Names:

Hannah Emad 2205123

Mariam Mostafa 2205084

Nada Mohamed 2205173

Cybersecurity Analysis and Anomaly Detection Code

Overview

This project analyzes cybersecurity data to detect anomalies in DNS traffic and evaluate patterns using machine learning and statistical methods.

It includes:

- Data cleaning and feature engineering
- Exploratory Data Analysis (EDA) with visualizations
- Anomaly detection using Isolation Forest
- Supervised machine learning models (Random Forest, SVM, and ANN)

Libraries and Warnings

```
# Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

```
# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
```

Import essential libraries for data manipulation, visualization, and machine learning while suppressing warnings.

Data Loading and Initial Exploration

```
[ ] df = pd.read_csv("/content/cybersecurity_attacks.csv")
```

```
df.head(5)
```

Packet Type	Traffic Type	Payload Data	...	Action Taken	Severity Level	User Information	Device Information	Network Segment	Geo-location Data	Proxy Information	Firewall Logs	IDS/IPS Alerts	Log Source
Data	HTTP	Qui natus odio asperiores nam. Optio nobis ius...	...	Logged	Low	Reyansh Dugal	Mozilla/5.0 (compatible; MSIE 8.0; Windows NT ...	Segment A	Jamshedpur, Sikkim	150.9.97.135	Log Data	NaN	Server
Data	HTTP	Aperiam quos modi officiis veritatis rem. Omni...	...	Blocked	Low	Sumer Rana	Mozilla/5.0 (compatible; MSIE 8.0; Windows NT ...	Segment B	Bilaspur, Nagaland	NaN	Log Data	NaN	Firewall
Control	HTTP	Perferendis sapiente vitae soluta. Hic delectu...	...	Ignored	Low	Himmat Karpe	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...	Segment C	Bokaro, Rajasthan	114.133.48.179	Log Data	Alert Data	Firewall
Data	HTTP	Totam maxime beatae expedita	...	Blocked	Medium	Fateh Kibe	Mozilla/5.0 (Macintosh; PPC Mac OS X 10_11_5;	Segment B	Jaunpur, Rajasthan	NaN	NaN	Alert Data	Firewall

Missing values

```
df.isnull().sum().sort_values(ascending=False)
```

	0
Alerts/Warnings	20067
IDS/IPS Alerts	20050
Malware Indicators	20000
Firewall Logs	19961
Proxy Information	19851
Attack Type	0
Geo-location Data	0
Network Segment	0
Device Information	0
User Information	0
Severity Level	0
Action Taken	0
Attack Signature	0
Timestamp	0
Source IP Address	0
Anomaly Scores	0

Handling Missing Values

```
[ ] # Determine recent activity
    df['Alerts/Warnings'] = df['Alerts/Warnings'].apply(lambda x: 'yes' if x == 'Alert Triggered' else 'no')

    df['Malware Indicators'] = df['Malware Indicators'].apply(lambda x: 'No Detection' if pd.isna(x) else x)

    df['Proxy Information'] = df['Proxy Information'].apply(lambda x: 'No proxy' if pd.isna(x) else x)

    df['Firewall Logs'] = df['Firewall Logs'].apply(lambda x: 'No Data' if pd.isna(x) else x)

    df['IDS/IPS Alerts'] = df['IDS/IPS Alerts'].apply(lambda x: 'No Data' if pd.isna(x) else x)
```

Check sum of missing values

```python
#check sum of missing values
df.isnull().sum().sort_values(ascending=False)
#All Missing Values are removed.
```

	0
Timestamp	0
Attack Type	0
IDS/IPS Alerts	0
Firewall Logs	0
Proxy Information	0
Geo-location Data	0
Network Segment	0
Device Information	0
User Information	0
Severity Level	0
Action Taken	0
Attack Signature	0
Alerts/Warnings	0
Source IP Address	0
Anomaly Scores	0

Extracting device/OS information from the 'Device Information' column using regular expressions.

```python
import re
# OS and device patterns to search for
patterns = [
    r'Windows',
    r'Linux',
    r'Android',
    r'iPad',
    r'iPod',
    r'iPhone',
    r'Macintosh',
]

def extract_device_or_os(user_agent):
    for pattern in patterns:
        match = re.search(pattern, user_agent, re.I)  # re.I makes the search case-insensitive
        if match:
            return match.group()
    return 'Unknown'  # Return 'Unknown' if no patterns match

# Extract device or OS
df['Device/OS'] = df['Device Information'].apply(extract_device_or_os)

# Display the extracted device or OS
df['Device/OS']
```

Output:

	Device/OS
0	Windows
1	Windows
2	Windows
3	Macintosh
4	Windows
...	...
39995	iPad
39996	Windows
39997	Windows
39998	Linux
39999	iPod

40000 rows × 1 columns

Converts the 'Timestamp' column to a datetime format and extracts the year and month from it, creating new columns (Year and Month) to store this information

```
[ ] def extract_time_features(df, Timestamp):
        # Convert timestamp column to datetime if it's not already
        df[Timestamp] = pd.to_datetime(df[Timestamp])

        # Extract time features
        df['Year'] = df[Timestamp].dt.year
        df['Month'] = df[Timestamp].dt.month
        df['Day'] = df[Timestamp].dt.day
        df['Hour'] = df[Timestamp].dt.hour
        df['Minute'] = df[Timestamp].dt.minute
        df['Second'] = df[Timestamp].dt.second
        df['DayOfWeek'] = df[Timestamp].dt.dayofweek

        return df
```

Output:

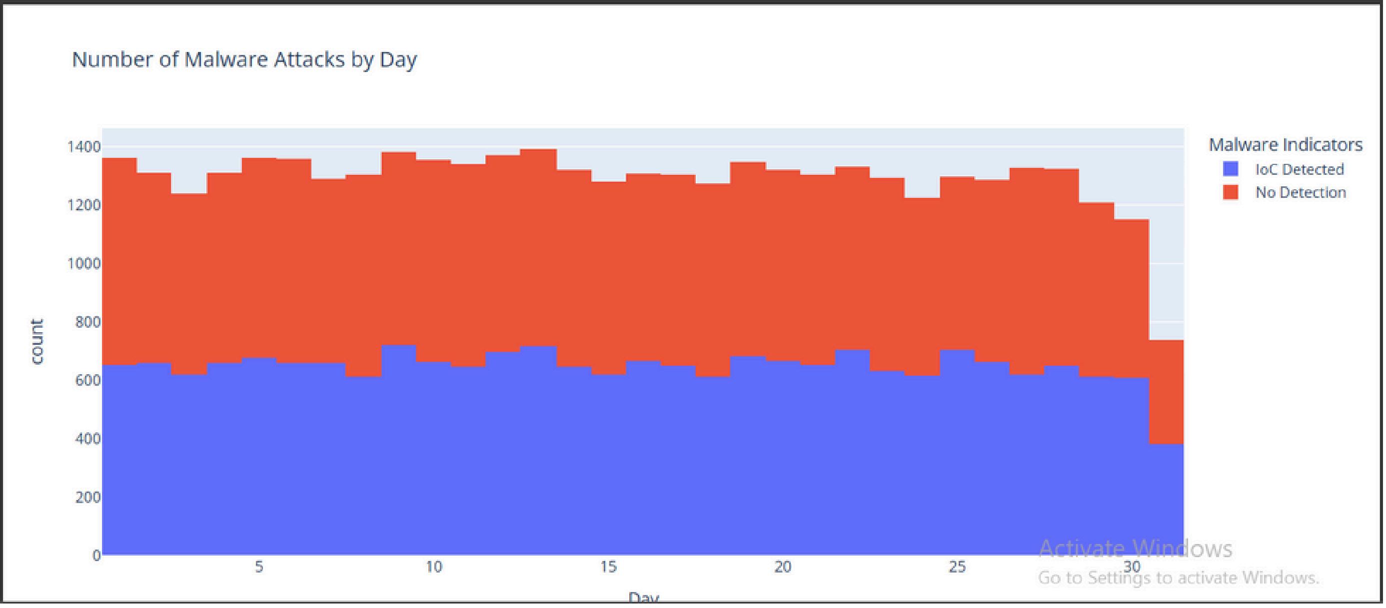
```
            Timestamp Source IP Address Destination IP Address  Source Port  \
0 2023-05-30 06:33:58    103.216.15.12           84.9.164.252        31225
1 2020-08-26 07:08:30   78.199.217.198         66.191.137.154        17245
2 2022-11-13 08:23:25     63.79.210.48          198.219.82.17        16811
3 2023-07-02 10:38:46    163.42.196.10        101.228.192.255        20018
4 2023-07-16 13:11:07     71.166.185.76        189.243.174.238         6131

   Destination Port Protocol  Packet Length Packet Type Traffic Type  \
0             17616     ICMP            503        Data         HTTP
1             48166     ICMP           1174        Data         HTTP
2             53600      UDP            306     Control         HTTP
3             32534      UDP            385        Data         HTTP
4             26646      TCP           1462        Data          DNS

                                        Payload Data  ... Log Source  Browser  \
0  Qui natus odio asperiores nam. Optio nobis ius...  ...     Server  Mozilla
1  Aperiam quos modi officiis veritatis rem. Omni...  ...   Firewall  Mozilla
2  Perferendis sapiente vitae soluta. Hic delectu...  ...   Firewall  Mozilla
3  Totam maxime beatae expedita explicabo porro l...  ...   Firewall  Mozilla
4  Odit nesciunt dolorem nisi iste iusto. Animi v...  ...   Firewall  Mozilla

   Device/OS  Year Month Day Hour Minute Second DayOfWeek
0    Windows  2023     5  30    6     33     58         1
1    Windows  2020     8  26    7      8     30         2
2    Windows  2022    11  13    8     23     25         6
3  Macintosh  2023     7   2   10     38     46         6
4    Windows  2023     7  16   13     11      7         6

[5 rows x 33 columns]
```

Exploratory Data Analysis (EDA)

Visualizations are created using Plotly to understand malware distributions, traffic patterns, and platform usage.

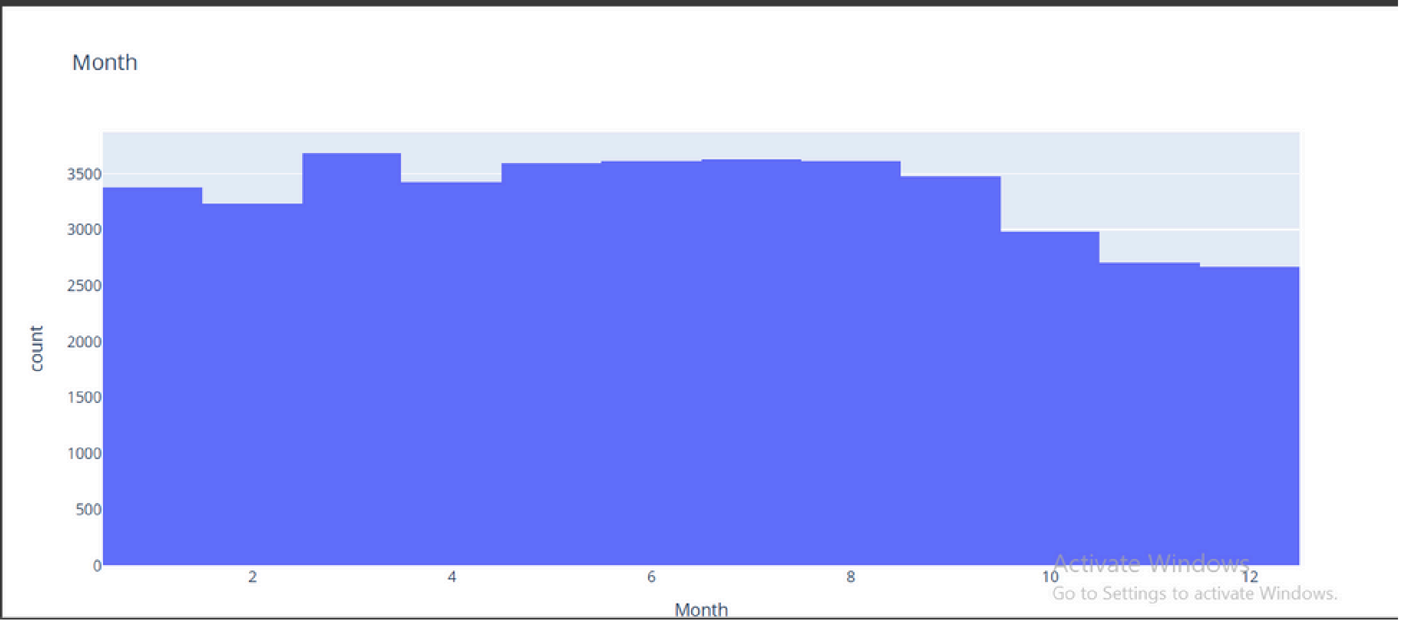
Checking the Day Column ploting with plotly

```python
# Checking the Day Column ploting with plotly
plt = px.histogram(df, x = 'Day', color = 'Malware Indicators', title = 'Number of Malware Attacks by Day')
plt.show()
```



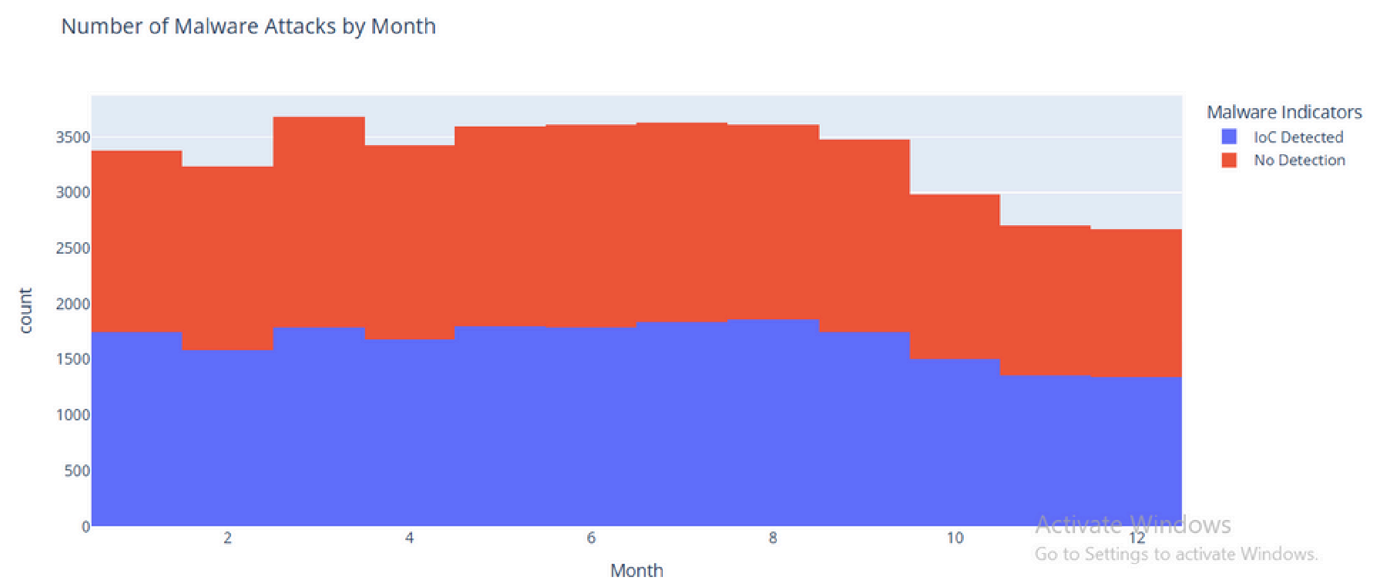
Month Distribution

```python
# month Distribution
plt = px.histogram(df, x = 'Month', title = 'Month')
plt.show()
```



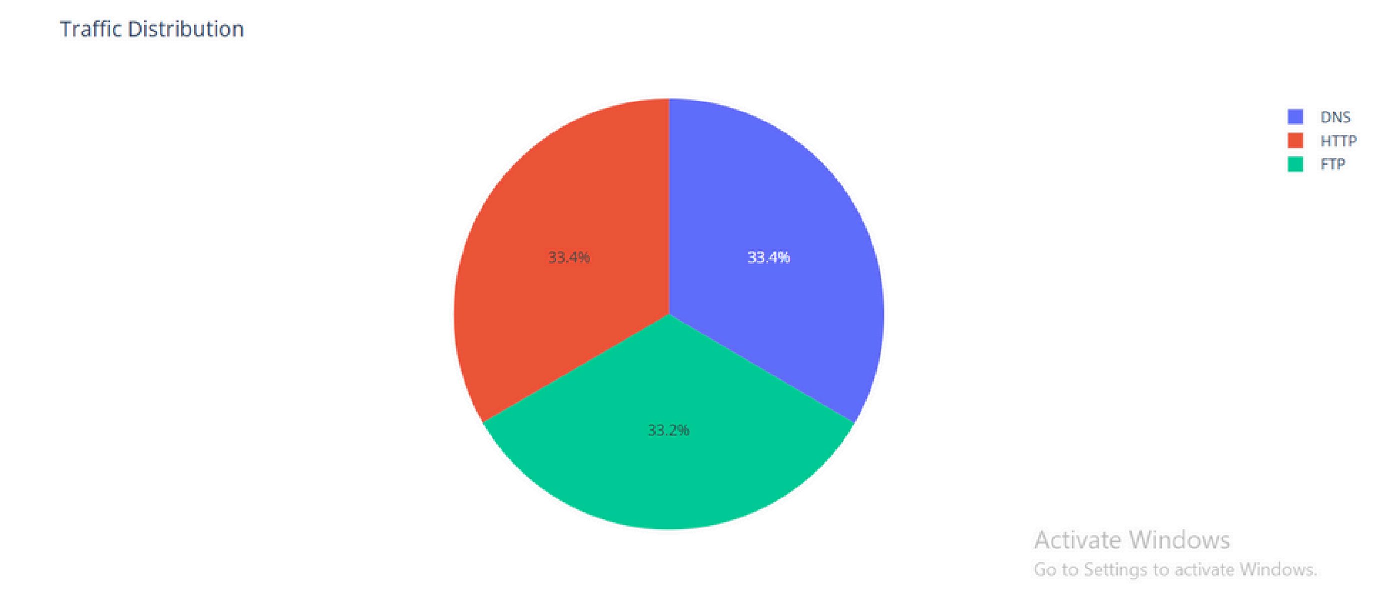
Checking the Month Column ploting with plotly

```python
# Checking the Month Column ploting with plotly
plt = px.histogram(df, x = 'Month', color = 'Malware Indicators', title = 'Number of Malware Attacks by Month')
plt.show()
```



Traffic Distribution

```python
# Traffic Distribution
plt = px.pie(df, names = 'Traffic Type', title = 'Traffic Distribution')
plt.show()
```

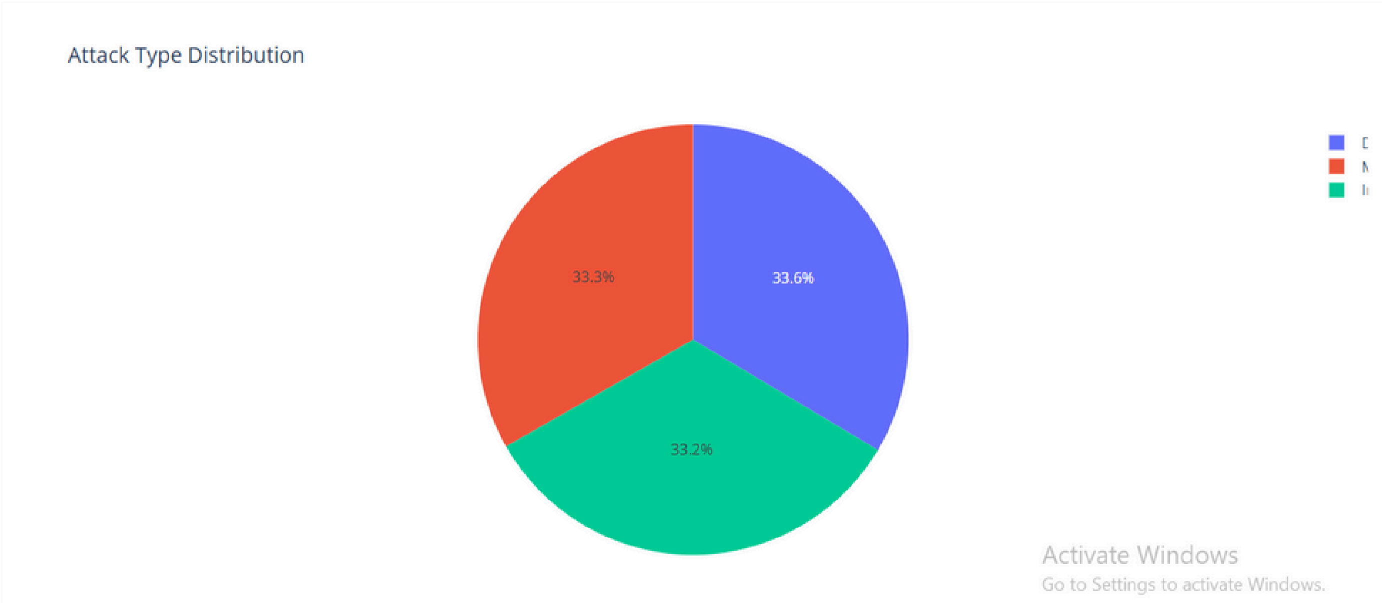


Ploting the Traffic Type distribution with Bar Chart Using Plotly

```
# Ploting the Traffic Type distribution with Bar Chart Using Plotly
plt = px.histogram(df, x = 'Traffic Type', color = 'Malware Indicators', title = 'Number of Malware Attacks by Traffic Type')
plt.show()
```

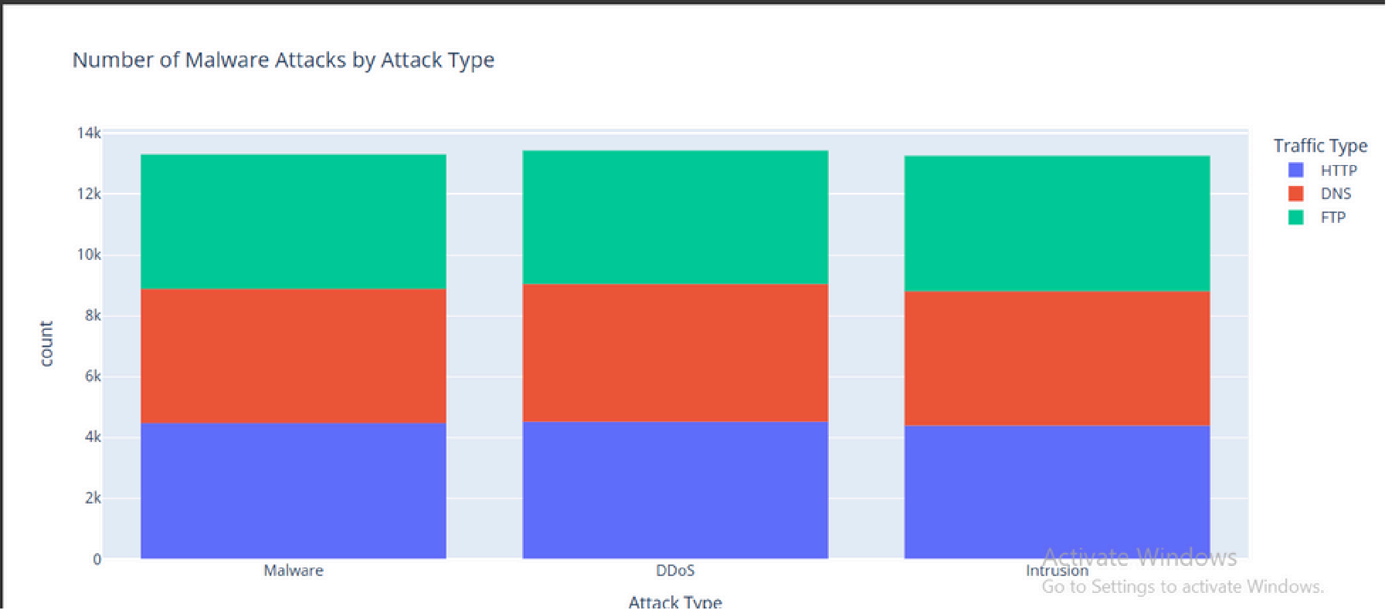
Attack Type Distribution

```
# Attack Type Distribution
plt = px.pie(df, names = 'Attack Type', title = 'Attack Type Distribution')
plt.show()
```



Checking the attack types distribution with Bar Chart Using Plotly

```python
# Checking the attack types distribution with Bar Chart Using Plotly
plt = px.histogram(df, x='Attack Type', color='Traffic Type', title='Number of Malware Attacks by Attack Type')
plt.show()
```



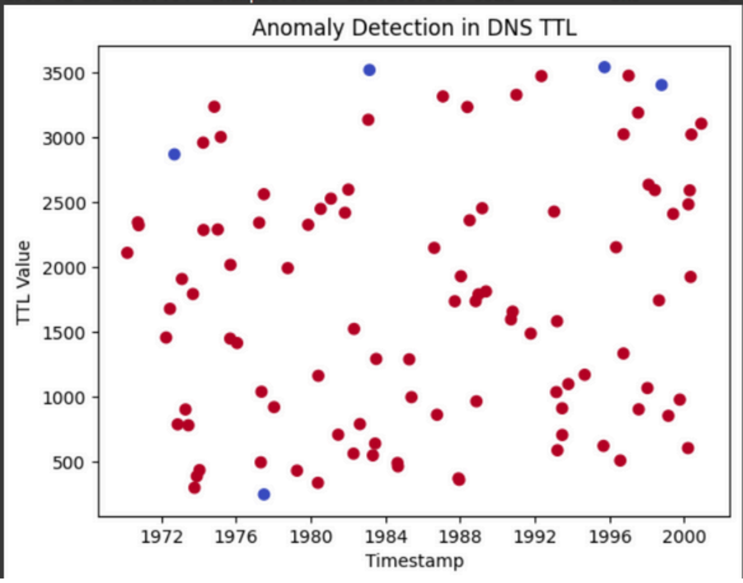
Anomaly Detection

An Isolation Forest model detects anomalies in DNS traffic based on TTL values

```python
from sklearn.ensemble import IsolationForest

# Select TTL as the feature for anomaly detection
X = filtered_df[['TTL']]

# Train an Isolation Forest model
model = IsolationForest(contamination=0.05)  # Assume 5% of data are anomalies
filtered_df['Anomaly'] = model.fit_predict(X)

# Mark anomalies as -1 and normal points as 1
anomalies = filtered_df[filtered_df['Anomaly'] == -1]
print(f"Detected anomalies: {anomalies}")

# Plot anomalies
# Reset the index to make 'Timestamp' a column again
filtered_df = filtered_df.reset_index()
plt.scatter(filtered_df['Timestamp'], filtered_df['TTL'], c=filtered_df['Anomaly'], cmap='coolwarm')
plt.title('Anomaly Detection in DNS TTL')
plt.xlabel('Timestamp')
plt.ylabel('TTL Value')
plt.show()
```

Output: **The anomalies were plotted to visualize potential issues in the DNS traffic**

```
Detected anomalies:                                  Query      Response    TTL Traffic Type  Anomaly
Timestamp
1977-07-04 09:36:23  example.com   192.168.1.1   245          DNS       -1
1972-09-15 20:28:34   github.com  140.82.121.3  2870          DNS       -1
1995-09-27 03:32:08   google.com  172.217.5.68  3543          DNS       -1
1998-10-17 05:42:38   github.com  140.82.121.3  3405          DNS       -1
1983-02-25 12:57:04  example.com   192.168.1.1  3521          DNS       -1
```



Calculate the entropy

```python
import numpy as np
from collections import Counter

# Use 'Query' column for entropy calculation
query_values = filtered_df['Query']

# Calculate frequency distribution of query values
query_counts = Counter(query_values)

# Calculate the total number of queries
total_queries = len(query_values)

# Calculate the entropy
entropy = -sum((count / total_queries) * np.log2(count / total_queries)
               for count in query_counts.values())

print(f"Query Entropy: {entropy}")
```

```
Query Entropy: 1.5534687653756747
```

Data Preprocessing for Machine Learning

Categorical features like 'Query', 'Response', and 'Traffic Type' were encoded using LabelEncoder to convert them into numerical format suitable for machine learning models

```python
# Encode Query, Response, and Traffic Type using LabelEncoder
label_enc_query = LabelEncoder()
filtered_df['Query_encoded'] = label_enc_query.fit_transform(filtered_df['Query'])
```

Additional time features were extracted for further modeling, such as Year, Month, Day, Hour, and other temporal attributes.

```python
label_enc_response = LabelEncoder()
filtered_df['Response_encoded'] = label_enc_response.fit_transform(filtered_df['Response'])

label_enc_traffic = LabelEncoder()
filtered_df['Traffic_Type_encoded'] = label_enc_traffic.fit_transform(filtered_df['Traffic Type'])

# Drop original categorical columns
filtered_df = filtered_df.drop(columns=['Query', 'Response', 'Traffic Type'])
```

We use Machine Learning Models

Three supervised machine learning models are implemented

- **Random Forest Classifier**
- **Artificial Neural Network (ANN)**
- **Support Vector Machine (SVM)**

Random Forest Classifier

Trains and evaluates a Random Forest Classifier model for classification tasks using scaled features.

Feature Scaling:

- **Standardizes the data so that all features have a mean of 0 and a standard deviation of 1, which is important for ensuring the model performs optimally.**

Model Initialization:

A RandomForestClassifier is initialized with 100 trees (n_estimators=100) and a fixed random_state for reproducibility.

Model Training:

- **The model is trained using the scaled training data (X_train_scaled and y_train).**

Predictions:

- **The trained model predicts the target variable on the scaled test set (X_test_scaled).**

Model Evaluation:

- **Accuracy is computed using accuracy_score() by comparing the predicted values (y_pred) with the actual test labels (y_test).**

The accuracy result is printed to evaluate the model's performance.

```
# the model RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train_scaled, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Make predictions
y_pred = model.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Feature importance
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

# Print the feature importance
print("\nFeature Importance:")
for f in range(X_train.shape[1]):
    print(f"{filtered_df.columns[indices[f]]}: {importances[indices[f]]:.4f}")
```

```
Accuracy: 0.95
```


Artificial Neural Network

we will builds, trains, and evaluates an Artificial Neural Network (ANN) model for binary classification tasks. The following steps outline the process.

Building the ANN Model

A Sequential model is created for a layer-by-layer feedforward network.

Input Layer: The first Dense layer has 64 neurons, ReLU activation, and accepts input features (input_dim=X_train.shape[1]).

Hidden Layers:

- **Second layer: 32 neurons with ReLU activation.**
- **Third layer: 16 neurons with ReLU activation.**

Output Layer: A single neuron with sigmoid activation for binary classification, outputting probabilities between 0 and 1.

```
[ ] # Build the ANN model
model = Sequential()

# Add input layer (input_dim = number of features in the dataset)
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))

# Add hidden layers
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=16, activation='relu'))

# Add output layer (for binary classification, use 'sigmoid' activation)
model.add(Dense(units=1, activation='sigmoid'))
```

Compiling the Model

- **Optimizer: adam is used for efficient gradient-based optimization.**

- **Loss Function**: binary_crossentropy is ideal for binary classification problems.
- **Metrics**: Accuracy is used to track performance during training.

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Training the Model

- The model trains using the fit()method with:
- Training data (X_train_scaled, y_train).
- 20 epochs and a batch size of 32 for iterations.
- Validation data (X_test_scaled, y_test) to monitor test performance during training.

```
[ ] # Train the model
    history = model.fit(X_train_scaled, y_train, epochs=20, batch_size=32, validation_data=(X_test_scaled, y_test))
```

Predictions and Evaluation

- The model predicts probabilities for the test data.
- Probabilities are converted to binary values (0 or 1) using a threshold of 0.5.
- accuracy_score() evaluates the model's accuracy by comparing predictions with actual test labels.

```
# Make predictions
y_pred = model.predict(X_test_scaled)
# Convert probabilities to binary (0 or 1)
y_pred = (y_pred > 0.5)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Output:

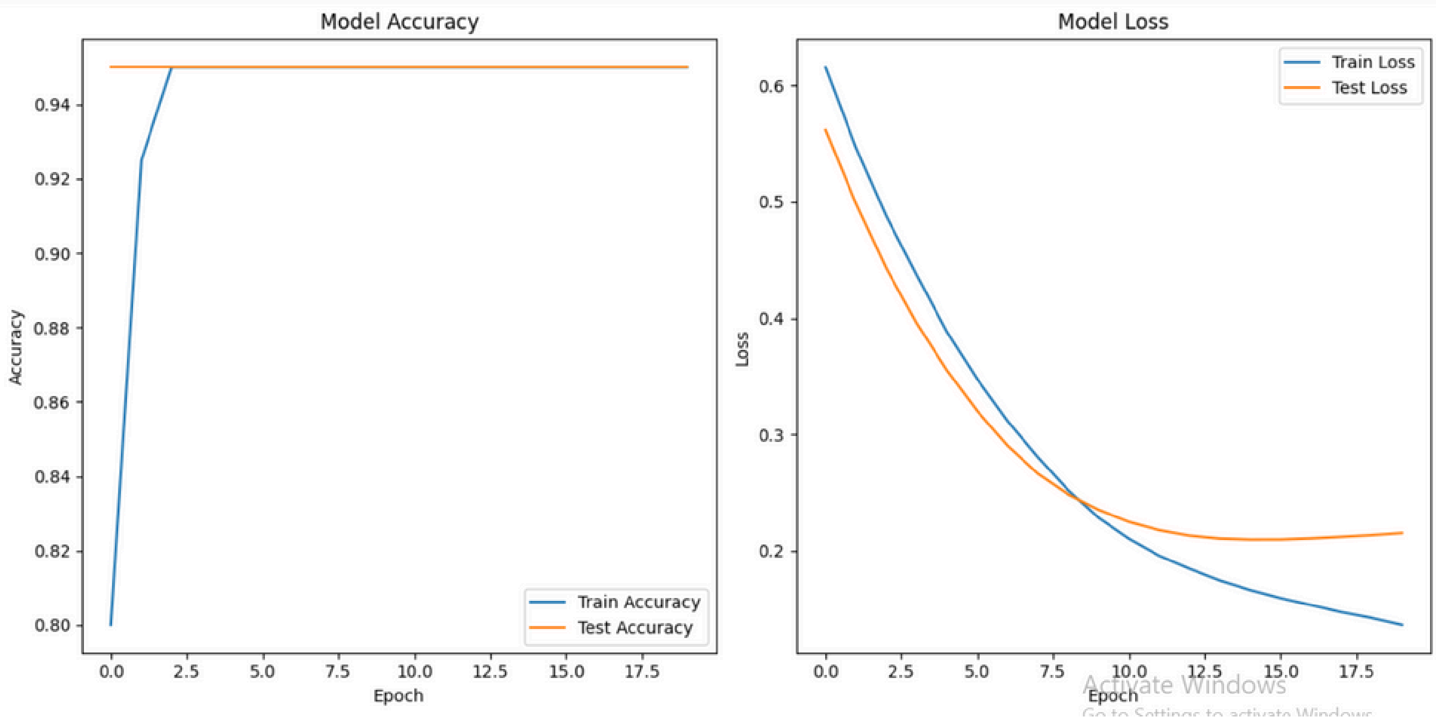
```
Epoch 1/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 239ms/step - accuracy: 0.9719 - loss: 0.0645 - val_accuracy: 0.9500 - val_loss: 0.2579
Epoch 2/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.9641 - loss: 0.0724 - val_accuracy: 0.9500 - val_loss: 0.2607
Epoch 3/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9758 - loss: 0.0578 - val_accuracy: 0.9500 - val_loss: 0.2637
Epoch 4/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.9719 - loss: 0.0560 - val_accuracy: 0.9500 - val_loss: 0.2666
Epoch 5/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.9836 - loss: 0.0373 - val_accuracy: 0.9500 - val_loss: 0.2696
Epoch 6/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9758 - loss: 0.0440 - val_accuracy: 0.9500 - val_loss: 0.2727
Epoch 7/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9836 - loss: 0.0386 - val_accuracy: 0.9500 - val_loss: 0.2757
Epoch 8/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9719 - loss: 0.0528 - val_accuracy: 0.9500 - val_loss: 0.2786
Epoch 9/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step - accuracy: 0.9719 - loss: 0.0479 - val_accuracy: 0.9500 - val_loss: 0.2816
Epoch 10/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9797 - loss: 0.0383 - val_accuracy: 0.9500 - val_loss: 0.2847
Epoch 11/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.9758 - loss: 0.0417 - val_accuracy: 0.9500 - val_loss: 0.2877
Epoch 12/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.9719 - loss: 0.0402 - val_accuracy: 0.9500 - val_loss: 0.2905
Epoch 13/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.9898 - loss: 0.0386 - val_accuracy: 0.9500 - val_loss: 0.2935
Epoch 14/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 0.9898 - loss: 0.0279 - val_accuracy: 0.9500 - val_loss: 0.2965
Epoch 15/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9898 - loss: 0.0271 - val_accuracy: 0.9500 - val_loss: 0.2996
Epoch 16/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9898 - loss: 0.0265 - val_accuracy: 0.9500 - val_loss: 0.3026
Epoch 17/20
```


Visualization of Performance

- **Accuracy Plot:** Shows the model's accuracy on training and test data across epochs.

- **Loss Plot:** Displays the model's training and test loss over epochs.
- **Visualizations:** help assess whether the model overfits or underfits.

```
# Plotting the loss and accuracy during training
plt.figure(figsize=(12, 6))

# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



Conclusion

- **Model Structure**: The ANN consists of an input layer, 2 hidden layers (32 and 16 neurons), and an output layer with sigmoid activation for binary classification.
- **Training and Evaluation:** The model uses the adam optimizer and binary_crossentropy loss. It achieves accuracy by converting predictions to binary values.
- **Performance Visualization:** Accuracy and loss plots are generated to monitor training and validation trends over epochs.

Support Vector Machine

We will trains, evaluates, and visualizes the performance of a Support Vector Machine (SVM) classifier for a classification

Training the SVM Model

- **The SVC class from sklearn is used to create the Support Vector Machine classifier.**
- **Kernel: linear is specified as the kernel type**
- **Random State: Set to 30 for reproducibility.**
- **The model is trained using the scaled training data (X_train_scaled) and corresponding labels (y_train).**

```
# Train an SVM classifier
svm_model = SVC(kernel='linear', random_state=30)  # You can use 'linear' or other kernels like 'rbf', 'poly'

# Fit the model on training data
svm_model.fit(X_train_scaled, y_train)
```

Making Predictions

- **The trained SVM model predicts class labels for the test dataset (X_test_scaled).**
- **Predicted results are stored in y_pred_svm.**

```
# Make predictions
y_pred_svm = svm_model.predict(X_test_scaled)
```

Model Evaluation:

- **Accuracy: The accuracy_score function calculates the proportion of correct predictions.**

```
# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {accuracy_svm:.2f}")
```

Performance Visualization

- **A bar plot is generated to display precision, recall, and F1-score for each class.**
- **X-axis: Class labels.**
- **Y-axis: Scores for precision, recall, and F1.**
- **This helps identify which classes are well-predicted and where improvements are needed.**

```
# Get classification report as a DataFrame
report = classification_report(y_test, y_pred_svm, output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Plot precision, recall, F1-score
report_df[['precision', 'recall', 'f1-score']].plot(kind='bar', figsize=(10, 6))
plt.title("SVM Classification Metrics")
plt.xlabel("Class")
plt.ylabel("Score")
plt.xticks(rotation=0)
plt.show()
```

Conclusion

- **SVM Model:** Trained a Support Vector Machine with a linear kernel to classify the dataset.
- **Evaluation:** The model's performance is evaluated using accuracy, precision, recall, and F1-score.
- **Visualization:** A bar chart displays classification metrics (precision, recall, F1-score) for each class.

Conclusion Of All code

This analysis covered several important aspects of the cybersecurity dataset, focusing on DNS traffic, attack patterns, anomaly detection, and feature extraction for potential machine learning applications. The steps outlined in this report provide insights into trends and irregularities in DNS queries, which can assist in detecting malicious activities and improving cybersecurity monitoring systems.

Future Work

- **Modeling:** **Future steps can involve training machine learning models to predict attack types based on extracted features.**
- **Anomaly Detection Refinement:** **The anomaly detection process could be improved by tuning the Isolation Forest model or experimenting with other methods.**
- **Real-Time Analysis:** **Applying this methodology to real-time data streams could help identify attacks in real-time.**