KNN Model on
RT-IOT 2022 Dataset

Supervised By

Dr. Eng. Muhammad Atta Khfagy

Team Members

Mohamed AbdElrahman Elkhlawy

Amina Younis Mohamed

Manar Hussain Mohamed

Norhan Nageh Elabd

Omar Mohamed Elhassan

Sara Nabil Kamel

INTRODUCTION

The RT-IoT2022 dataset is a proprietary dataset derived from a real-time IoT infrastructure. It serves as a comprehensive resource integrating a diverse range of IoT devices and sophisticated network attack methodologies.

This dataset encompasses both normal and adversarial network behaviors, providing a general representation of real-world scenarios. Incorporating data from IoT devices such as ThingSpeak-LED, Wipro-Bulb, and MQTT-Temp, as well as simulated attack scenarios involving Brute-Force SSH attacks, DDoS attacks using Hping and Slowloris, and Nmap patterns, RT-IoT2022 offers a detailed perspective on the complex nature of network traffic. The bidirectional attributes of network traffic are meticulously captured using the Zeek network monitoring tool and the Flowmeter plugin.

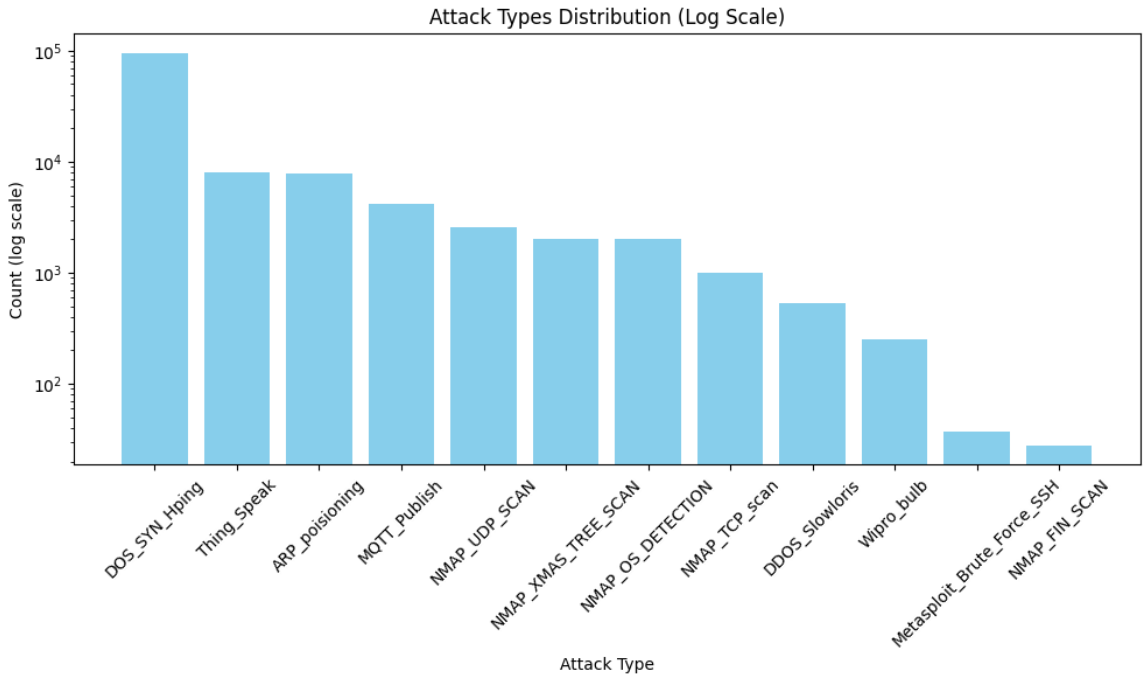
Researchers can leverage the RT-IoT2022 dataset to advance the capabilities of Intrusion Detection Systems (IDS), fostering the development of robust and adaptive security solutions for real-time IoT networks.

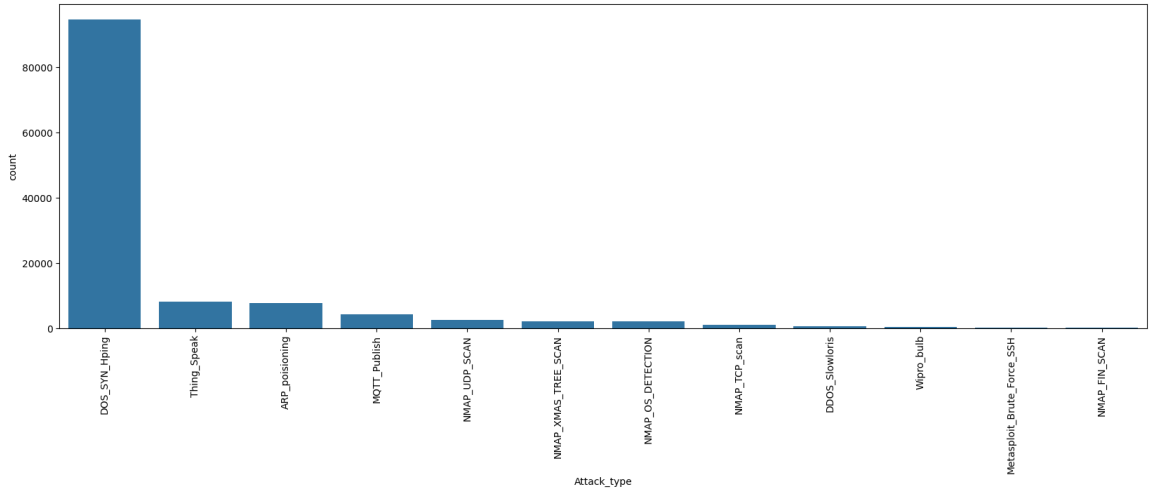
STEP 1: DATA EXPLORATION

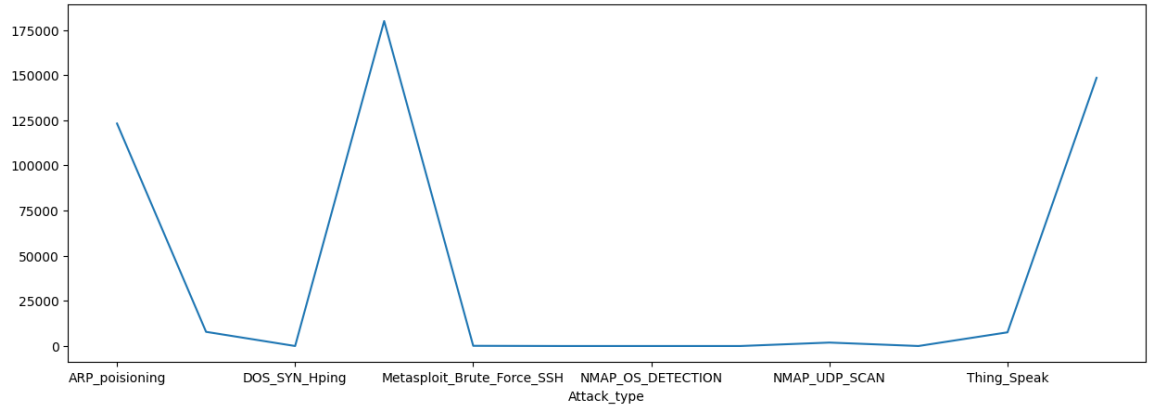
EDA Definition

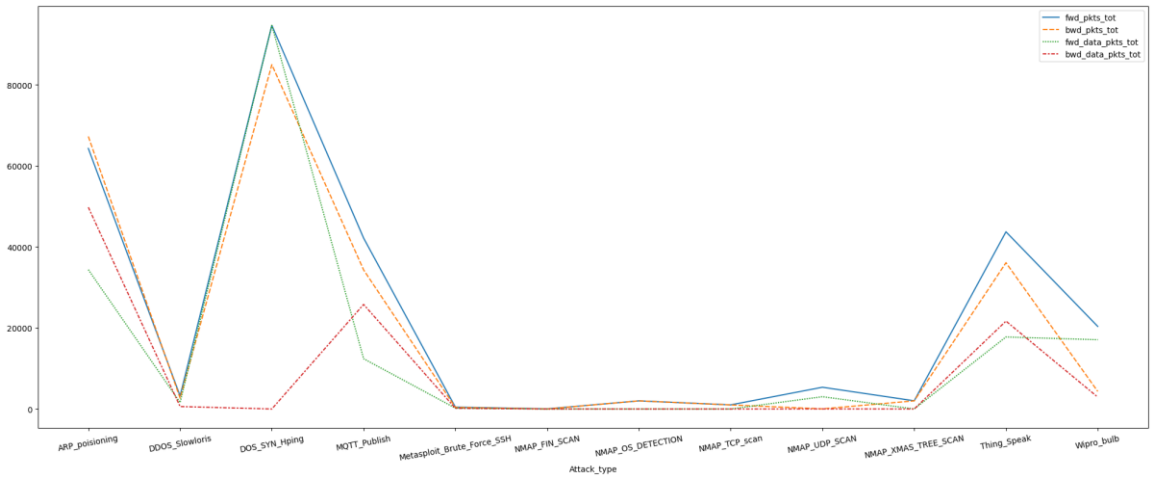
Exploratory Data Analysis (EDA) is a vital first step in building machine learning models. It involves using statistical methods and visualizations to understand the characteristics of your data. Through EDA, you can uncover patterns, identify relationships between features, and detect potential issues like missing values or outliers. These insights help prepare your data for modeling and can even inform the choice of machine learning algorithms best suited for the task.

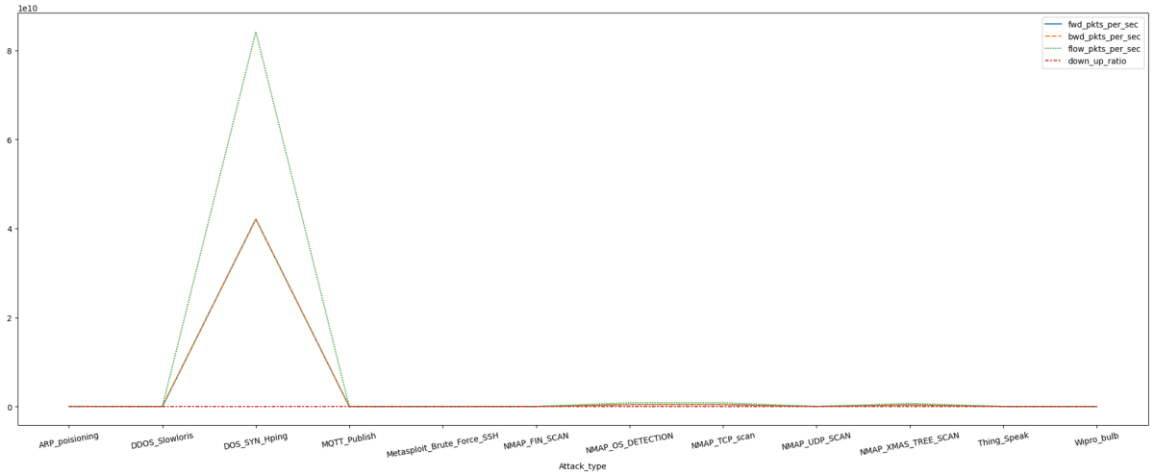
EDA for Model

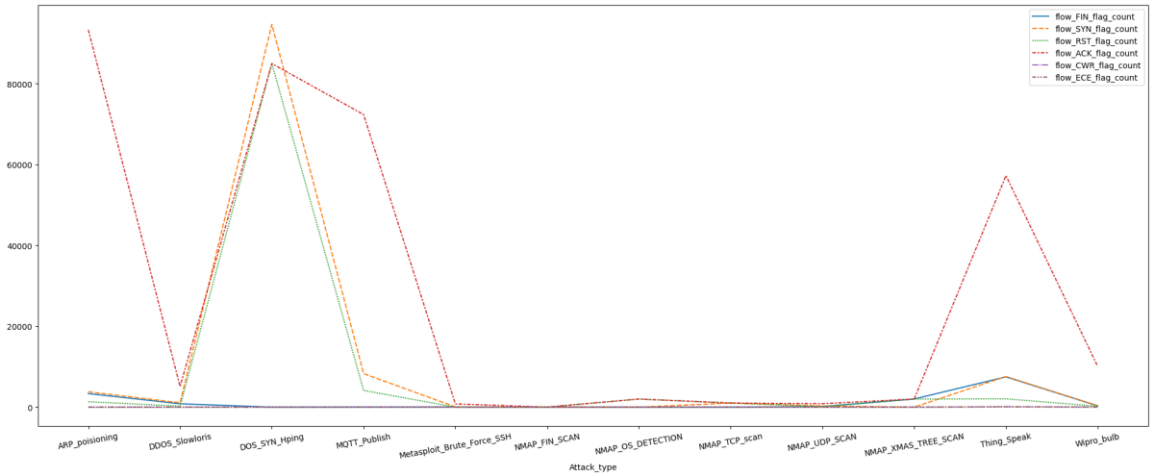


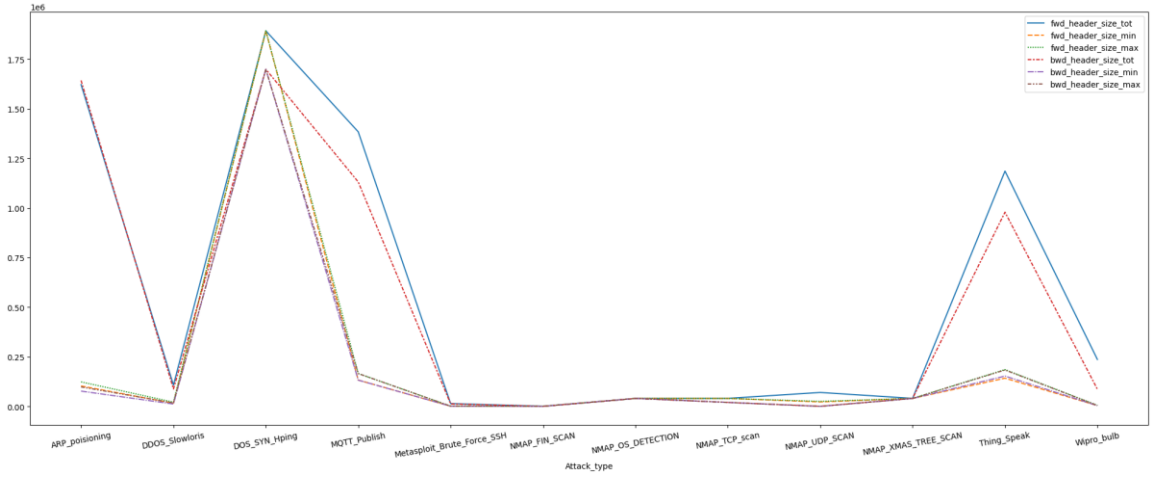

count
80000
60000
40000
20000
0
DOS_SYN_Hping
Thing_Speak
ARP_poisioning
MQTT_Publish
NMAP_UDP_SCAN
NMAP_XMAS_TREE_SCAN
NMAP_OS_DETECTION
NMAP_TCP_scan
DDOS_Slowloris
Wipro_bulb
Metasploit_Brute_Force_SSH
NMAP_FIN_SCAN
Attack_type

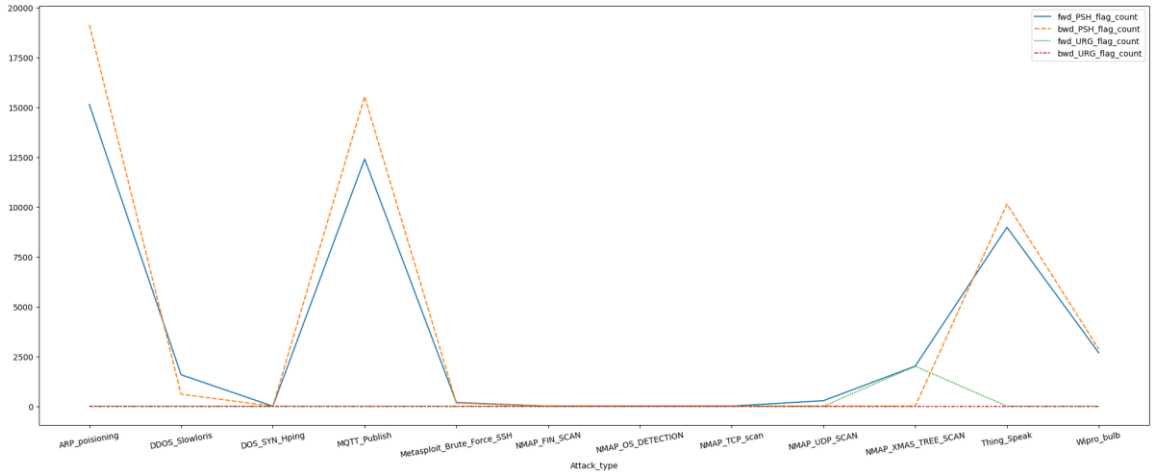

175000
150000
125000
100000
75000
50000
25000
0
ARP_poisioning
DOS_SYN_Hping
Metasploit_Brute_Force_SSH
NMAP_OS_DETECTION
NMAP_UDP_SCAN
Thing_Speak
Attack_type

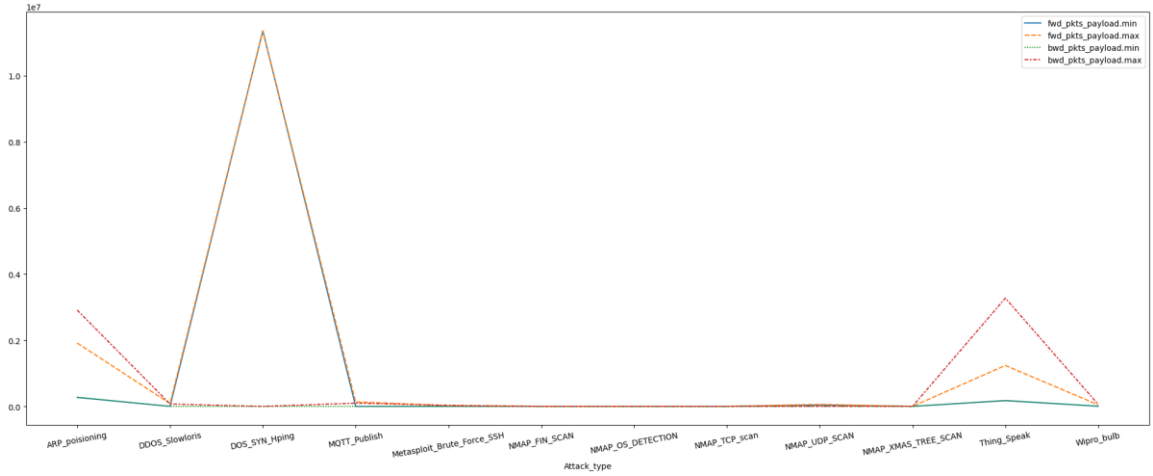

fwd_pkts_tot
bwd_pkts_tot
fwd_data_pkts_tot
bwd_data_pkts_tot
80000
60000
40000
20000
0
ARP_poisioning
DDOS_Slowloris
DOS_SYN_Hping
MQTT_Publish
Metasploit_Brute_Force_SSH
NMAP_FIN_SCAN
NMAP_OS_DETECTION
NMAP_TCP_scan
NMAP_UDP_SCAN
NMAP_XMAS_TREE_SCAN
Thing_Speak
Wipro_bulb
Attack_type

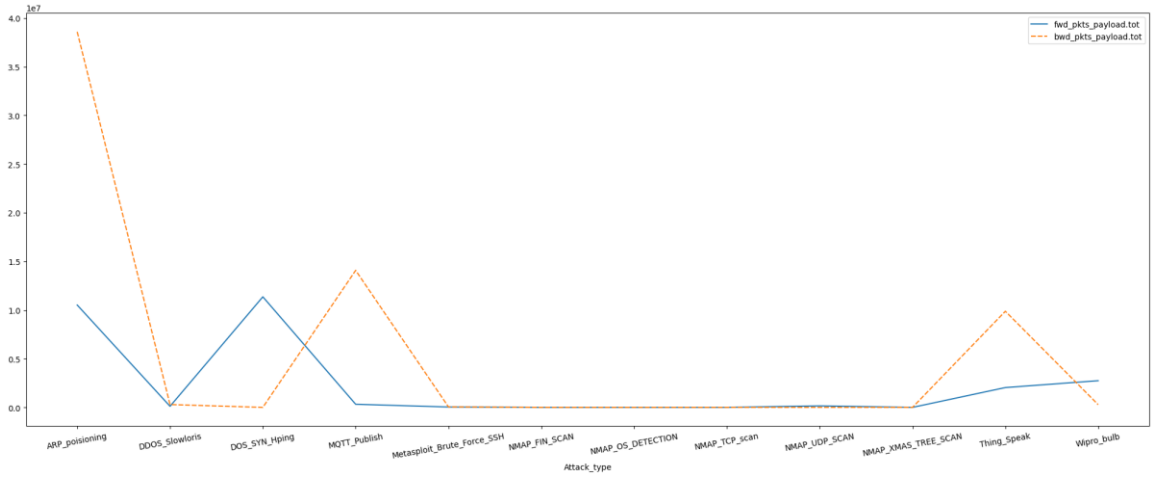

1e10
fwd_pkts_per_sec
bwd_pkts_per_sec
flow_pkts_per_sec
down_up_ratio
ARP_poisioning
DDOS_Slowloris
DOS_SYN_Hping
MQTT_Publish
Metasploit_Brute_Force_SSH
NMAP_FIN_SCAN
NMAP_OS_DETECTION
NMAP_TCP_scan
NMAP_UDP_SCAN
NMAP_XMAS_TREE_SCAN
Thing_Speak
Wipro_bulb
Attack_type

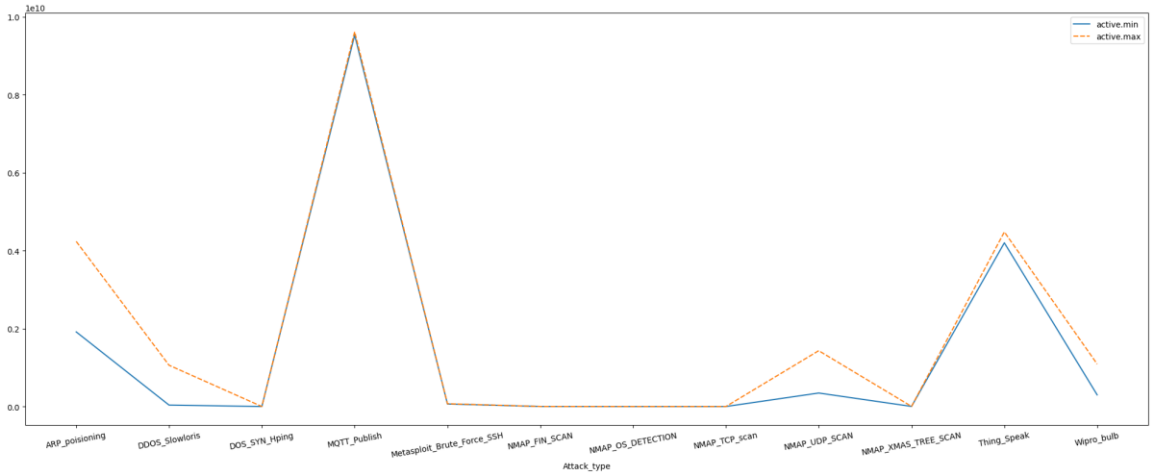

flow_FIN_flag_count
flow_SYN_flag_count
flow_RST_flag_count
flow_ACK_flag_count
flow_CWR_flag_count
flow_ECE_flag_count
ARP_poisioning
DDOS_Slowloris
DOS_SYN_Hping
MQTT_Publish
Metasploit_Brute_Force_SSH
NMAP_FIN_SCAN
NMAP_OS_DETECTION
NMAP_TCP_scan
NMAP_UDP_SCAN
NMAP_XMAS_TREE_SCAN
Thing_Speak
Wipro_bulb
Attack_type


1e6
fwd_header_size_tot
fwd_header_size_min
fwd_header_size_max
bwd_header_size_tot
bwd_header_size_min
bwd_header_size_max
ARP_poisioning
DDOS_Slowloris
DOS_SYN_Hping
MQTT_Publish
Metasploit_Brute_Force_SSH
NMAP_FIN_SCAN
NMAP_OS_DETECTION
NMAP_TCP_scan
NMAP_UDP_SCAN
NMAP_XMAS_TREE_SCAN
Thing_Speak
Wipro_bulb
Attack_type

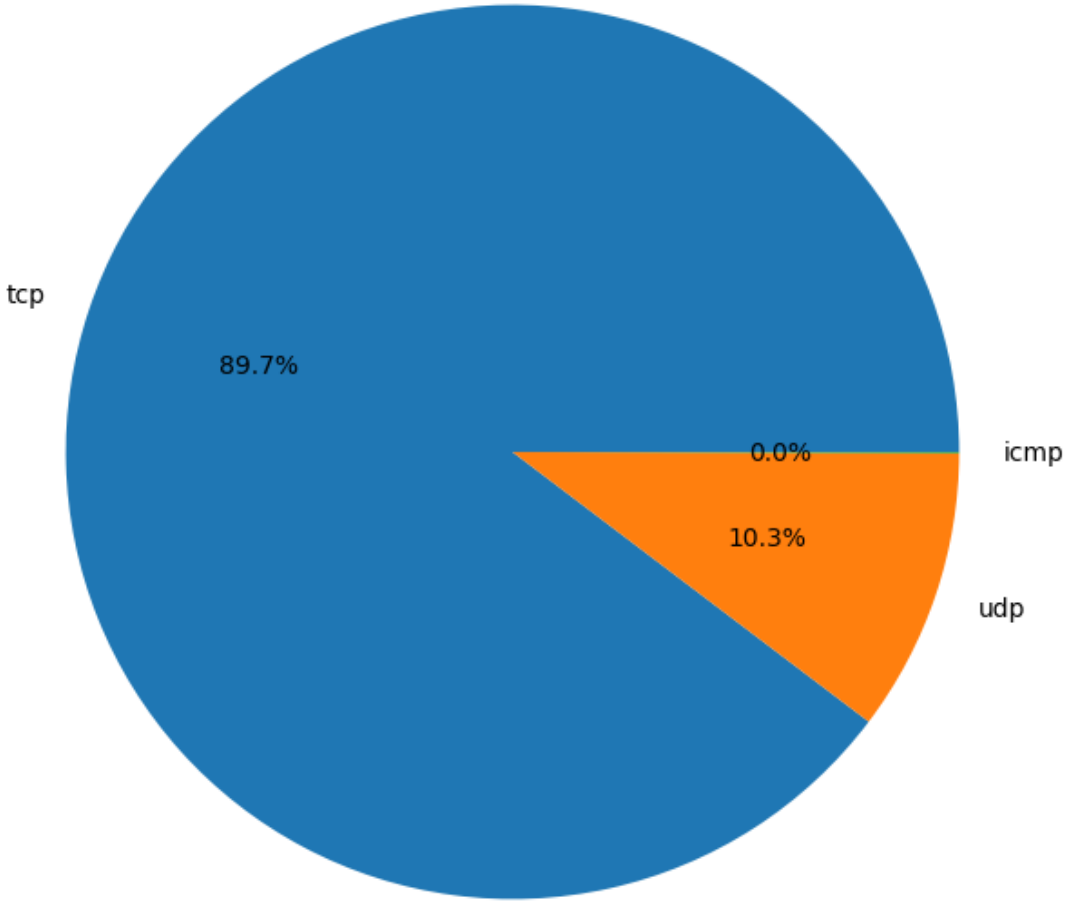

fwd_PSH_flag_count
bwd_PSH_flag_count
fwd_URG_flag_count
bwd_URG_flag_count
20000
17500
15000
12500
10000
7500
5000
2500
0
ARP_poisioning
DDOS_Slowloris
DOS_SYN_Hping
MQTT_Publish
Metasploit_Brute_Force_SSH
NMAP_FIN_SCAN
NMAP_OS_DETECTION
NMAP_TCP_scan
NMAP_UDP_SCAN
NMAP_XMAS_TREE_SCAN
Thing_Speak
Wipro_bulb
Attack_type

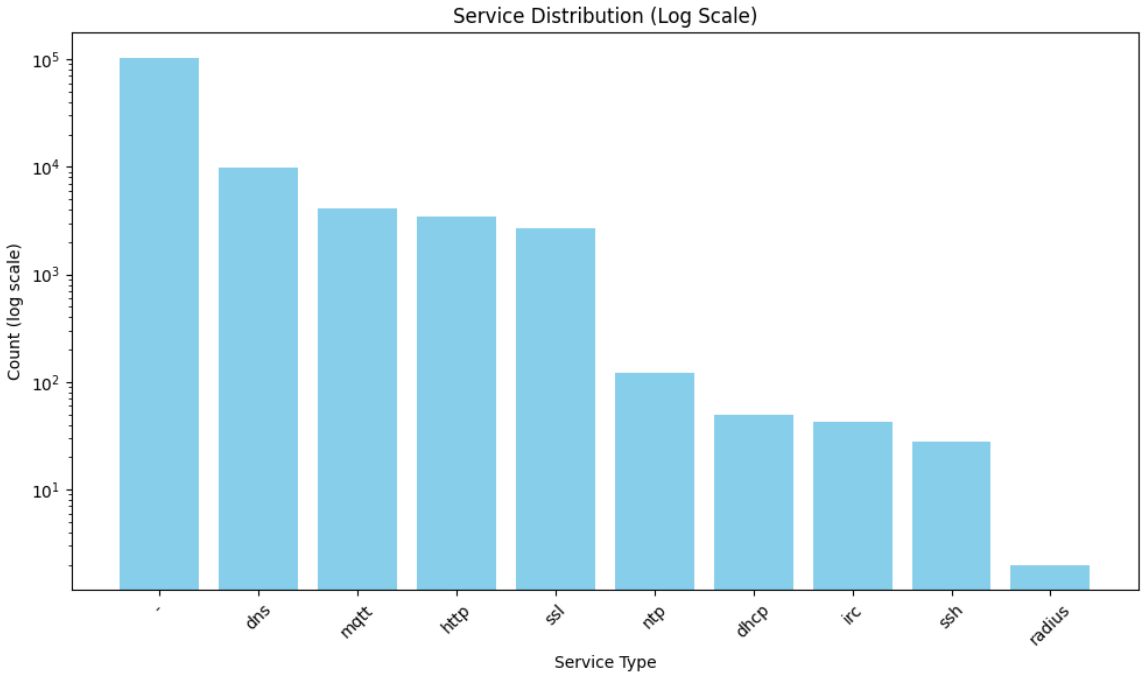

1e7
fwd_pkts_payload.min
fwd_pkts_payload.max
bwd_pkts_payload.min
bwd_pkts_payload.max
1.0
0.8
0.6
0.4
0.2
0.0
ARP_poisioning
DDOS_Slowloris
DOS_SYN_Hping
MQTT_Publish
Metasploit_Brute_Force_SSH
NMAP_FIN_SCAN
NMAP_OS_DETECTION
NMAP_TCP_scan
NMAP_UDP_SCAN
NMAP_XMAS_TREE_SCAN
Thing_Speak
Wipro_bulb
Attack_type


1e7
fwd_pkts_payload.tot
bwd_pkts_payload.tot
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
ARP_poisioning
DDOS_Slowloris
DOS_SYN_Hping
MQTT_Publish
Metasploit_Brute_Force_SSH
NMAP_FIN_SCAN
NMAP_OS_DETECTION
NMAP_TCP_scan
NMAP_UDP_SCAN
NMAP_XMAS_TREE_SCAN
Thing_Speak
Wipro_bulb
Attack_type



Protocols Distribution




Service Distribution (Log Scale)
Count (log scale)
10^5
10^4
10^3
10^2
10^1
dns
mqtt
http
ssl
ntp
dhcp
irc
ssh
radius
Service Type

STEP 2: MODEL SELECTION AND TRAINING

In this step we apply the following in the model:

1. Label Encoding
2. Data Scaling
3. Split Dataset

Figures

Data Scaling

```python
from sklearn.preprocessing import MinMaxScaler

columns_to_scale = dataset.columns[:-1]

scaler = MinMaxScaler()

dataset[columns_to_scale] = scaler.fit_transform(dataset[columns_to_scale])

dataset.head()
```

Python

no	id.orig_p	id.resp_p	proto	service	flow_duration	fwd_pkts_tot	bwd_pkts_tot	fwd_data_pkts_tot	bwd_da
0	0.590021	0.028797	0.5	0.555556	0.001473	0.002071	0.000494	0.00069	
1	0.780392	0.028797	0.5	0.555556	0.001467	0.002071	0.000494	0.00069	
2	0.683009	0.028797	0.5	0.555556	0.001478	0.002071	0.000494	0.00069	
3	0.929168	0.028797	0.5	0.555556	0.001471	0.002071	0.000494	0.00069	
4	0.779538	0.028797	0.5	0.555556	0.001468	0.002071	0.000494	0.00069	

5 rows × 84 columns

Label Encoding

```python
#before Encode
proto = set(dataset["proto"])
service = set(dataset["service"])
Attack_type = set(dataset["Attack_type"])

print(f"proto column before encode = {proto}")
print(f"service column before encode = {service}")
print(f"attack type column before encode = {Attack_type}")
```

[21] Python

```
proto column before encode = {'icmp', 'udp', 'tcp'}
service column before encode = {'-', 'ssl', 'mqtt', 'http', 'ssh', 'radius', 'dns', 'irc', 'ntp', 'dhc
attack type column before encode = {'Thing_Speak', 'MQTT_Publish', 'NMAP_OS_DETECTION', 'NMAP_UDP_SCAN
```

```python
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
dataset["proto"] = encoder.fit_transform(dataset["proto"])
dataset["service"] = encoder.fit_transform(dataset["service"])
dataset["Attack_type"] = encoder.fit_transform(dataset["Attack_type"])

dataset.head()
```

Python

no	id.orig_p	id.resp_p	proto	service	flow_duration	fwd_pkts_tot	bwd_pkts_tot	fwd_data_pkts_tot	bwd_dat
0	38667	1883	1	5	32.011598	9	5	3	
1	51143	1883	1	5	31.883584	9	5	3	
2	44761	1883	1	5	32.124053	9	5	3	
3	60893	1883	1	5	31.961063	9	5	3	

Split Dataset

+ Code + Markdown

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Assuming 'scaled_df' is your scaled dataset and the last column is the target variable
X = dataset.iloc[:, :-1]  # Features (all columns except the last one)
y = dataset.iloc[:, -1]   # Target variable (last column)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Create a KNN classifier instance
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors as needed

# Train the classifier on the training data
knn.fit(X_train, y_train)

# Predict on the test data
y_pred = knn.predict(X_test)
print(y_pred)

# Evaluate the performance of the classifier (e.g., using accuracy)
accuracy = knn.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9951267056530214
```

STEP 3: MODEL EVALUATION

In this step we calculate the model accuracy manually as following

Figures

```python
# Calculate accuracy
correct_predictions = sum(pred == true_label for pred, true_label in zip(y_pred, y_test))
total_predictions = len(y_test)
accuracy = correct_predictions / total_predictions
print("Accuracy (Manual):", accuracy)

# Calculate precision, recall, and F1-score
true_positives = sum(pred == 1 and true_label == 1 for pred, true_label in zip(y_pred, y_test))
false_positives = sum(pred == 1 and true_label == 0 for pred, true_label in zip(y_pred, y_test))
false_negatives = sum(pred == 0 and true_label == 1 for pred, true_label in zip(y_pred, y_test))

precision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)
f1_score = 2 * (precision * recall) / (precision + recall)

print("Precision (Manual):", precision)
print("Recall (Manual):", recall)
print("F1-score (Manual):", f1_score)

# Generate confusion matrix
unique_labels = sorted(set(y_test))
conf_matrix = [[sum((pred == label_pred) and (true_label == label_true) for pred, true_label in zip(y_pred, y_test)) for label_pred in
unique_labels] for label_true in unique_labels]

print("Confusion Matrix (Manual):")
for row in conf_matrix:
    print(row)
```

```
Accuracy (Manual): 0.9951267056530214
Precision (Manual): 0.9795918367346939
Recall (Manual): 0.9795918367346939
F1-score (Manual): 0.9795918367346939
Confusion Matrix (Manual):
[1539, 2, 0, 0, 2, 0, 0, 0, 0, 0, 35, 0]
[2, 96, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
[0, 0, 18897, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 870, 0, 0, 0, 0, 0, 0, 0, 0]
[2, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 393, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 220, 0, 0, 0, 0]
[7, 0, 0, 0, 0, 0, 0, 0, 476, 0, 6, 0]
[3, 0, 0, 0, 0, 0, 0, 0, 0, 381, 0, 0]
[43, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1581, 0]
[4, 0, 0, 0, 0, 1, 2, 0, 0, 0, 6, 45]
```

STEP 4: MODEL OPTIMIZATION

In this step we optimize the model to reach to the best number of K Neighbors as following

Figures

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_neighbors': [3, 5, 7, 9],  # List of K values to try
    # Add other parameters of KNN you want to tune here
}

# Create a KNN classifier instance
knn = KNeighborsClassifier()

# Create a GridSearchCV instance
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5)

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

# Get the best K value
best_k = best_params['n_neighbors']

# Get the best estimator (model)
best_model = grid_search.best_estimator_

# Report the results
print("Best K value:", best_k)
print("Best parameters:", best_params)
print("Best model:", best_model)
```

```
Best K value: 3
Best parameters: {'n_neighbors': 3}
Best model: KNeighborsClassifier(n_neighbors=3)
```

STEP 5: MODEL INFORMATION

Google Colab URL

- https://colab.research.google.com/drive/1J5R6SbhNmgJsDjEhwZdhMiLhLmAjPt_S

Thanks For Reading