## FUTURE SCOPE

1. **Real-time Deployment at Scale:** Scaling the solution to handle a larger volume of network traffic in real-time,

potentially through cloud-based services, would provide broader applicability in enterprise environments.

2. **Adaptation to New Attack Patterns:** As new attack techniques and network behaviors emerge, it would be beneficial to retrain the models regularly with updated data to improve their ability to detect unknown anomalies.

3. **Deep Learning Models:** Exploring deep learning models such as Autoencoders for anomaly detection or Recurrent Neural Networks (RNNs) for sequential data could potentially enhance performance, especially for detecting complex attack patterns.

][po9780lpimage.png# CHAPTER 9 - REFERENCES

1. <https://www.scaler.com/topics/>
2. <https://woolf.university/library>
3. <https://kdd.ics.uci.edu/databases/kddcup99/task.html>
4. <https://web.archive.org/web/20150205070216/http://nsl.cs.unb.ca/NSL-KDD/>
5. <https://www.kaggle.com/datasets/hassan06/nslkdd/data>
6. Baek, S., Kwon, D., Suh, S. C., Kim, H., Kim, I., & Kim, J. (2021). Clustering-based label estimation for network anomaly detection. *Digital Communications and Networks*, 7(1), 37–44. <https://doi.org/10.1016/j.dcan.2020.06.001>