

CMPE 209 – Network Security and Applications

Homework2

First Name: Harish

Last Name: Marepalli

SJSU ID: 016707314

Professor: Dr. Younghee Park

TABLE OF CONTENTS

- 1. Differences between SDN and a traditional network**
- 2. Header information in OpenFlow 1.0**
- 3. OpenFlow Operation**
- 4. Three vulnerabilities in SDN and explain each attack**
- 5. Improving accuracy using Random Forest Algorithm**
- 6. Benefits of Machine Learning techniques for network intrusion detection**
- 7. Benefits of using NFV**
- 8. Conclusion**
- 9. Appendix**

1. Differences between SDN and a traditional network:

Software Defined Networking and traditional networks have main differences in their approach to network architecture and management. The differences are listed below:

- a. **Control and Data Plane Separation:** Traditional networks have a close relationship between the control and data planes, where switches and routers handle both traffic forwarding (data plane) and traffic control (control plane). On the other hand, SDN separates these planes, where a centralized controller manages the control plane and communicates with network devices to decide how traffic should be forwarded.
- b. **Programmability:** In conventional networks, configuring network devices is usually done via command-line interfaces (CLIs) or graphical user interfaces (GUIs), which can be tedious and prone to errors. Conversely, SDN offers a programmable interface that enables network administrators to automate network configuration and management via software.
- c. **Flexibility:** Typically, conventional networks are static, where network devices are configured to perform specific functions and are challenging to modify without substantial manual effort. Conversely, SDN provides more flexibility by enabling administrators to modify network behavior via software, making it simpler to adjust to evolving network needs.
- d. **Scalability:** Traditional networks may face scalability issues due to the intricacy of handling a vast number of network devices. SDN streamlines network management by consolidating control, rendering large-scale network management more manageable.

Overall, SDN provides greater control, flexibility, and scalability compared to traditional networks, making it a popular choice for modern network architectures.

Answer in table form:

Feature	Traditional Network	SDN
Control Plane	Tightly Coupled	Separated from Data Plane
Programmability	Limited	Highly Programmable
Flexibility	Limited	Highly Flexible
Scalability	Limited	Highly Scalable
Management	Device-based	Centralized Controller-based
Traffic Management	Static	Dynamic and Adaptive
Network Architecture	Hierarchical	Flat or Hybrid Architecture
Security	Device-based	Centralized and Policy-based

2. Header information in OpenFlow 1.0:

There are 12 tuples in the header part of the OpenFlow 1.0 and they are listed below:

- a. Ingress Port: The "Ingress Port" field of the OpenFlow 1.0 match structure specifies the physical port on the switch where a packet was received. It allows packets from different physical interfaces on the switch to be distinguished and treated differently as needed. Number of bits depends on the implementation. It is applicable to all packets received by the switch. It represents the physical port on the switch where the packet was received, using a numerical value starting at 1.
- b. Ethernet Source Address: This field indicates the source MAC address of the Ethernet frame. It can be used to match packets from a specific device or group of devices on the network. Number of bits are 48. Regardless of the type of data being transmitted or the specific protocol being used, the Ethernet source address is included in the Ethernet header of every packet sent over the network on an enabled Ethernet port.
- c. Ethernet Destination Address: This field indicates the destination MAC address of the Ethernet frame. It can be used to match packets destined for a specific device or group of devices on the network. Number of bits are 48. The Ethernet destination address is applicable to all packets sent over an enabled Ethernet port, regardless of the type of data being transmitted or the specific protocol being used. It is a fundamental part of the Ethernet protocol and is used to ensure that packets are delivered to the correct destination device on the network.
- d. Ethernet Type: This field indicates the type of the Ethernet frame, which specifies the protocol encapsulated in the payload. Examples of Ethernet types include IPv4, IPv6, ARP, and others. Number of bits are 16. The Ethernet Type field is applicable to all packets sent over an enabled Ethernet port, regardless of the type of data being transmitted or the specific protocol being used.
- e. VLAN id: This field indicates the VLAN identifier if the frame is VLAN-tagged. VLANs are used to logically separate network traffic into different broadcast domains and can be used to apply different policies or security measures to different VLANs. Number of bits are 12. The VLAN ID is applicable only to packets with Ethernet type 0x8100, which are VLAN-tagged Ethernet frames.
- f. VLAN priority: This field indicates the priority of the VLAN if the frame is VLAN-tagged. This priority can be used to prioritize traffic or apply quality of service (QoS) policies based on VLAN membership. Number of bits are 3. When it is specified that VLAN priority applies to all packets of Ethernet type 0x8100, it means that the VLAN priority setting will be applied to all VLAN-tagged frames, regardless of the specific VLAN ID or other VLAN-related settings.
- g. IP source address: This field indicates the source IP address of the IP packet, if present. This address can be used to match packets from a specific device or group of devices on the network. Number of bits are 3. By specifying that IP source address applies to all IP and ARP packets, it means that the source IP address field will be present and used in every IP and ARP packet, regardless of the specific protocol or other packet fields.
- h. IP destination address: This field indicates the destination IP address of the IP packet, if present. This address can be used to match packets destined for a specific device or group of devices on the network. Number of bits are 3. By specifying that IP destination address applies to all IP and ARP packets, it means that the destination IP address field will be present and used in every IP and ARP packet, regardless of the specific protocol or other packet fields.

- i. IP protocol: This field indicates the protocol identifier in the IP header, which specifies the protocol encapsulated in the payload. Examples of IP protocols include TCP, UDP, ICMP, and others. Number of bits are 8. By specifying that IP protocol applies to all IP and IP over Ethernet, ARP packets, it means that the protocol field will be present and used in every IP packet, regardless of the specific transport protocol used in the payload, as well as in IP over Ethernet packets and ARP packets.
- j. IP ToS bits: This field indicates the type of service bits in the IP header, which may indicate the priority or class of service. These bits can be used to apply QoS policies based on the type of traffic. Number of bits are 8. By specifying that IP ToS bits applies to all IP packets, it means that the ToS field will be present and used in every IP packet, regardless of its specific purpose or content.
- k. TCP/UDP source port/ Transport source port / ICMP Type: This field indicates the source port of the TCP or UDP packet, if present. This port can be used to match packets from a specific application or service. Number of bits are 16. By specifying that TCP/UDP source port applies to all TCP, UDP, and ICMP packets, it means that the source port field will be present and used in every IP packet that carries one of these protocols.
- l. TCP/UDP destination port/ Transport destination port/ICMP Code: This field indicates the destination port of the TCP or UDP packet, if present. This port can be used to match packets destined for a specific application or service. Number of bits are 16. By specifying that TCP/UDP destination port applies to all TCP, UDP, and ICMP packets, it means that the destination port field will be present and used in every IP packet that carries one of these protocols.

3. OpenFlow Operation:

The figure on page 24 in the mentioned pdf shows the Openflow operation.

OpenFlow is a communication protocol that enables communication between a software-defined networking (SDN) controller and an SDN switch, allowing the controller to program the switch's flow table to forward network traffic. In this scenario, we have a network with an SDN controller, two hosts (A and B), and an SDN switch with a flow table.

Initially, the flow table on the switch is empty, and there is no existing rule to handle traffic between host A and B. When host A sends a packet to host B, the SDN switch receives the packet and checks its flow table to determine how to forward it. Since there is no matching flow rule in the table, the switch sends a packet-in message to the SDN controller, requesting instructions on how to handle the packet.

Upon receiving the packet-in message, the SDN controller processes the packet and decides how to handle it based on its predefined policies and rules. It then sends a packet-out message to the switch, containing the necessary instructions to forward the packet to host B. The instructions specify the required actions to modify the packet header, such as setting the output port and destination address, and update the flow table with a new flow rule to handle subsequent packets between hosts A and B.

The switch then executes the instructions in the packet-out message, modifies the packet header accordingly, and forwards the packet to host B. It also updates its flow table with the new flow rule that matches the traffic between hosts A and B, ensuring that subsequent packets will be forwarded directly without the need for further communication with the controller.

This OpenFlow operation allows the SDN controller to dynamically program the switch's flow table to handle network traffic according to its policies and rules, providing centralized control and management of the network. It also enables flexible and efficient network configuration, as the controller can adapt to changing network conditions and traffic patterns in real-time.

4. Three vulnerabilities in SDN and explain each attack:

While software-defined networking (SDN) has brought significant changes to network architecture and management, it is not immune to vulnerabilities. In the following sections, we will discuss three different vulnerabilities and how they can be exploited:

1. **Control Plane Attacks:** SDN networks are vulnerable to various types of attacks, and one of the most significant is related to the control plane. The control plane is responsible for directing and managing network traffic flow by sending instructions to the data plane. Control plane attacks target the central SDN controller and can have severe consequences. Attackers can exploit vulnerabilities in the control plane by sending malicious packets to the controller, leading to a denial of service (DoS) attack or unauthorized network access. In certain instances, attackers can take control of the entire network by manipulating the flow rules in the controller.
2. **Switch Attacks:** In SDN networks, switches are responsible for forwarding packets and managing flow tables, making them a prime target for attackers. Switches are highly vulnerable to attacks, which can be initiated by sending malicious packets or compromising the switch's firmware. Once attacked, the switch's flow table can be manipulated, causing a DoS attack or redirecting traffic to an unintended destination.
3. **Man-in-the-Middle (MitM) Attacks:** Another vulnerability that SDN networks face is man-in-the-middle (MitM) attacks. These types of attacks happen when a malicious actor intercepts network traffic and modifies it for nefarious purposes. In SDN, MitM attacks can happen between the controller and the switches, or between switches themselves. By intercepting and modifying packets, attackers can redirect traffic to unintended locations, manipulate flow tables, or gain unauthorized access to sensitive information.

To mitigate these vulnerabilities, network administrators should implement proper security measures, such as encryption, access controls, and intrusion detection systems. Additionally, they should keep the SDN software up to date, use strong passwords, and implement proper network segmentation to limit the impact of an attack. It is also crucial to perform regular security audits and penetration testing to identify vulnerabilities and prevent attacks before they occur.

5. Improving accuracy using Random Forest Algorithm:

Here, we have been given two datasets, Train_data.csv and Test_data.csv. Train_data.csv is used here to train the model on how to detect whether an incoming traffic is normal or anomaly. In these datasets we there are so many columns. The Train_data.csv contains 42 columns and the Test_data.csv contains 41 columns. The difference of one column is the 'Class' column, which is used to detect normal and anomaly traffic.

Here, we were given the code to find the Random Forest algorithms output. By using the normal approach, the output achieved was 94% and we have to improve it to 98-99% by making modifications.

The procedure that I followed in this activity is I have installed jupyter labs and written the code in python notebook file. By using the normal code, I got the output as 94.7%.

First, to improve the accuracy, I have included 3 more columns of train dataset in the code and these columns are 'count', 'srv_count', and 'logged_in'. I have chosen only these columns is because I felt that these may contribute to the future prediction efficiently because remaining other column values are just 0's and 1's, which does not much contribute to the future prediction.

Some of the other columns might also contribute to the accurate prediction but we do not include more columns as it will be an overload/overfitting case. So, this is the reason I have included these columns.

Second, I have splitted the dataset into 70% and 30% to get the accurate result.

Third, in the modeling section, I have increased the max_depth from 2 to 7. The default value of number of trees is 100. The value of max_depth indicates the depth that we traverse for a tree. If the value of it is less, it means we are not deeply going through the random forest code. If the value is more, then we are diving deep into the process and can know how the process works and also the accurate output.

I have tried all the values of max depth from 2 to 7 and stopped at 7 because it is considered to be the limit. If the value is more than 7, then it will be considered as overfitting, which we don't want to do.

After making all these changes in the code, the output that I achieved is 0.9886213283937549 and that comes to be 98.8% which can be approximated to **99%**.

I have also written a few lines to display the confusion matrix and the matrix is as follows:

```
[[4017,    18],
 [   68,   3455]]
```

The scheme of confusion matrix is:

		Predicted condition	
		Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Here, the number of True positive (TP) values are 4017.

The number of False negative (FN) values are 18.

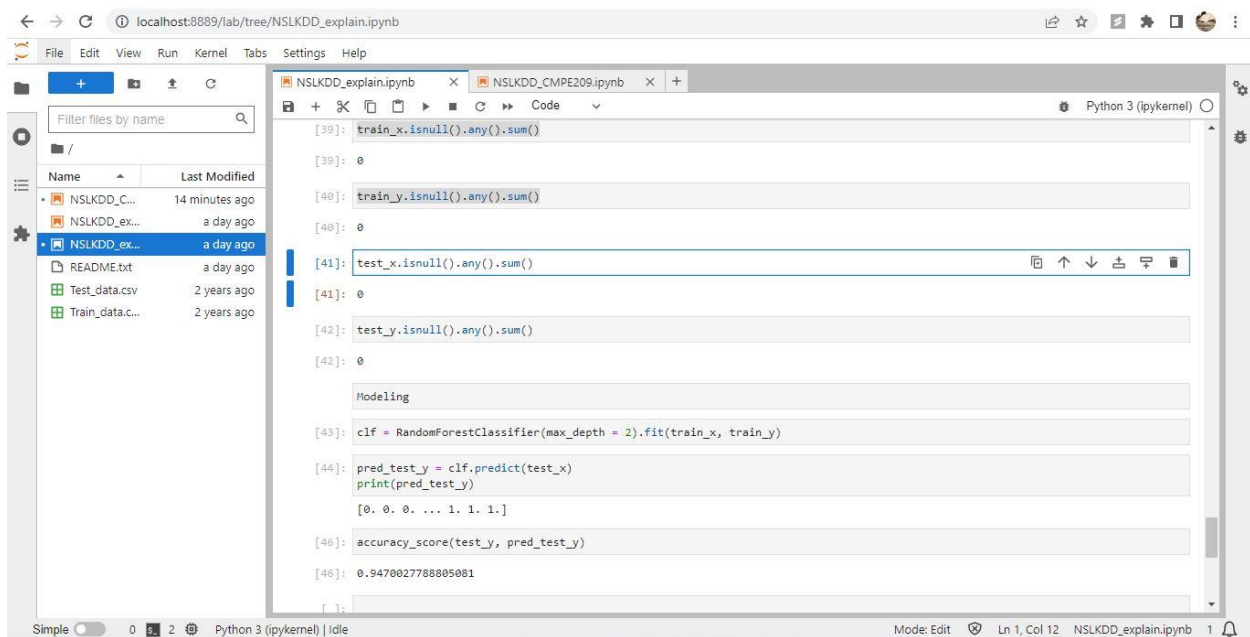
The number of False positive (FP) values are 68.

The number of True negative (TN) values are 3455.

Code is provided in the Appendix.

Snippets:

1. Figure 1 shows the output before the improvement.



```
[39]: train_x.isnull().any().sum()
[39]: 0
[40]: train_y.isnull().any().sum()
[40]: 0
[41]: test_x.isnull().any().sum()
[41]: 0
[42]: test_y.isnull().any().sum()
[42]: 0

Modeling
[43]: clf = RandomForestClassifier(max_depth = 2).fit(train_x, train_y)
[44]: pred_test_y = clf.predict(test_x)
      print(pred_test_y)
      [0. 0. 0. ... 1. 1. 1.]
[46]: accuracy_score(test_y, pred_test_y)
[46]: 0.9470027788805081
```

Fig. 1: Output before improvement

2. Figure 2 shows the improved code.

The screenshot shows a Jupyter Notebook with the following code cells:

```
[45]: import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import sklearn.preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

[46]: train_data = pd.read_csv('Train_data.csv')
test_data = pd.read_csv('Test_data.csv')

[47]: train_data_df = pd.DataFrame(train_data)
print(train_data_df.shape)

(25192, 42)

[48]: test_data_df = pd.DataFrame(test_data)
print(test_data_df.shape)

(22544, 41)

[49]: set(train_data_df).difference(set(test_data_df))

[49]: {'class'}

[50]: train_data_df.columns

[50]: Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
'dst_host_srv_count', 'dst_host_same_src_port_rate',
'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
'dst_host_srv_rerror_rate', 'class'],
dtype='object')
```

Fig. 2: Improved code 1

3. Figure 3 shows the continued code.

The screenshot shows a Jupyter Notebook with the following code cells:

```
[50]: Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
'dst_host_srv_count', 'dst_host_same_src_port_rate',
'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
'dst_host_srv_rerror_rate', 'class'],
dtype='object')

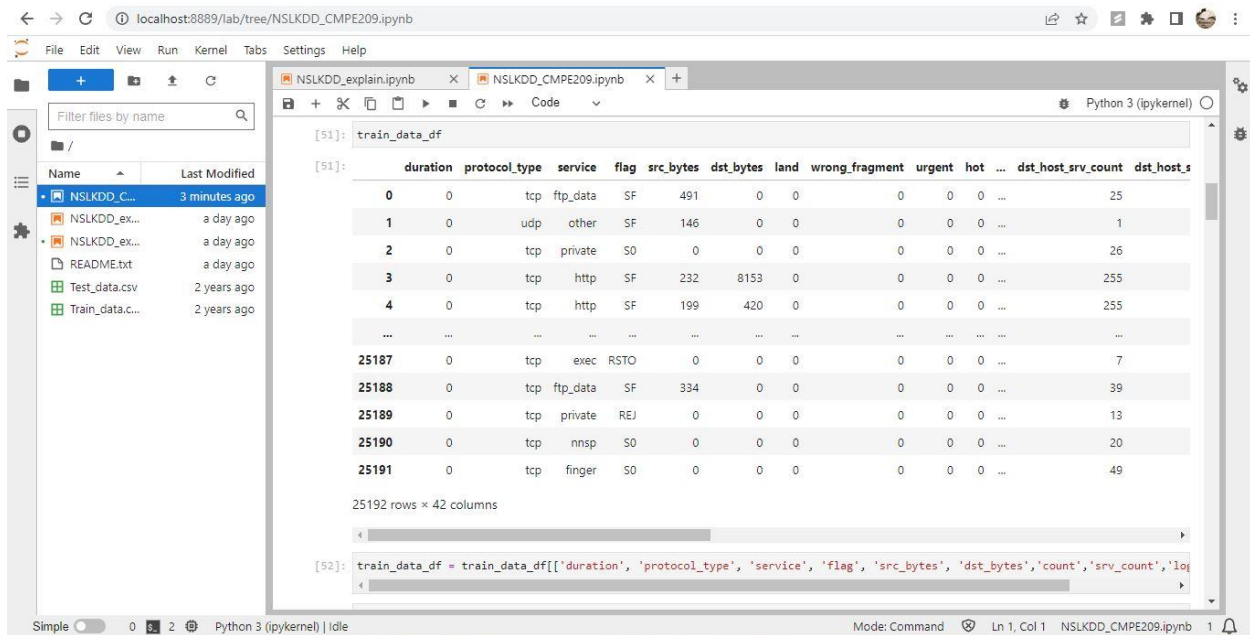
[51]: train_data_df

[51]:
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_s
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	
2	0	tcp	private	SO	0	0	0	0	0	0	...	26	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	
...	
25187	0	tcp	exec	RSTO	0	0	0	0	0	0	...	7	

Fig. 3: Improved code 2

4. Figure 4 shows the continued code.



The screenshot shows a Jupyter Notebook interface with a file browser on the left and a code editor on the right. The file browser shows a directory with files like NSLKDD_C..., NSLKDD_ex..., README.txt, Test_data.csv, and Train_data.csv. The code editor shows the following code:

```
[51]: train_data_df
```

```
[51]:
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_s
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	
...
25187	0	tcp	exec	RSTO	0	0	0	0	0	0	...	7	
25188	0	tcp	ftp_data	SF	334	0	0	0	0	0	...	39	
25189	0	tcp	private	REJ	0	0	0	0	0	0	...	13	
25190	0	tcp	nnsf	S0	0	0	0	0	0	0	...	20	
25191	0	tcp	finger	S0	0	0	0	0	0	0	...	49	

25192 rows x 42 columns

```
[52]: train_data_df = train_data_df[['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'count', 'srv_count', 'logged_in', 'class']]
```

Fig. 4: Improved code 3

5. Figure 5 shows the added columns.

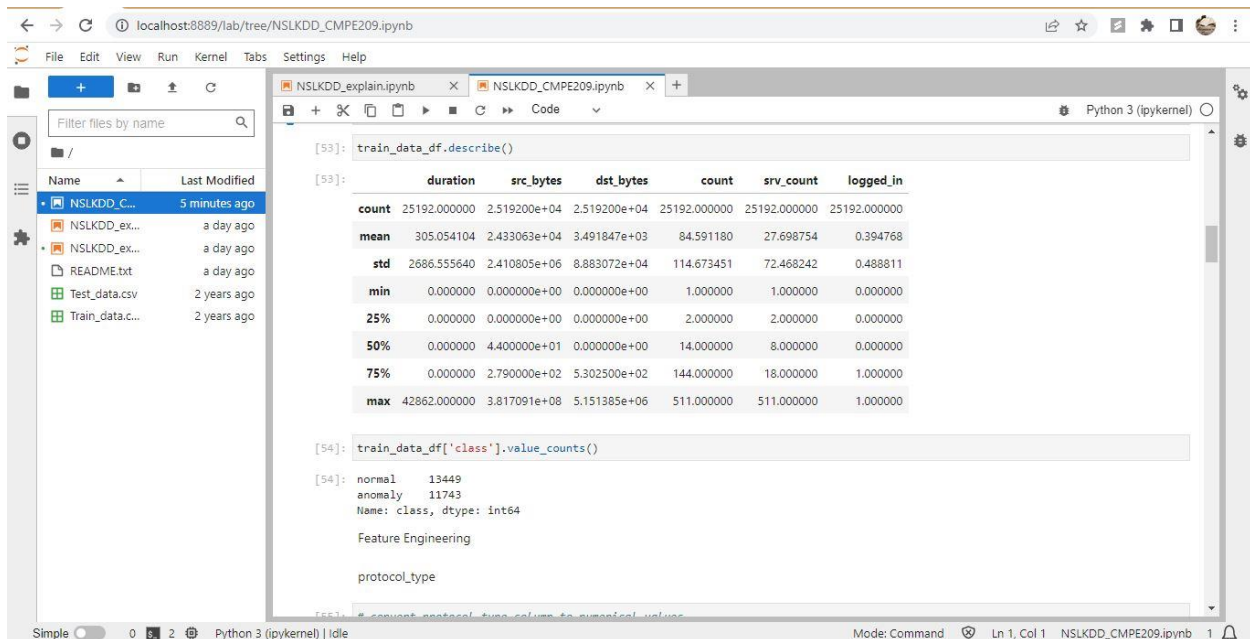


The screenshot shows a Jupyter Notebook interface with a file browser on the left and a code editor on the right. The file browser shows a directory with files like NSLKDD_C..., NSLKDD_ex..., README.txt, Test_data.csv, and Train_data.csv. The code editor shows the following code:

```
[52]: train_data_df[['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'count', 'srv_count', 'logged_in', 'class']]
```

Fig. 5: Added columns

6. Figure 6 shows the continued code.



The screenshot shows a Jupyter Notebook interface with a file browser on the left and a code editor on the right. The file browser shows a directory with files like NSLKDD_C..., NSLKDD_ex..., README.txt, Test_data.csv, and Train_data.csv. The code editor shows the following code:

```
[53]: train_data_df.describe()
```

```
[53]:
```

	duration	src_bytes	dst_bytes	count	srv_count	logged_in
count	25192.000000	2.519200e+04	2.519200e+04	25192.000000	25192.000000	25192.000000
mean	305.054104	2.433063e+04	3.491847e+03	84.591180	27.698754	0.394768
std	2686.555640	2.410805e+06	8.883072e+04	114.673451	72.468242	0.488811
min	0.000000	0.000000e+00	0.000000e+00	1.000000	1.000000	0.000000
25%	0.000000	0.000000e+00	0.000000e+00	2.000000	2.000000	0.000000
50%	0.000000	4.400000e+01	0.000000e+00	14.000000	8.000000	0.000000
75%	0.000000	2.790000e+02	5.302500e+02	144.000000	18.000000	1.000000
max	42862.000000	3.817091e+08	5.151385e+06	511.000000	511.000000	1.000000

```
[54]: train_data_df['class'].value_counts()
```

```
[54]:
```

class	count
normal	13449
anomaly	11743

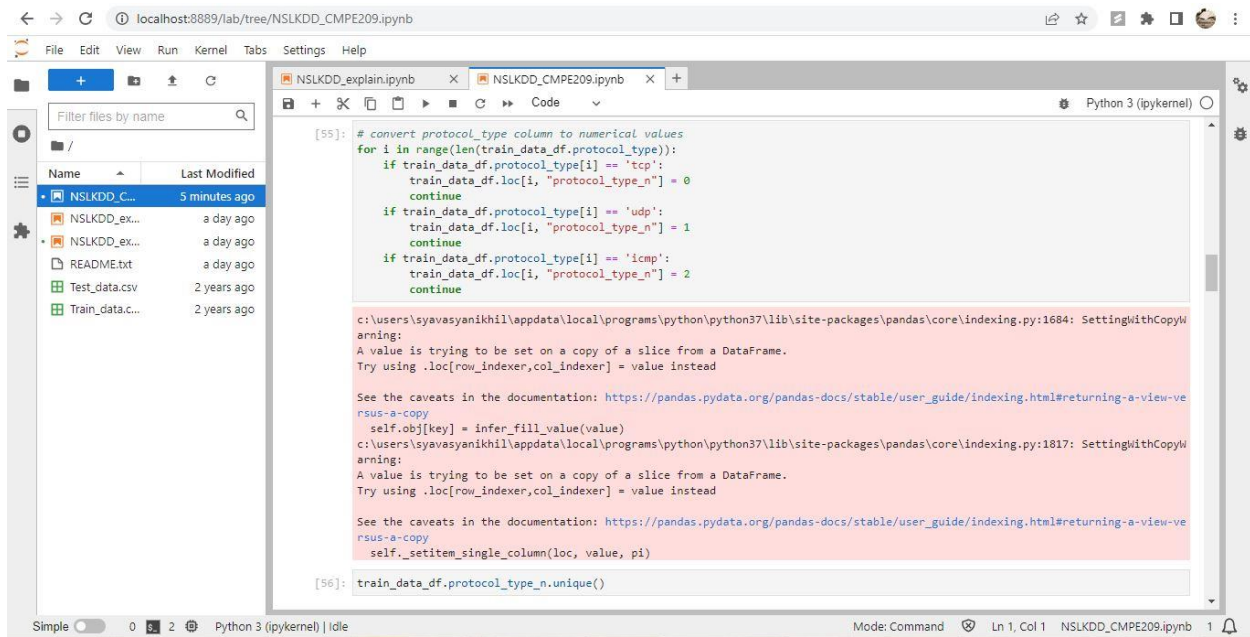
Name: class, dtype: int64

Feature Engineering

protocol_type

Fig. 6: Improved code 4

7. Figure 7 shows the continued code.



The screenshot shows a Jupyter Notebook with a file explorer on the left and a code editor on the right. The file explorer shows files like NSLKDD_C..., NSLKDD_ex..., README.txt, Test_data.csv, and Train_data.csv. The code editor shows the following code:

```
[55]: # convert protocol_type column to numerical values
for i in range(len(train_data_df.protocol_type)):
    if train_data_df.protocol_type[i] == 'tcp':
        train_data_df.loc[i, "protocol_type_n"] = 0
        continue
    if train_data_df.protocol_type[i] == 'udp':
        train_data_df.loc[i, "protocol_type_n"] = 1
        continue
    if train_data_df.protocol_type[i] == 'icmp':
        train_data_df.loc[i, "protocol_type_n"] = 2
        continue

c:\users\syavasyanikhil\appdata\local\programs\python\python37\lib\site-packages\pandas\core\indexing.py:1684: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

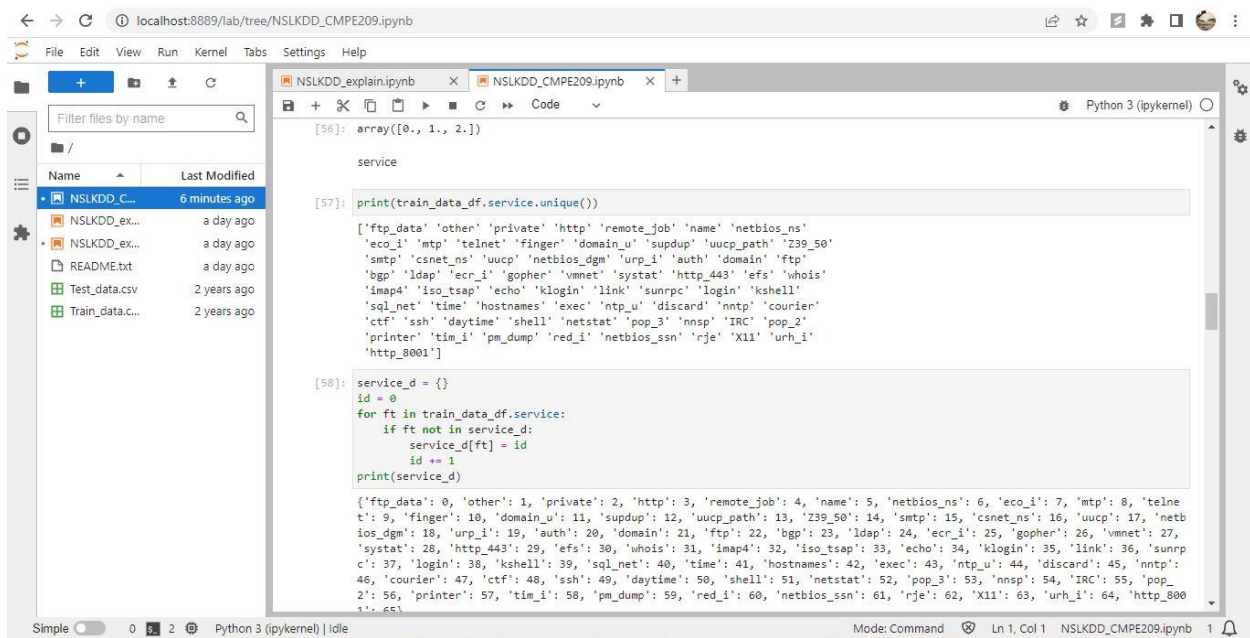
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
self.obj[key] = infer_fill_value(value)
c:\users\syavasyanikhil\appdata\local\programs\python\python37\lib\site-packages\pandas\core\indexing.py:1817: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
self.setitem_single_column(loc, value, pi)

[56]: train_data_df.protocol_type_n.unique()
```

Fig. 7: Improved code 5

8. Figure 8 shows the continued code.



The screenshot shows a Jupyter Notebook with a file explorer on the left and a code editor on the right. The file explorer shows files like NSLKDD_C..., NSLKDD_ex..., README.txt, Test_data.csv, and Train_data.csv. The code editor shows the following code:

```
[56]: array([0., 1., 2.])

service

[57]: print(train_data_df.service.unique())

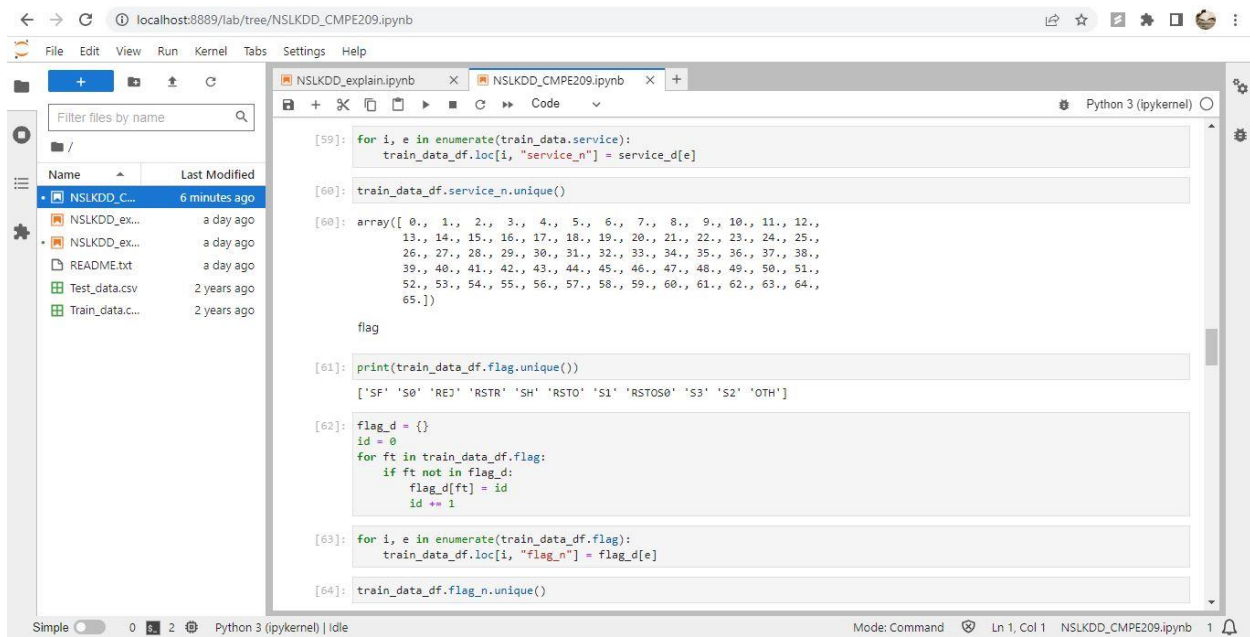
['ftp_data' 'other' 'private' 'http' 'remote_job' 'name' 'netbios_ns'
'eco_i' 'mtp' 'telnet' 'finger' 'domain_u' 'supdup' 'uucp_path' 'Z39_50'
'smtp' 'csnet_ns' 'uucp' 'netbios_dgm' 'urp_i' 'auth' 'domain' 'ftp'
'bgp' 'ldap' 'ecr_i' 'gopher' 'vmnet' 'sysstat' 'http_443' 'efs' 'whois'
'imap4' 'iso_tsap'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

[58]: service_d = {}
id = 0
for ft in train_data_df.service:
    if ft not in service_d:
        service_d[ft] = id
        id += 1
print(service_d)

{'ftp_data': 0, 'other': 1, 'private': 2, 'http': 3, 'remote_job': 4, 'name': 5, 'netbios_ns': 6, 'eco_i': 7, 'mtp': 8, 'telnet': 9, 'finger': 10, 'domain_u': 11, 'supdup': 12, 'uucp_path': 13, 'Z39_50': 14, 'smtp': 15, 'csnet_ns': 16, 'uucp': 17, 'netbios_dgm': 18, 'urp_i': 19, 'auth': 20, 'domain': 21, 'ftp': 22, 'bgp': 23, 'ldap': 24, 'ecr_i': 25, 'gopher': 26, 'vmnet': 27, 'sysstat': 28, 'http_443': 29, 'efs': 30, 'whois': 31, 'imap4': 32, 'iso_tsap': 33, 'echo': 34, 'klogin': 35, 'link': 36, 'sunrpc': 37, 'login': 38, 'kshell': 39, 'sql_net': 40, 'time': 41, 'hostnames': 42, 'exec': 43, 'ntp_u': 44, 'discard': 45, 'nntp': 46, 'courier': 47, 'ctf': 48, 'ssh': 49, 'daytime': 50, 'shell': 51, 'netstat': 52, 'pop_3': 53, 'nnsdp': 54, 'IRC': 55, 'pop_2': 56, 'printer': 57, 'tim_i': 58, 'pm_dump': 59, 'red_i': 60, 'netbios_ssn': 61, 'rje': 62, 'X11': 63, 'urh_i': 64, 'http_8001': 65}
```

Fig. 8: Improved code 6

9. Figure 9 shows the continued code.



```
[59]: for i, e in enumerate(train_data.service):
      train_data_df.loc[i, "service_n"] = service_d[e]

[60]: train_data_df.service_n.unique()

[60]: array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
        13., 14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25.,
        26., 27., 28., 29., 30., 31., 32., 33., 34., 35., 36., 37., 38.,
        39., 40., 41., 42., 43., 44., 45., 46., 47., 48., 49., 50., 51.,
        52., 53., 54., 55., 56., 57., 58., 59., 60., 61., 62., 63., 64.,
        65.])

      flag

[61]: print(train_data_df.flag.unique())

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

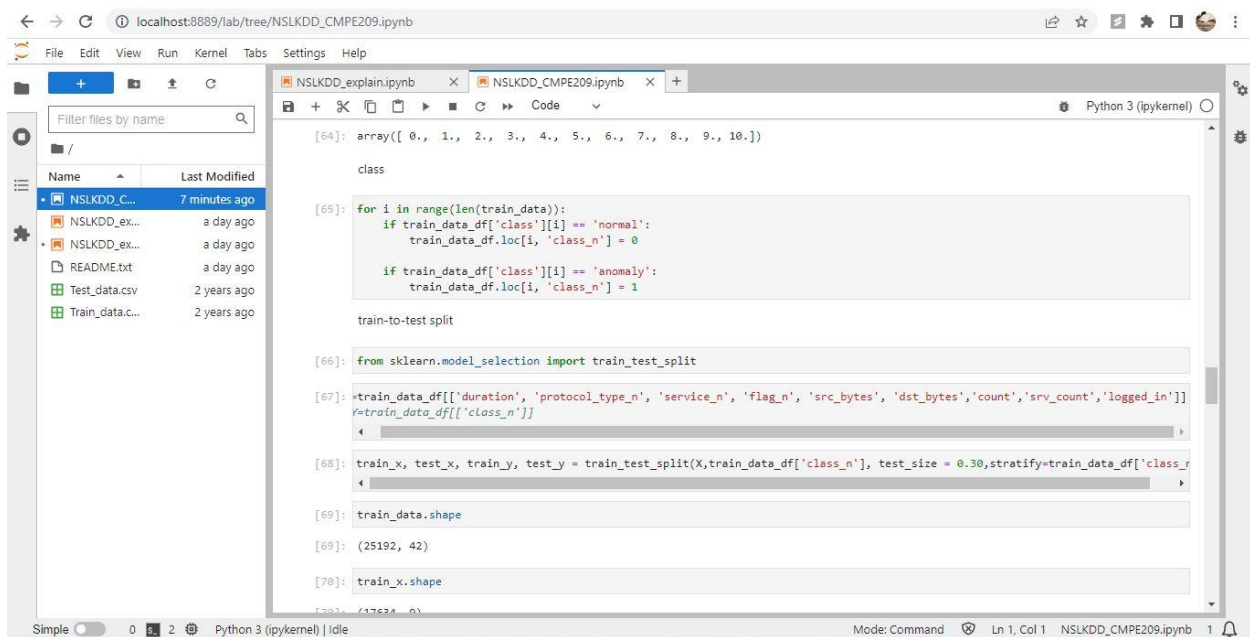
[62]: flag_d = {}
      id = 0
      for ft in train_data_df.flag:
          if ft not in flag_d:
              flag_d[ft] = id
              id += 1

[63]: for i, e in enumerate(train_data_df.flag):
      train_data_df.loc[i, "flag_n"] = flag_d[e]

[64]: train_data_df.flag_n.unique()
```

Fig. 9: Improved code 7

10. Figure 10 shows the continued code.



```
[64]: array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])

      class

[65]: for i in range(len(train_data)):
      if train_data_df['class'][i] == 'normal':
          train_data_df.loc[i, 'class_n'] = 0

      if train_data_df['class'][i] == 'anomaly':
          train_data_df.loc[i, 'class_n'] = 1

      train-to-test split

[66]: from sklearn.model_selection import train_test_split

[67]: train_data_df[['duration', 'protocol_type_n', 'service_n', 'flag_n', 'src_bytes', 'dst_bytes', 'count', 'srv_count', 'logged_in']]
      ~~~~~train_data_df[['class_n']]

[68]: train_x, test_x, train_y, test_y = train_test_split(X=train_data_df[['class_n']], test_size = 0.30, stratify=train_data_df[['class_n']])

[69]: train_data.shape

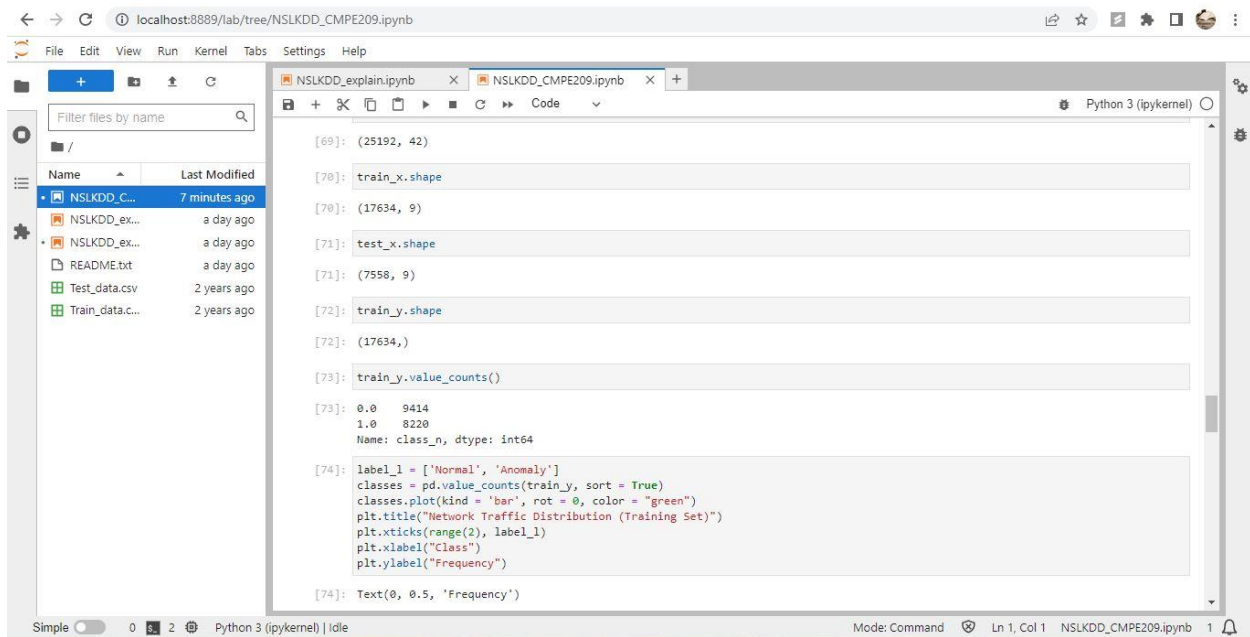
[69]: (25192, 42)

[70]: train_x.shape

[70]: (17634, 42)
```

Fig. 10: Improved code 8

11. Figure 11 shows the continued code.



The screenshot shows a Jupyter Notebook interface with a file browser on the left and a code editor on the right. The file browser lists files: NSLKDD_C..., NSLKDD_ex..., README.txt, Test_data.csv, and Train_data.csv. The code editor shows the following code:

```
[69]: (25192, 42)
[70]: train_x.shape
[70]: (17634, 9)
[71]: test_x.shape
[71]: (7558, 9)
[72]: train_y.shape
[72]: (17634,)
[73]: train_y.value_counts()
[73]: 0.0    9414
      1.0    8220
      Name: class_n, dtype: int64
[74]: label_1 = ['Normal', 'Anomaly']
      classes = pd.value_counts(train_y, sort = True)
      classes.plot(kind = 'bar', rot = 0, color = "green")
      plt.title("Network Traffic Distribution (Training Set)")
      plt.xticks(range(2), label_1)
      plt.xlabel("Class")
      plt.ylabel("Frequency")
[74]: Text(0, 0.5, 'Frequency')
```

Fig. 11: Improved code 9

12. Figure 12 shows the continued code.

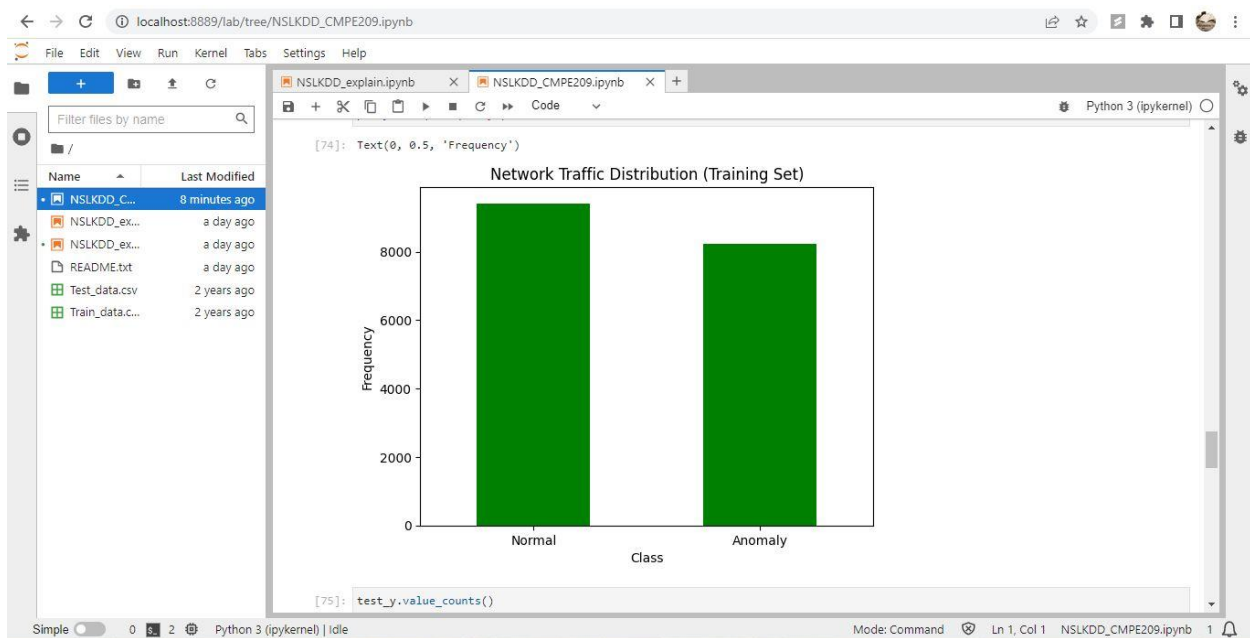


Fig. 12: Improved code 10

13. Figure 13 shows the continued code.

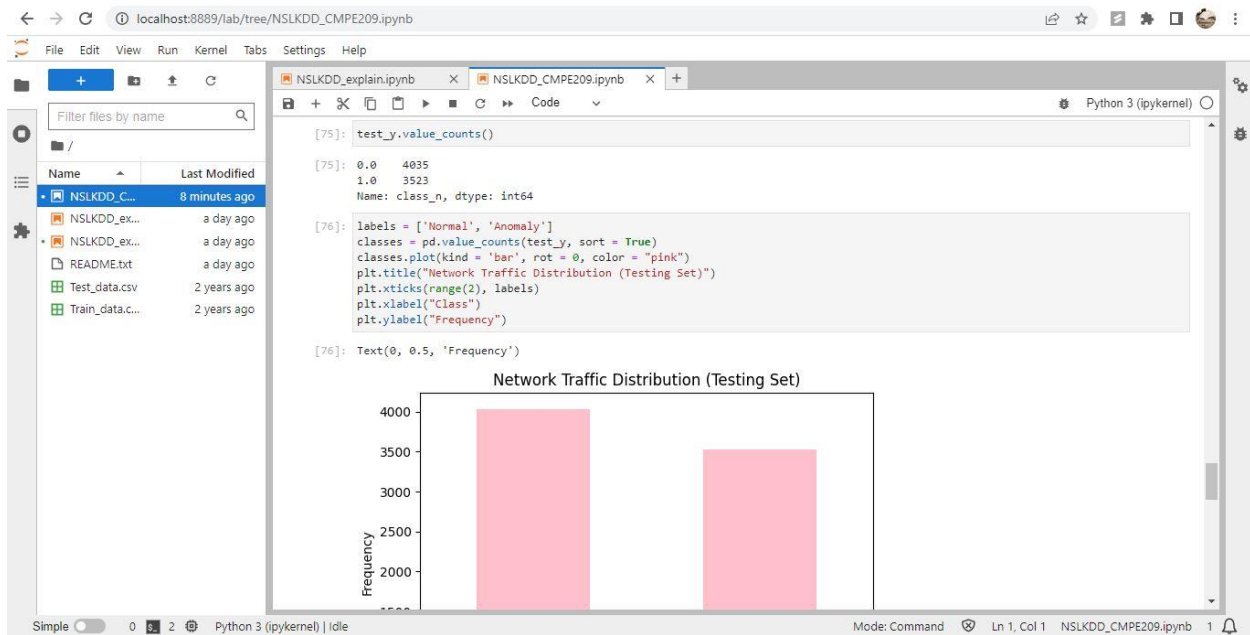


Fig. 13: Improved code 11

14. Figure 14 shows the continued code.

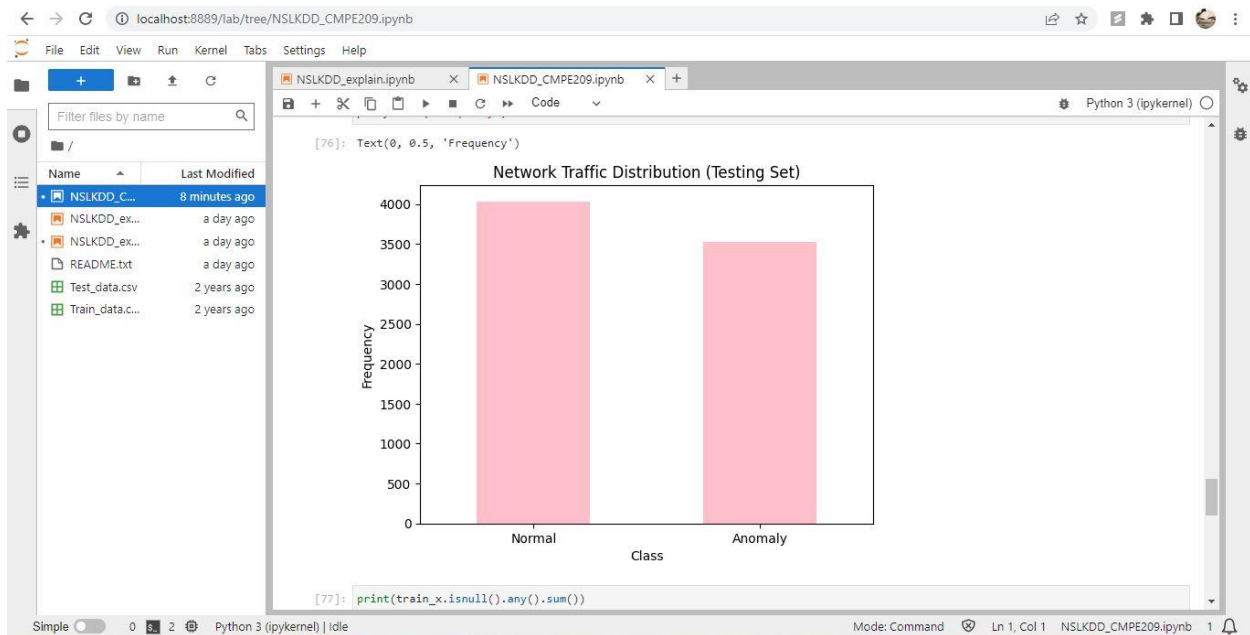
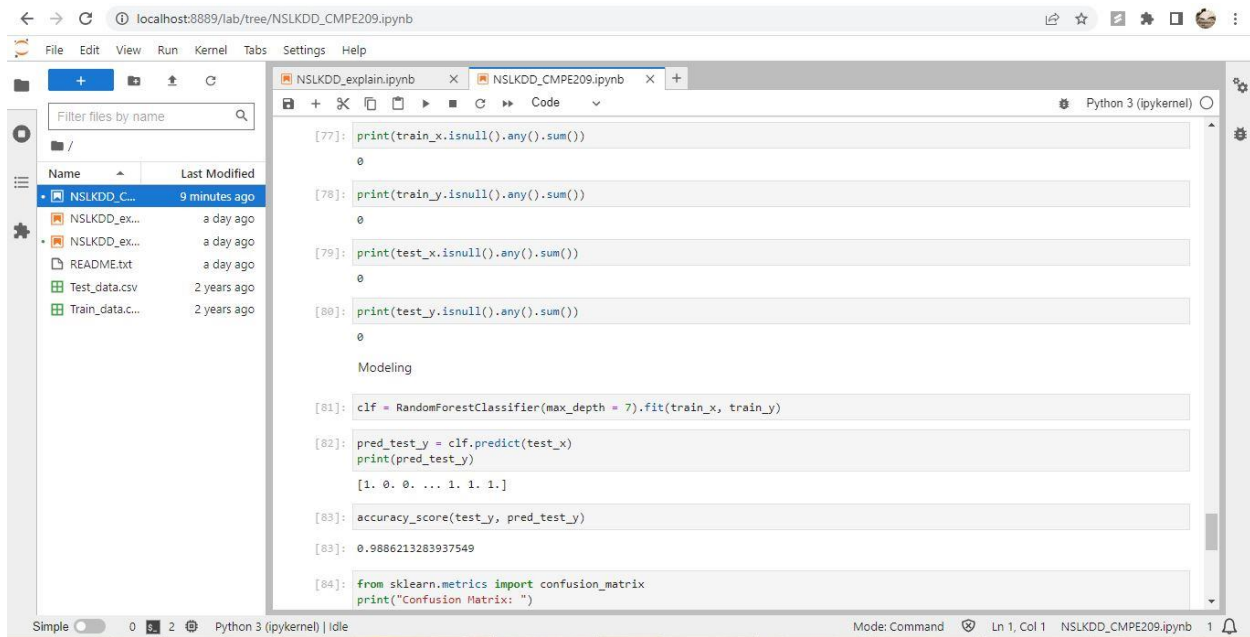


Fig. 14: Improved code 12

15. Figure 15 shows the achieved 99% output.



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory with files: NSLKDD_C..., NSLKDD_ex..., README.txt, Test_data.csv, and Train_data.c... The code editor contains the following Python code:

```
[77]: print(train_x.isnull().any().sum())
0

[78]: print(train_y.isnull().any().sum())
0

[79]: print(test_x.isnull().any().sum())
0

[80]: print(test_y.isnull().any().sum())
0

Modeling

[81]: clf = RandomForestClassifier(max_depth = 7).fit(train_x, train_y)

[82]: pred_test_y = clf.predict(test_x)
print(pred_test_y)
[1. 0. 0. ... 1. 1. 1.]

[83]: accuracy_score(test_y, pred_test_y)

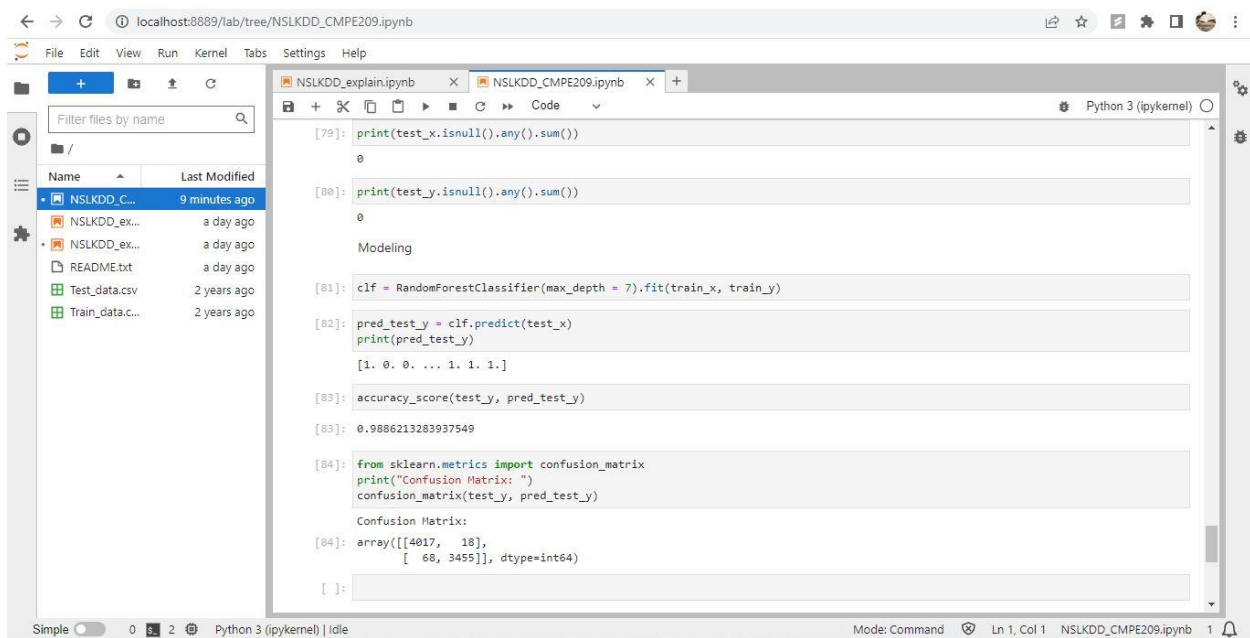
[83]: 0.9886213283937549

[84]: from sklearn.metrics import confusion_matrix
print("Confusion Matrix: ")
```

The output of the notebook shows the accuracy score of 0.9886213283937549, which is approximately 99%.

Fig. 15: Output

16. Figure 16 shows the confusion matrix.



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory with files: NSLKDD_C..., NSLKDD_ex..., README.txt, Test_data.csv, and Train_data.c... The code editor contains the following Python code:

```
[79]: print(test_x.isnull().any().sum())
0

[80]: print(test_y.isnull().any().sum())
0

Modeling

[81]: clf = RandomForestClassifier(max_depth = 7).fit(train_x, train_y)

[82]: pred_test_y = clf.predict(test_x)
print(pred_test_y)
[1. 0. 0. ... 1. 1. 1.]

[83]: accuracy_score(test_y, pred_test_y)

[83]: 0.9886213283937549

[84]: from sklearn.metrics import confusion_matrix
print("Confusion Matrix: ")
confusion_matrix(test_y, pred_test_y)

Confusion Matrix:
[84]: array([[4017, 18],
[ 68, 3455]], dtype=int64)

[ ]:
```

The output of the notebook shows the confusion matrix for the model. The confusion matrix is a 2x2 array with the following values:

	Actual 0	Actual 1
Predicted 0	4017	18
Predicted 1	68	3455

Fig. 16: Confusion matrix

6. Benefits of Machine Learning techniques for network intrusion detection:

Due to the advantages, they provide, machine learning techniques have gained popularity in the realm of network intrusion detection in recent years. The 5 advantages/benefits are:

1. **Anomaly detection:** Machine learning algorithms can identify anomalies in network traffic that may be indicative of an intrusion, even if the attack does not fit a known pattern. This is important because many attacks are not simple to detect and may involve complex techniques such as polymorphic malware or multi-stage attacks. Machine learning algorithms can learn to recognize anomalous behavior and identify potential threats even if the attack doesn't fit a known pattern.
2. **Real-time detection:** Machine learning algorithms can analyze network traffic in real-time, allowing for the rapid detection of network intrusions as they occur. This is particularly important in today's fast-paced and constantly changing threat landscape, where attackers are constantly developing new techniques to bypass traditional security measures.
3. **Improved accuracy:** Machine learning algorithms can analyze large amounts of network traffic and learn to identify patterns that are indicative of a network intrusion. Traditional intrusion detection systems (IDS) use pre-defined rules to detect attacks, but these rules may not be comprehensive enough to detect all attacks or may generate false positives. Machine learning algorithms can learn from data and improve their accuracy over time, resulting in more accurate detection of network intrusions.
4. **Adaptability:** Machine learning algorithms can adapt to changes in network traffic patterns and can be trained to detect new types of attacks as they emerge. This is crucial in a constantly evolving threat landscape where new attack techniques are constantly being developed. Machine learning algorithms can be trained on new data to recognize new patterns and identify emerging threats.
5. **Reduced human effort:** Machine learning algorithms can automate the process of intrusion detection, reducing the need for manual intervention and allowing security teams to focus on more complex tasks. This can help reduce the workload on security teams and allow them to be more efficient in identifying and responding to threats.

Overall, the use of machine learning techniques for network intrusion detection can lead to more accurate and efficient detection of network intrusions, helping to improve overall network security. Machine learning algorithms can provide real-time, scalable, and adaptable detection capabilities that can reduce the workload on security teams and help identify emerging threats.

7. Benefits of using NFV:

NFV, or Network Function Virtualization, is a technology that enables the virtualization of network functions such as firewalls, routers, load balancers, and other network appliances, and facilitates their operation on conventional hardware. Here are 5 benefits of using NFV:

1. **Enhanced Security:** Virtualized network functions can be easily updated and patched, improving security and reducing the risk of security breaches. Since virtualized network functions run on standard hardware, businesses can take advantage of industry-standard security features and protocols to protect their network infrastructure.

2. **Cost Savings:** NFV can help businesses reduce capital expenses associated with purchasing proprietary hardware-based network appliances. Since virtualized network functions can run on standard hardware, businesses can avoid purchasing expensive proprietary hardware. Additionally, operational expenses can also be reduced as virtualized network functions can be easily managed from a single interface, reducing the need for specialized personnel to manage multiple hardware-based appliances.
3. **Faster Time-to-Market:** By leveraging virtualized network functions, businesses can deploy network services more quickly than with traditional hardware-based appliances. This can help businesses be more responsive to market demands and introduce new services more rapidly.
4. **Scalability:** Virtualized network functions can be easily scaled up or down to meet changing demand. This is particularly useful for businesses that experience seasonal spikes in network traffic or that have unpredictable traffic patterns. Scaling can be done by adding or removing resources to meet demand, without the need for additional hardware.
5. **Flexibility:** Virtualized network functions can be easily moved between physical servers, making it easier to manage and maintain the network infrastructure. This means businesses can easily move virtualized network functions between physical locations, enabling them to optimize their network infrastructure based on their changing needs.

CONCLUSION:

From this Homework activity, I learned about Software Defined Networks and their advantages. I also learned about data sets and how to improve the accuracy of Random Forest by changing a few parameters. Further, I learned how Machine Learning is used for network intrusion detection.

APPENDIX

Code:

Name: NSLKDD_CMPE209.ipynb

```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import sklearn.preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
train_data = pd.read_csv('Train_data.csv')
test_data = pd.read_csv('Test_data.csv')
```

```
train_data_df = pd.DataFrame(train_data)
print(train_data_df.shape)
```

```
test_data_df = pd.DataFrame(test_data)
print(test_data_df.shape)
```

```
set(train_data_df).difference(set(test_data_df))
```

```
train_data_df.columns
```

```
train_data_df
```

```
train_data_df = train_data_df[['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
'dst_bytes', 'count', 'srv_count', 'logged_in', 'class']]
```

```
train_data_df.describe()
```

```
train_data_df['class'].value_counts()
```

Feature Engineering

protocol_type

```
# convert protocol_type column to numerical values
for i in range(len(train_data_df.protocol_type)):
    if train_data_df.protocol_type[i] == 'tcp':
        train_data_df.loc[i, "protocol_type_n"] = 0
        continue
    if train_data_df.protocol_type[i] == 'udp':
        train_data_df.loc[i, "protocol_type_n"] = 1
        continue
```

```
if train_data_df.protocol_type[i] == 'icmp':  
    train_data_df.loc[i, "protocol_type_n"] = 2  
    continue
```

```
train_data_df.protocol_type_n.unique()
```

service

```
print(train_data_df.service.unique())
```

```
service_d = { }  
id = 0  
for ft in train_data_df.service:  
    if ft not in service_d:  
        service_d[ft] = id  
        id += 1  
print(service_d)
```

```
for i, e in enumerate(train_data.service):  
    train_data_df.loc[i, "service_n"] = service_d[e]
```

```
train_data_df.service_n.unique()
```

flag

```
print(train_data_df.flag.unique())
```

```
flag_d = { }  
id = 0  
for ft in train_data_df.flag:  
    if ft not in flag_d:  
        flag_d[ft] = id  
        id += 1
```

```
for i, e in enumerate(train_data_df.flag):  
    train_data_df.loc[i, "flag_n"] = flag_d[e]
```

```
train_data_df.flag_n.unique()
```

class

```
for i in range(len(train_data)):   
    if train_data_df['class'][i] == 'normal':  
        train_data_df.loc[i, 'class_n'] = 0  
  
    if train_data_df['class'][i] == 'anomaly':  
        train_data_df.loc[i, 'class_n'] = 1
```

train-to-test split

```
from sklearn.model_selection import train_test_split
```

```
X=train_data_df[['duration', 'protocol_type_n', 'service_n', 'flag_n', 'src_bytes',  
'dst_bytes','count','srv_count','logged_in']]  
#Y=train_data_df[['class_n']]
```

```
train_x, test_x, train_y, test_y = train_test_split(X,train_data_df['class_n'], test_size =  
0.30,stratify=train_data_df['class_n'])
```

```
train_data.shape
```

```
train_x.shape
```

```
test_x.shape
```

```
train_y.shape
```

```
train_y.value_counts()
```

```
label_l = ['Normal', 'Anomaly']  
classes = pd.value_counts(train_y, sort = True)  
classes.plot(kind = 'bar', rot = 0, color = "green")  
plt.title("Network Traffic Distribution (Training Set)")  
plt.xticks(range(2), label_l)  
plt.xlabel("Class")  
plt.ylabel("Frequency")
```

```
test_y.value_counts()
```

```
labels = ['Normal', 'Anomaly']  
classes = pd.value_counts(test_y, sort = True)  
classes.plot(kind = 'bar', rot = 0, color = "pink")  
plt.title("Network Traffic Distribution (Testing Set)")  
plt.xticks(range(2), labels)  
plt.xlabel("Class")  
plt.ylabel("Frequency")
```

```
print(train_x.isnull().any().sum())
```

```
print(train_y.isnull().any().sum())
```

```
print(test_x.isnull().any().sum())
```

```
print(test_y.isnull().any().sum())
```

Modeling

```
clf = RandomForestClassifier(max_depth = 7).fit(train_x, train_y)
```

```
pred_test_y = clf.predict(test_x)  
print(pred_test_y)
```

```
accuracy_score(test_y, pred_test_y)
```

```
from sklearn.metrics import confusion_matrix  
print("Confusion Matrix: ")  
confusion_matrix(test_y, pred_test_y)
```