TCP Packet-Network Traffic Analyzer with Anomaly Detection

By Reginald Saintil

Project Overview

This project focuses on analyzing **TCP network traffic** captured using **Wireshark** and saved as a .pcap file. By extracting key features such as **source IP**, **destination IP**, **packet length**, and **timestamp.** I've applied advanced **data analysis** and **machine learning** techniques to:

1. **Identify Anomalous Packets**: Using unsupervised learning (**KMeans clustering**), the system detects unusual network patterns that deviate from normal behavior.
2. **Classify Network Traffic**: A supervised machine learning model (**Random Forest Classifier**) is trained to distinguish between **normal** and **malicious** packets.
3. **Visualize Insights**: The results, including anomalies and cluster distributions, are visualized to provide clear insights into network traffic.

Objectives

The key goals of this project are:

- **Automated Packet Analysis**: Load and process TCP packets from a .pcap file.
- **Feature Extraction**: Extract meaningful features such as packet length, source, and destination.
- **Anomaly Detection**: Detect outliers in network traffic using clustering algorithms (KMeans).
- **Traffic Classification**: Build a supervised machine learning model to classify packets as normal or anomalous.
- **Visualization and Reporting**: Present findings through visualizations for easy interpretation

Project Workflow

- Install and import required libraries
- Capture TCP Packets
- Load and Process Packets
- Preprocess Data
- KMeans for Anomaly Detection
- Train Random Forest
- Visualize Insights
- Prediction

Install and import required libraries

- 1. Scapy - pip install scapy
- \- For reading, parsing, and analyzing packet data (ex: from .pcap files). It allows us to extract key features like source IP, destination IP, protocol, packet length, and timestamp.
- 2. Pandas - pip install pandas
- Organizes extracted packet data into a structured **DataFrame**, which simplifies preprocessing, analysis, and exporting data to formats like CSV.
- 3. NumPy - pip install numpy
- Used for handling numerical data efficiently, such as standardizing features or calculating distances to cluster centers in anomaly detection.

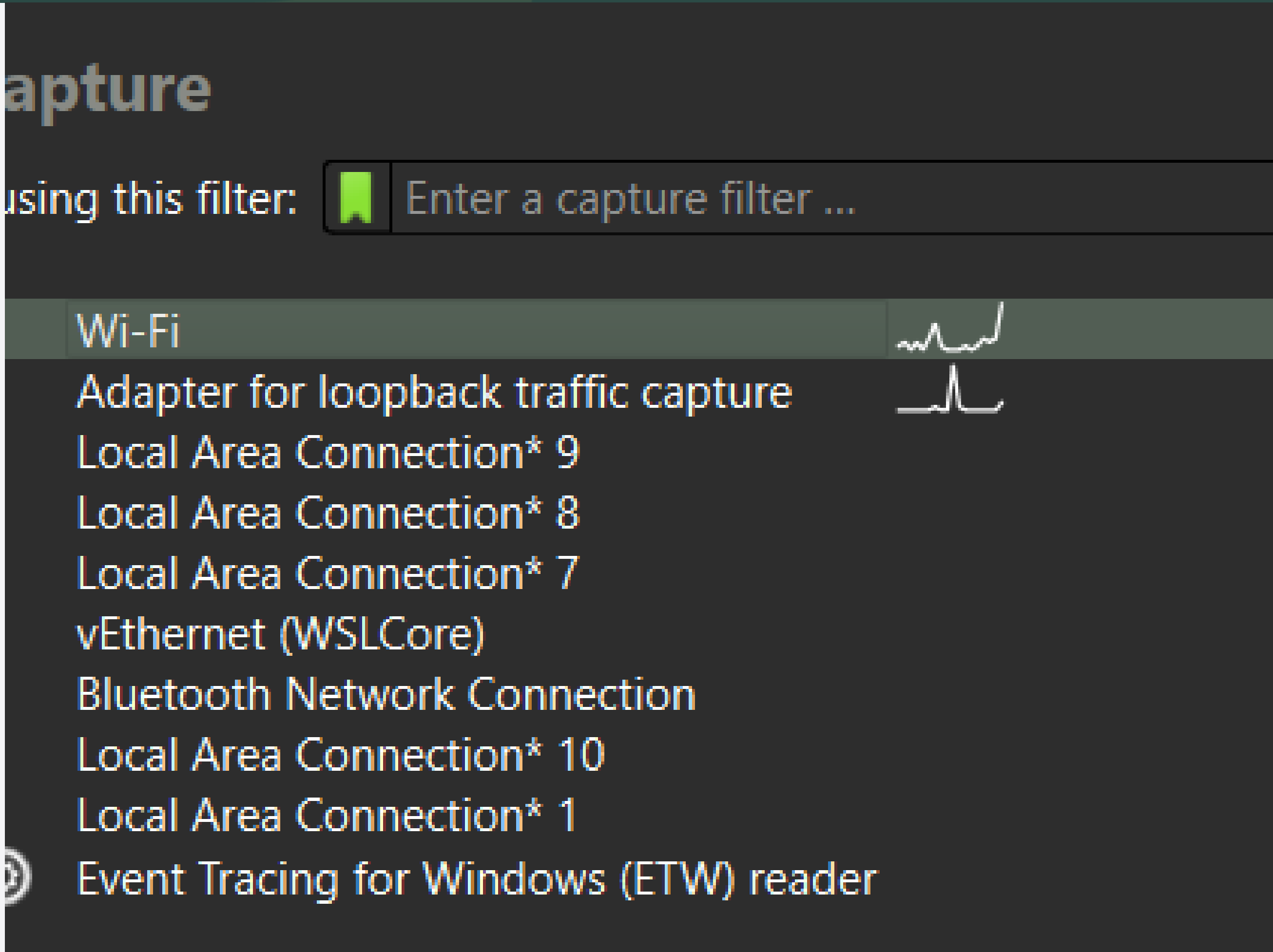
Install and import required libraries

- 4. Scikit-learn - pip install scikit-learn
- KMeans: Unsupervised clustering for anomaly detection.
- Random Forest Classifier: Supervised classification of network packets as normal or malicious.
- StandardScaler: Standardizes data (mean = 0, variance = 1) for machine learning models.
- Train-Test Split, Metrics: Splits data for training/testing and evaluates model performance.
- 5. Matplotlib - pip install matplotlib
- Used to create Visualizations.

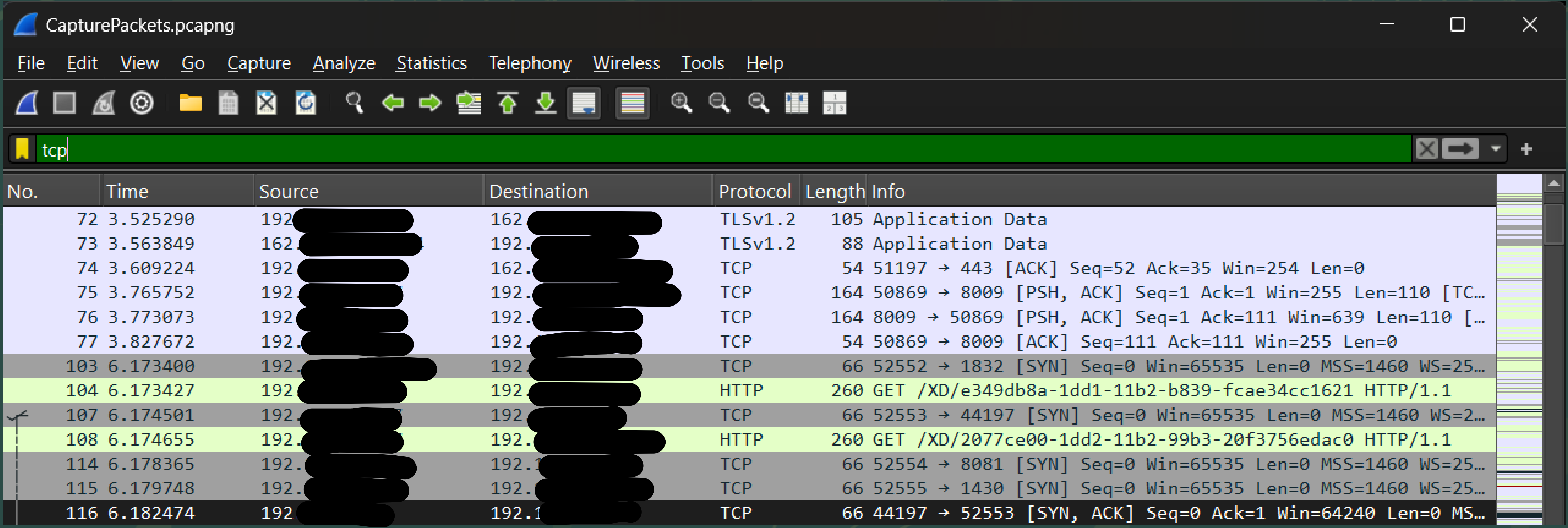
Install and import required libraries

- 6. Hashlib – Built in python library
- Converts IP addresses into numeric representations using hashing. This ensures IPs can be used in machine learning models
- 7. **Wireshark** - Wireshark · Go Deep
- - Application used to capture real network traffic (TCP packets) and save it to a .pcap file, which was analyzed.

Capture TCP Packets in WireShark



Capture TCP Packets



Load and Process packets

```
# Load the pcap file
packets = rdpcap(r"C:\Users\saint\OneDrive\Documents\ClientServerHW\CapturePackets.pcapng")
```

```
 # Extract relevant features from TCP packets
packet_data = []
for pkt in packets:
    if pkt.haslayer(TCP) and pkt.haslayer('IP'):  # Focus only on TCP packets
        packet_data.append({
            'Source': pkt['IP'].src,
            'Destination': pkt['IP'].dst,
            'Protocol': 'TCP',
            'Packet Length': len(pkt),
            'Timestamp': pkt.time
        })
```

```python
# Convert to a DataFrame
df = pd.DataFrame(packet_data)
print("Extracted TCP Packet Data:")
print(df.head())
```

```
Extracted TCP Packet Data:
            Source      Destination Protocol  Packet Length         Timestamp
0  192.[redacted]     192.[redacted]      TCP             66  1734468509.568721
1  192.[redacted]     192.[redacted]      TCP             66  1734468509.568885
2    192.[redacted]  192.[redacted]      TCP             66  1734468509.575903
3    192.[redacted]  192.[redacted]      TCP             66  1734468509.575903
4  192.[redacted]     192.[redacted]      TCP             54  1734468509.576146
```

```python
# Save to CSV for further processing
df.to_csv("tcp_packet_data.csv", index=False)
```

```python
# Load TCP packet data
df = pd.read_csv("tcp_packet_data.csv")
```

```python
# Encode IP addresses using hashing
df['Source'] = df['Source'].apply(lambda x: int(hashlib.md5(x.encode()).hexdigest(), 16) % 10**8)
df['Destination'] = df['Destination'].apply(lambda x: int(hashlib.md5(x.encode()).hexdigest(), 16) % 10**8)
```

Preprocess Data

Preprocess Data

```
# Select features and standardize
features = ['Source', 'Destination', 'Packet Length']
scaler = StandardScaler()
X = scaler.fit_transform(df[features])
```

```
print("Preprocessed Data (First 5 Rows):")
print(X[:5])
```

```
Preprocessed Data (First 5 Rows):
[[-0.03846188  0.39494923 -0.25184057]
 [-0.03846188  0.39494923 -0.25184057]
 [ 0.29607053  0.01414707 -0.25184057]
 [ 0.29607053  0.01414707 -0.25184057]
 [-0.03846188  0.39494923 -0.26038741]]
```

KMeans for Anomaly Detection

```
# Apply KMeans clustering
kmeans = KMeans(n_clusters=2, random_state=42)  # Assume 2 clusters: normal and anomalous
df['Cluster'] = kmeans.fit_predict(X)
```

```
# Compute distance to cluster centers
distances = kmeans.transform(X).min(axis=1)
threshold = np.percentile(distances, 95)  # 95th percentile as anomaly threshold
df['Anomaly'] = distances > threshold
```

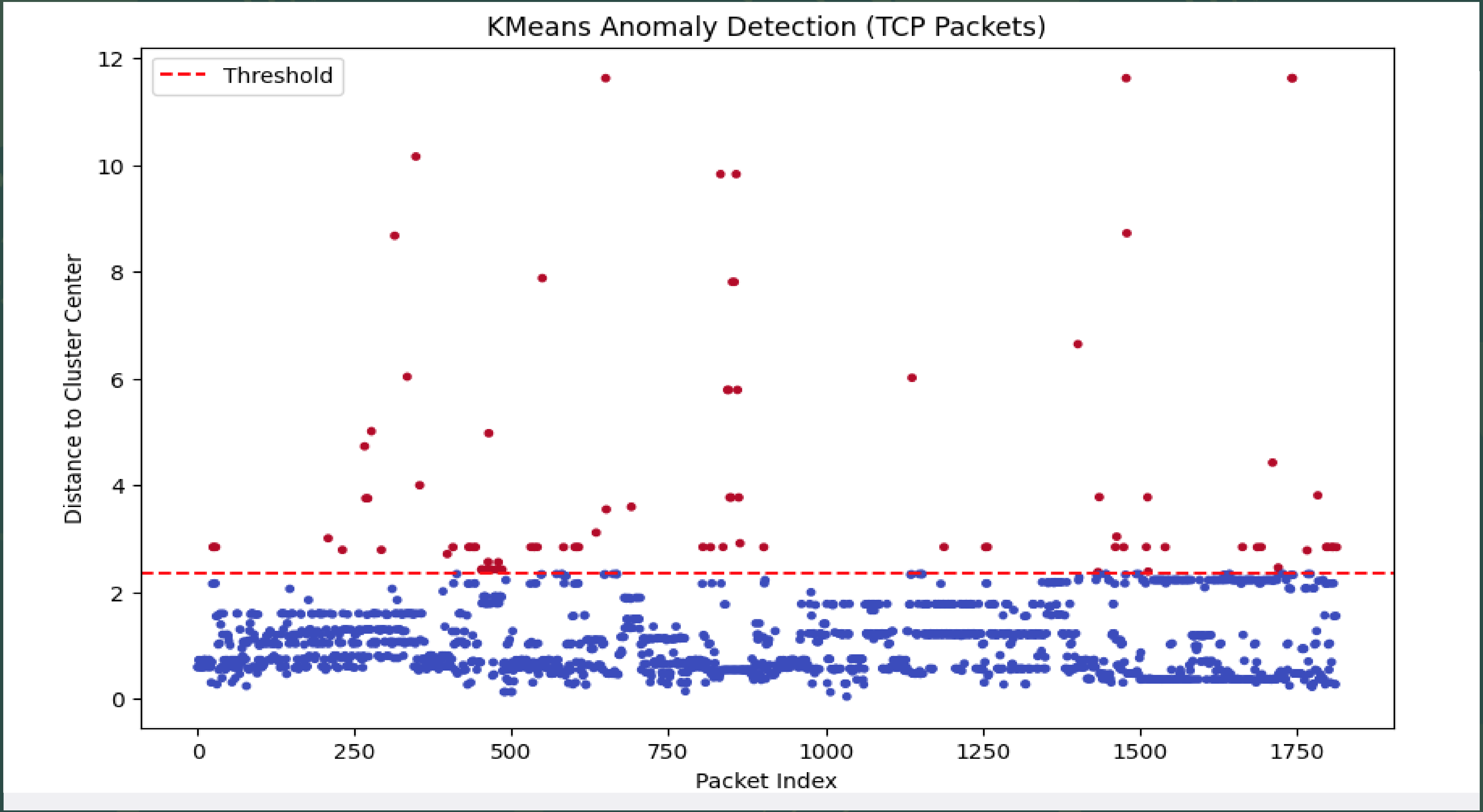
KMeans for Anomaly Detection

```python
# Display results
print(f"Number of anomalies detected: {df['Anomaly'].sum()}")
print(df[df['Anomaly']])
```

```
Number of anomalies detected: 88
        Source  Destination Protocol  Packet Length     Timestamp  Cluster  \
25    55305822      1177113      TCP             54  1.734469e+09        0
26    55305822      1177113      TCP             85  1.734469e+09        0
29    55305822      1177113      TCP             54  1.734469e+09        0
208   25540421     55305822      TCP           4434  1.734469e+09        0
231   31528732     55305822      TCP           4236  1.734469e+09        0
...        ...          ...      ...            ...           ...      ...
1798  55305822      1177113      TCP             54  1.734469e+09        0
1800  55305822      1177113      TCP             66  1.734469e+09        0
1807  55305822      1177113      TCP             54  1.734469e+09        0
1808  55305822      1177113      TCP             85  1.734469e+09        0
1814  55305822      1177113      TCP             82  1.734469e+09        0

      Anomaly
25       True
26       True
29       True
208      True
231      True
...       ...
1798     True
1800     True
1807     True
1808     True
1814     True
```

KMeans for Anomaly Detection

- 88 anomalies detected: Out of the total data, 88 points exceeded the anomaly threshold.
- The table displays:
- Source: Source address.
- Destination: Destination address.
- Protocol: Protocol type (e.g., TCP).
- Packet Length: Size of the network packet.
- Timestamp: Time when the packet was captured.
- Cluster: Cluster label assigned by KMeans.
- Anomaly: True indicates an anomalous point.



KMeans Anomaly Detection (TCP Packets)
Threshold
Distance to Cluster Center
Packet Index
0
2
4
6
8
10
12
0
250
500
750
1000
1250
1500
1750

Train Random Forest

[21]:
```
# Label anomalies as 1 (malicious), others as 0
df['Label'] = df['Anomaly'].astype(int)
```

[22]:
```
# Prepare features and target
X = df[['Source', 'Destination', 'Packet Length']]
y = df['Label']
```

[23]:
```
# Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

[24]:
```
# Train Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

[24]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

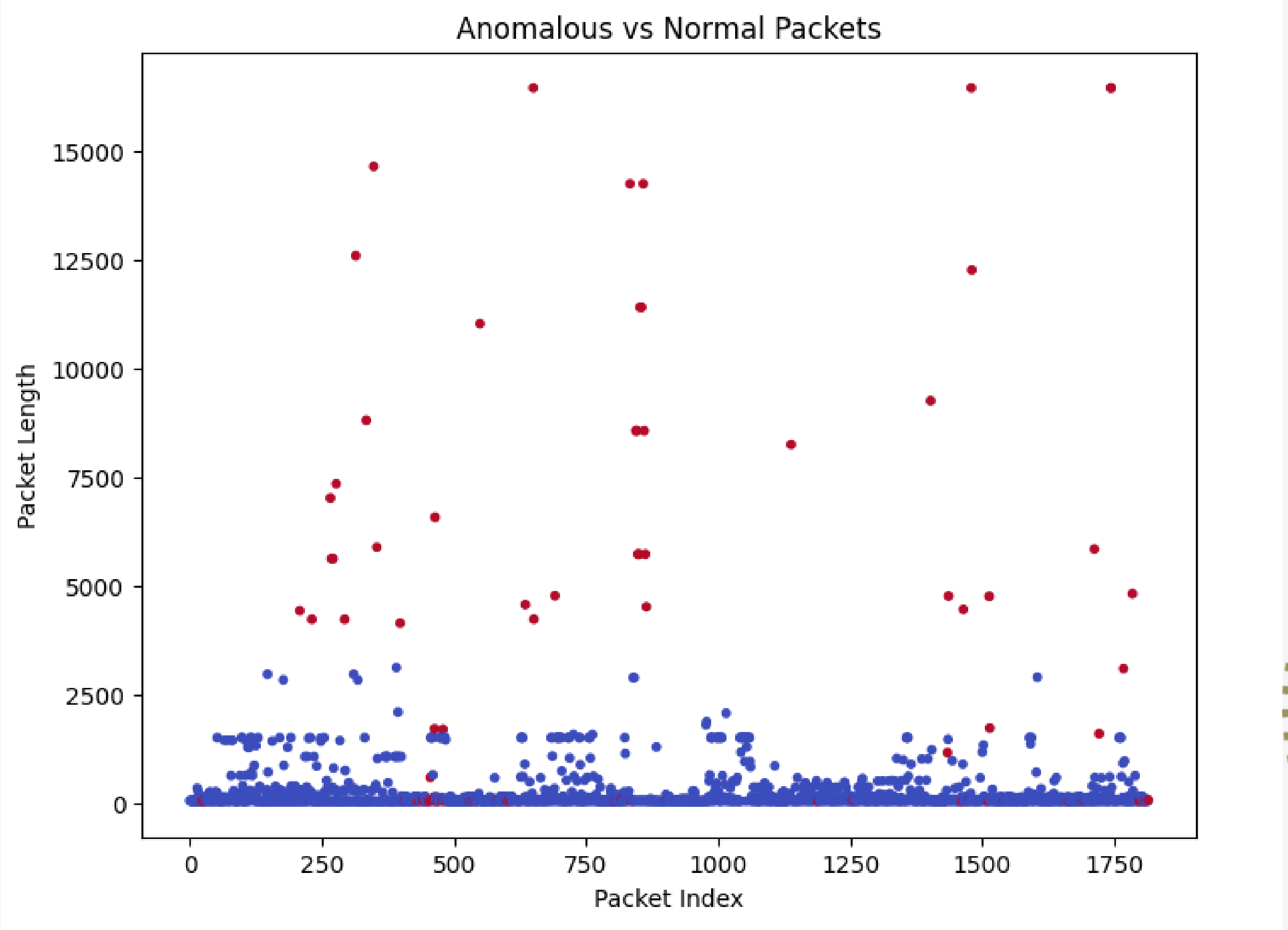
Train Random Forest

```
# Evaluate the model
y_pred = rf_model.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       515
           1       1.00      0.97      0.98        30

    accuracy                           1.00       545
   macro avg       1.00      0.98      0.99       545
weighted avg       1.00      1.00      1.00       545

Confusion Matrix:
[[515   0]
 [  1  29]]
```

Visualize Insights

- The **Random Forest classifier** correctly identified the anomalous packets (red dots) based on deviations in packet size.
- **Packet Length** appears to be a strong feature for anomaly detection, as most anomalies correlate with larger-than-usual packet sizes.
- This scatter plot demonstrates how the model effectively differentiates anomalies (outliers) from normal traffic.



Prediction

- The trained Random Forest model identified this new packet as normal, based on its features. The model effectively leveraged prior knowledge (patterns in Packet Length and IP mappings) to classify the packet.

```
# Step 6: Predict on New Packet Data
```

```
# Simulated new TCP packet
new_packet = pd.DataFrame({
    'Source': [int(hashlib.md5("192[redacted]".encode()).hexdigest(), 16) % 10**8],
    'Destination': [int(hashlib.md5("10.[redacted]".encode()).hexdigest(), 16) % 10**8],
    'Packet Length': [700]
})

# Predict if the packet is anomalous
prediction = rf_model.predict(new_packet)
print("New Packet Classification:", "Malicious" if prediction[0] == 1 else "Normal")
```

```
New Packet Classification: Normal
```

The End