Bilel RAHMOUNI

LAB 2 - IOT Classification

Introduction

The aim of this lab was to detect IoT devices by classifying network traffic using two machine learning algorithms: Logistic Regression (LR) and K-Nearest Neighbors (KNN). The process involved extracting features from pcap files, exploring the dataset, and training/evaluating the models to identify which algorithm performed better in classifying IoT device types based on network traffic.

Part 1: Feature Extraction from pcap Files

We started by uploading pcap files, which store network traffic data. To automate feature extraction from these files, we used a Python class called PcapProcessor.

```
from google.colab import files
files.upload()
```

The key steps in this process were:

Installing Wireshark (tshark): This tool was necessary for extracting features from the network traffic captured in pcap files.

```
[ ]  !apt-get install tshark
```

Running Feature Extraction: Using the PcapProcessor, we filtered pcap files and generated a CSV file named label_feature_IOT.csv. This CSV contained key network traffic features such as IP length, TTL, Protocol, Source/Destination Ports, and TCP lengths.

```
import os
import glob

processor = PcapProcessor(pcap_folder='/content/filtered_pcap')
processor.process()
```

Previewing the Data: Once extracted, the CSV file was loaded into a Pandas DataFrame to preview the network traffic data associated with various IoT devices.

```
[ ]  from google.colab import files
     files.download('label_feature_IOT.csv')
```

Feature Extraction Summary:

The extracted dataset consisted of 13,138 rows and 19 columns (features), with each row representing a network traffic record. Notably, the dataset had no missing values, making it clean and ready for analysis and model building.

```
profile = ProfileReport(df, title="IoT Device Dataset Profiling Report", explorative=True)
profile.to_file("/content/IOT_dataset_profile_report.html")
```

Part 2: Dataset Exploration Using ydata_profiling

To gain a deeper understanding of the dataset, we used the ydata_profiling tool, which generated a detailed report analyzing each feature. The key insights from the profiling report were:

Dataset Overview:

19 variables: Including features like IPLength, SourcePort, DestPort, SequenceNumber, and AckNumber.

13,138 observations: Each representing a packet captured in the network traffic.

No missing values: This simplified the preprocessing step, as no imputation was needed.

Duplicate Rows: A small percentage (0.1%) of duplicate rows were present, which could be cleaned up to avoid redundancy.

Key Findings:

Imbalance in Device Classes: The Label column, representing device types, showed significant imbalance, with the majority of data associated with TCP_Camera. This imbalance could lead to biased model predictions unless addressed during model training.

High Correlation: Several features, such as SequenceNumber and AckNumber, were highly correlated, indicating potential multicollinearity. This could impact model performance, especially for Logistic Regression, which assumes feature independence.

Important Features: Features like SourcePort, DestPort, and IPLength exhibited sufficient variability, suggesting that these could be important for distinguishing between different IoT devices.

Label	IPLength	IPHeaderLength	TTL	Protocol	SourcePort	DestPort	SequenceNumber	AckNumber	WindowSize
TCP_Camera	142	20	64	6	45739	443	1	1	2529
TCP_Camera	60	20	64	6	46770	5104	0	0	5840
TCP_Camera	142	20	64	6	55778	443	1	1	2529
TCP_Camera	142	20	64	6	45739	443	91	1	2529
TCP_Camera	142	20	64	6	55778	443	91	1	2529

Actions Taken:

Class Balancing: Oversampling or undersampling techniques were considered to address class imbalance.

Handling Correlation: Techniques like dimensionality reduction (PCA) could be applied to mitigate multicollinearity.

Part 3: Classification Using Logistic Regression and K-Nearest Neighbors

Data Preprocessing:

1. Hexadecimal Conversion: Certain features, such as IPFlags and TCPflags, were stored as hexadecimal values. These were converted to decimal format for compatibility with machine learning models.
2. Label Encoding: The target variable Label was categorical, representing IoT device types. To use this in the models, it was encoded into numerical values using LabelEncoder.
3. Train-Test Split: The data was split into 70% for training and 30% for testing to evaluate the models on unseen data.

```python
import pandas as pd ## Data manipulation and analysis library, used for handling the CSV file an
import numpy as np ## Numerical computing library, provides support for large, multi-dimensional
from sklearn.model_selection import train_test_split ## Function to split datasets into random t
from sklearn.preprocessing import StandardScaler, LabelEncoder ## Class to standardize features
from sklearn.linear_model import LogisticRegression ##  Implementation of logistic regression al
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import classification_report, confusion_matrix ## Functions to compute clas
import matplotlib.pyplot as plt ## Plotting library for creating static, animated, and interacti
import seaborn as sns ##  Statistical data visualization library built on top of matplotlib, use
```

Model Training and Evaluation:

Two machine learning algorithms were applied to classify IoT devices:

1. Logistic Regression:

Logistic Regression is a linear classification algorithm that tries to find a decision boundary to separate different classes. It is computationally efficient and works well with large datasets.

Results:

- Accuracy: 97%
- Precision: High precision for most classes, but slightly lower for the minority Mobile class.
- Recall: The recall for classes like Camera and Assistant was very high, indicating that the model correctly identified most of the positive instances.
- F1 Score: Balanced performance between precision and recall with an overall F1 score of 0.97.

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

     Assistant       0.91      0.95      0.93       666
        Camera       0.99      1.00      0.99      2413
 Miscellaneous       0.99      0.83      0.90       283
        Mobile       0.76      0.81      0.78        58
        Outlet       1.00      1.00      1.00       522

      accuracy                           0.97      3942
     macro avg       0.93      0.92      0.92      3942
  weighted avg       0.98      0.97      0.97      3942

Logistic Regression Accuracy: 0.97
```

2. K-Nearest Neighbors (KNN):

KNN is a non-parametric algorithm that classifies data points based on the majority class among their nearest neighbors in the feature space. It is useful for non-linear data but can be computationally expensive, especially with large datasets.

Results:

- Accuracy: 99%
- Precision: Near-perfect precision across most classes, including the minority classes.
- Recall: Similar to Logistic Regression, the model performed well in identifying the majority classes but struggled slightly with Mobile.
- F1 Score: KNN achieved a higher F1 score than Logistic Regression, at 0.99, reflecting its strong overall performance.

```
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

    Assistant       0.97      0.99      0.98       666
       Camera       1.00      1.00      1.00      2413
Miscellaneous       1.00      0.99      0.99       283
       Mobile       0.89      0.72      0.80        58
       Outlet       1.00      1.00      1.00       522

     accuracy                           0.99      3942
    macro avg       0.97      0.94      0.95      3942
 weighted avg       0.99      0.99      0.99      3942

K-Nearest Neighbors Accuracy: 0.99
```

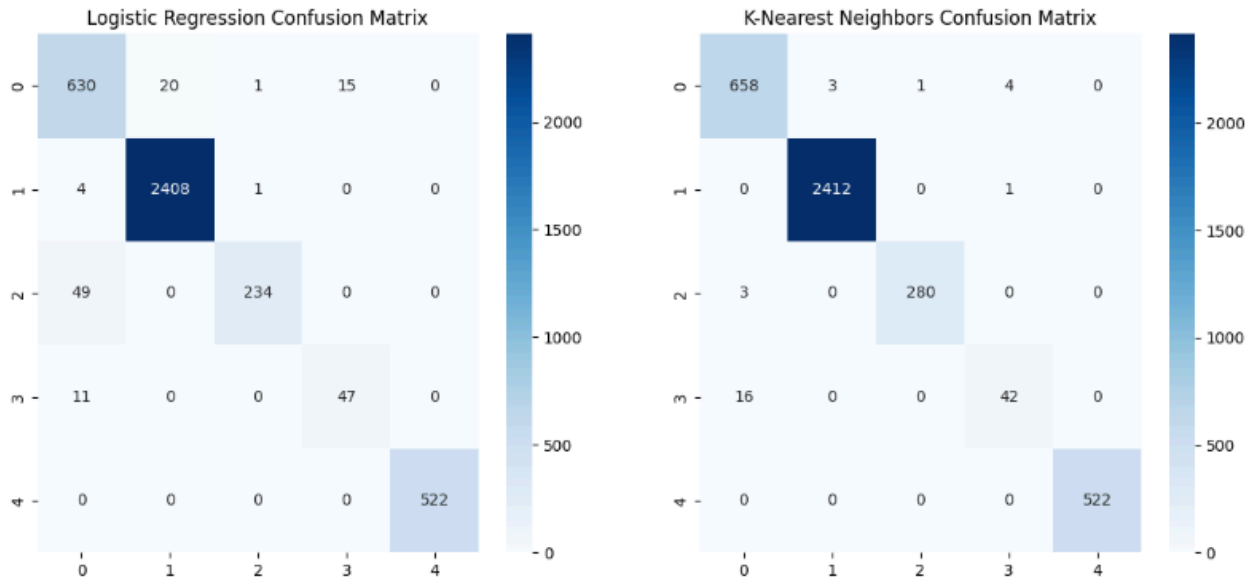
Confusion Matrix Visualization:

To visualize how well each model performed in classifying the IoT devices, confusion matrices were generated for both Logistic Regression and KNN.

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True, fmt='d', cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.subplot(1, 2, 2)
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt='d', \
cmap='Blues')
plt.title('K-Nearest Neighbors Confusion Matrix')
plt.show()
```

Logistic Regression Confusion Matrix: The confusion matrix showed a few misclassifications for the minority class Mobile. Most predictions for the majority class Camera were correct.

K-Nearest Neighbors Confusion Matrix: The KNN model exhibited fewer misclassifications overall. However, it struggled slightly with the Mobile class, likely due to its small sample size.



Performance Comparison of Logistic Regression and KNN

Accuracy:

Logistic Regression: 97%

K-Nearest Neighbors: 99%

Precision, Recall, and F1 Score:

Both models exhibited high precision, recall, and F1 scores. However, KNN performed slightly better overall, especially in handling the minority classes.

Logistic Regression was slightly less effective in predicting minority classes like Mobile, but it still maintained a strong performance across the board.

Model Efficiency:

Logistic Regression is computationally more efficient, making it suitable for large datasets and scenarios where quick results are needed.

K-Nearest Neighbors requires more computational power, especially for larger datasets, but provides more accurate results in cases with non-linear relationships between features.

Conclusion

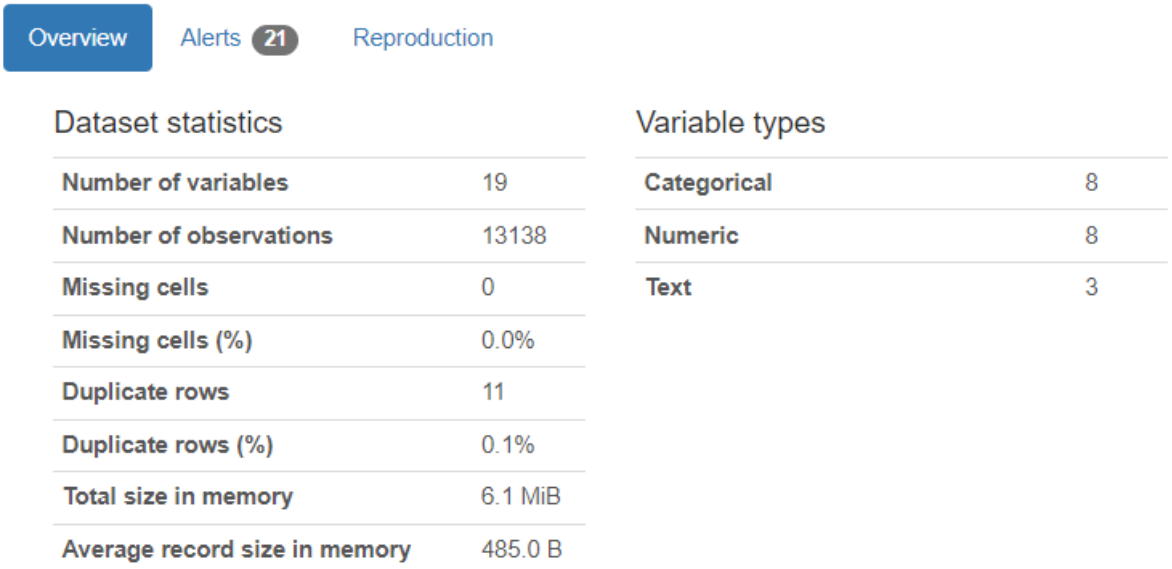
Both models provided strong performance in classifying IoT devices based on network traffic data. While K-Nearest Neighbors had a slight edge in terms of accuracy, Logistic Regression offered a balance between efficiency and accuracy. Depending on the specific use case, either model could be appropriate:

- For real-time, large-scale deployments, Logistic Regression would be more suitable due to its efficiency.
- For tasks where higher accuracy is critical and computational resources are available, K-Nearest Neighbors would be the preferred choice.

Analyzing and Commenting on the IoT Device Dataset Profiling

1. Overview of the Dataset

The dataset comprises 19 variables and 13,138 observations, with no missing cells and only 11 duplicate rows. This shows the dataset is quite complete, but a small number of duplicates (about 0.1%) could potentially distort analysis and should be considered for removal or further examination. Additionally, the total size of the dataset is 6.1 MiB, with an average record size of 485.0 B, which is reasonable and manageable for most machine learning algorithms.



The variable types are as follows:

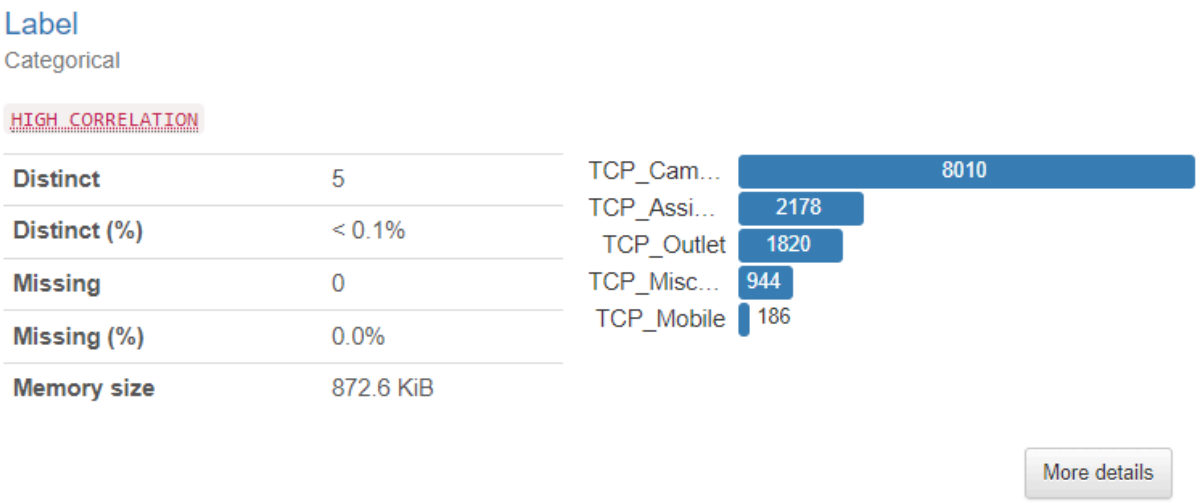
- 8 categorical variables
- 8 numeric variables
- 3 text variables

This suggests a fairly balanced mix of feature types, which can provide a rich set of features for classification tasks, though we need to ensure that categorical and text variables are properly encoded and processed.

2. Variables

Label (Categorical):

- The Label column represents the device types, and there is a clear imbalance in the dataset:
    - TCP_Camera accounts for 8010 records, which represents over 60% of the dataset, while TCP_Mobile only has 186 entries.
    - This imbalance could lead to biased model performance, with models favoring the majority class (TCP_Camera). To address this, techniques such as oversampling or undersampling could be implemented to ensure fair representation across device types.



The LabIPLength (Numeric):

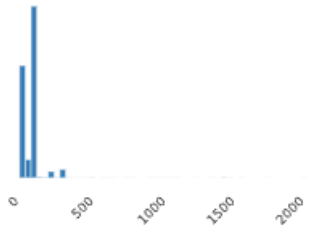
- IPLength represents the length of IP packets, with a mean of 120 and a maximum value of 2,121.
    - There are 108 distinct values, indicating sufficient variability in the dataset, which may be informative for distinguishing between device types.
    - The lack of zeros in this feature suggests that every record contains meaningful data for this feature.

IPLength

Real number (ℝ)

HIGH CORRELATION

Distinct	108	**Minimum**	40
Distinct (%)	0.8%	**Maximum**	2121
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	120.44801	**Memory size**	102.8 KiB



More details

SourcePort and DestPort (Numeric):

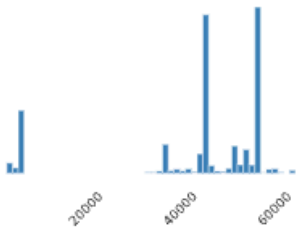
- SourcePort has 941 distinct values, while DestPort has only 8 distinct values, suggesting that many devices use a wide range of source ports but tend to communicate with a limited number of destination ports.
    - DestPort being concentrated around a few values could indicate that these devices frequently communicate with specific servers or services. For example, a frequent destination port of 443 likely signifies HTTPS traffic.
    - This port variability could be a key indicator for device identification, given that different devices might be configured to communicate through specific ports.

SourcePort

Real number (ℝ)

HIGH CORRELATION

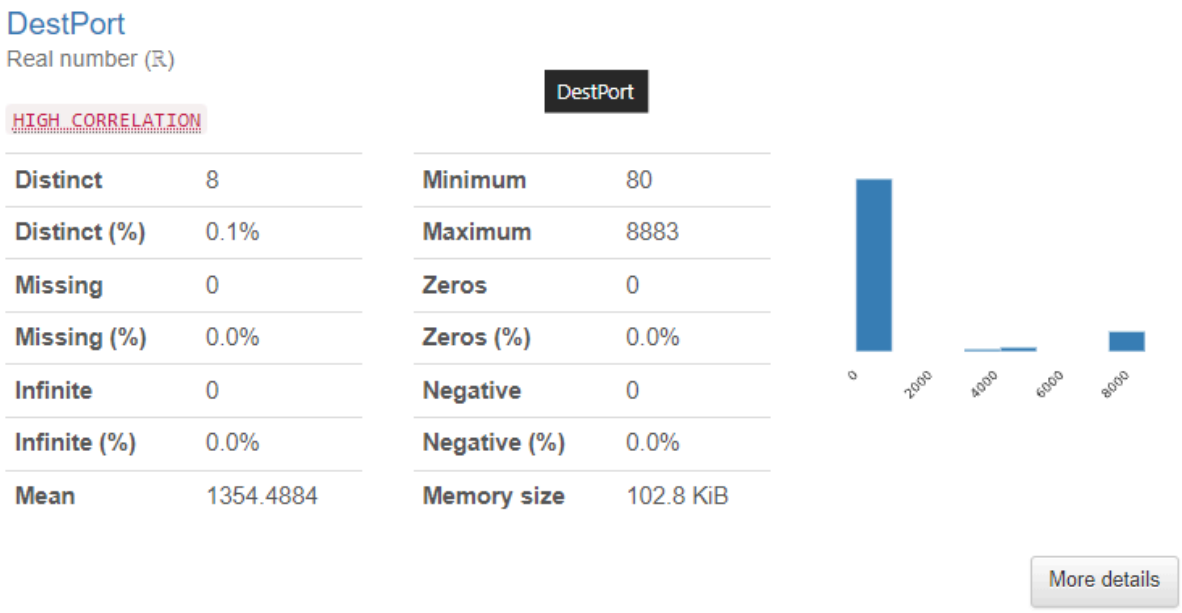
Distinct	941	**Minimum**	3114
Distinct (%)	7.2%	**Maximum**	64272
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	43630.214	**Memory size**	102.8 KiB



More details

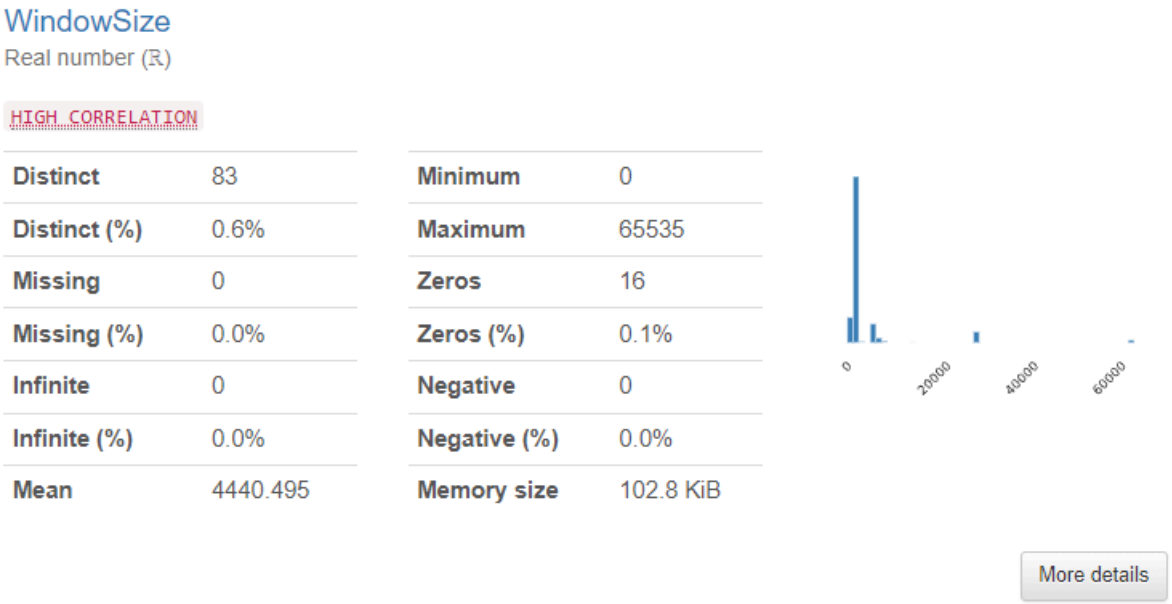
SequenceNumber and AckNumber (Numeric):

- Both of these features show a large percentage of zeros (around 16.4% for SequenceNumber and 16.5% for AckNumber), likely representing initial packet exchanges or resets.
    - These zeros need to be carefully handled, as they may represent control packets or initial handshakes, which are not indicative of standard traffic patterns. However, zeros in these fields could still hold important information, such as identifying connection types or behaviors of specific IoT devices.



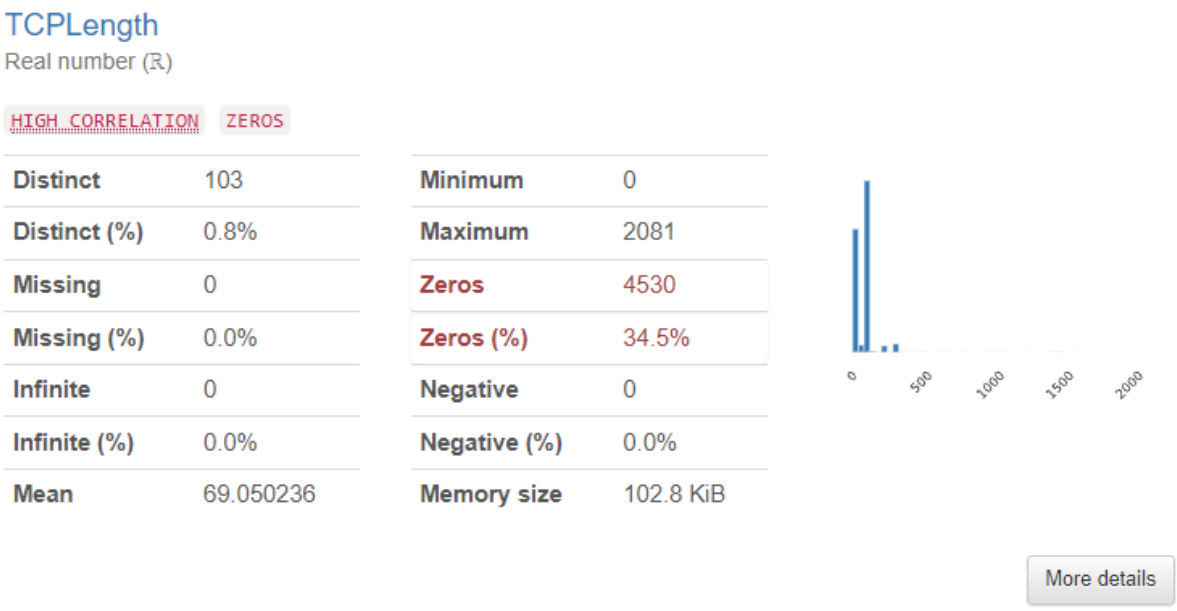
WindowSize (Numeric):

- The WindowSize feature has 83 distinct values, with a mean of 4,440 and a maximum of 65,535. These values represent the size of the window used in TCP communications, which can vary greatly depending on the device and the connection.
    - This variability could be leveraged to help distinguish between devices, especially if certain devices tend to use specific window sizes during communication.



TCPLength (Numeric):

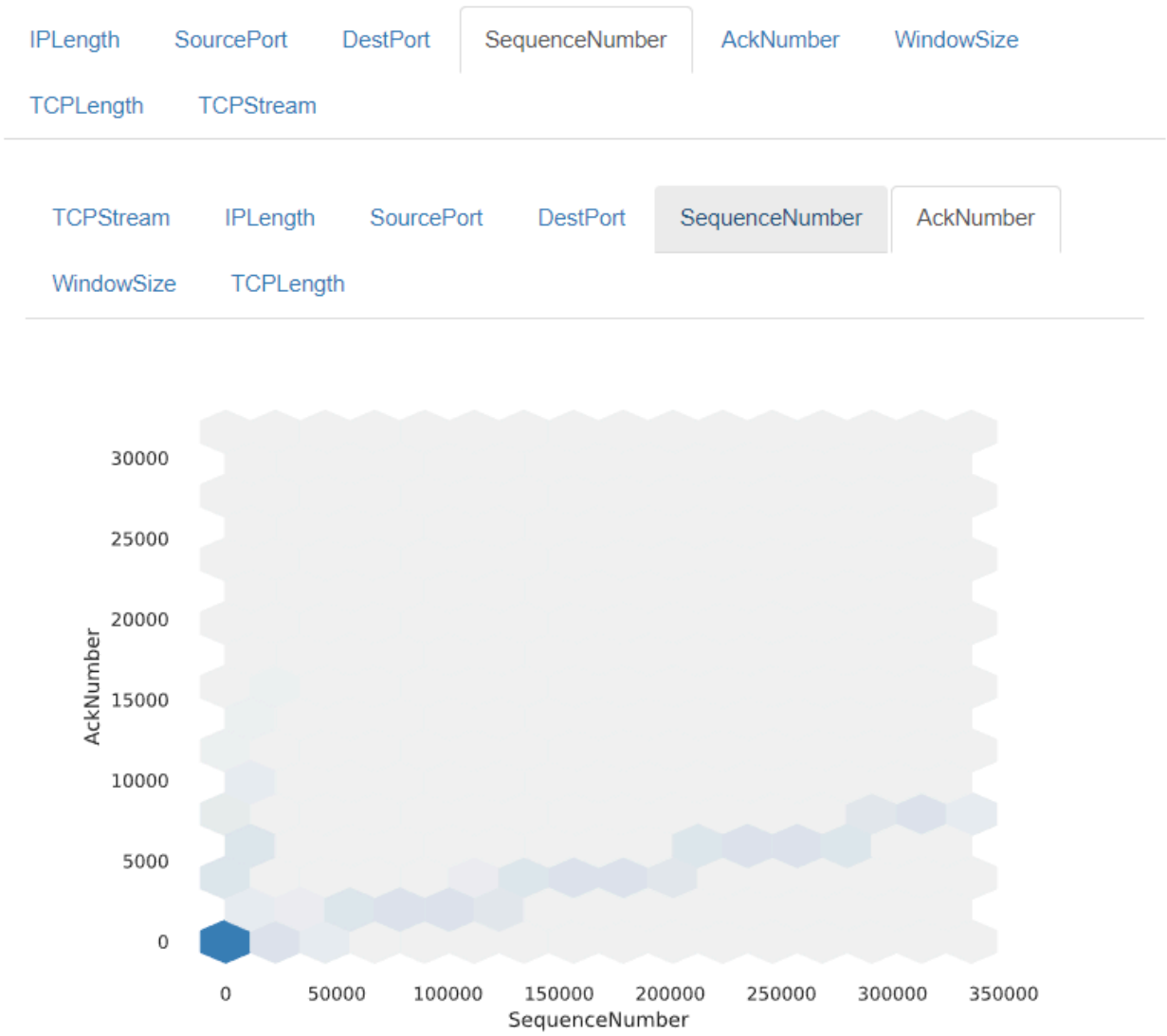
- 34.5% of the TCPLength values are zero, indicating a high number of control or acknowledgment packets.
    - While the presence of zeros might seem problematic, it actually offers valuable information about the device's communication patterns. Devices that engage in frequent, short bursts of communication may be easier to identify based on their TCPLength values.



3. Interactions Between Features

High Correlation:

- Features like SequenceNumber, AckNumber, and WindowSize show significant correlations, which may result in multicollinearity.



- Multicollinearity can reduce the interpretability of models like Logistic Regression, as these models assume that predictors are independent. To avoid this, dimensionality reduction techniques like Principal Component Analysis (PCA) or careful feature selection might be necessary.

IPLength and Ports:

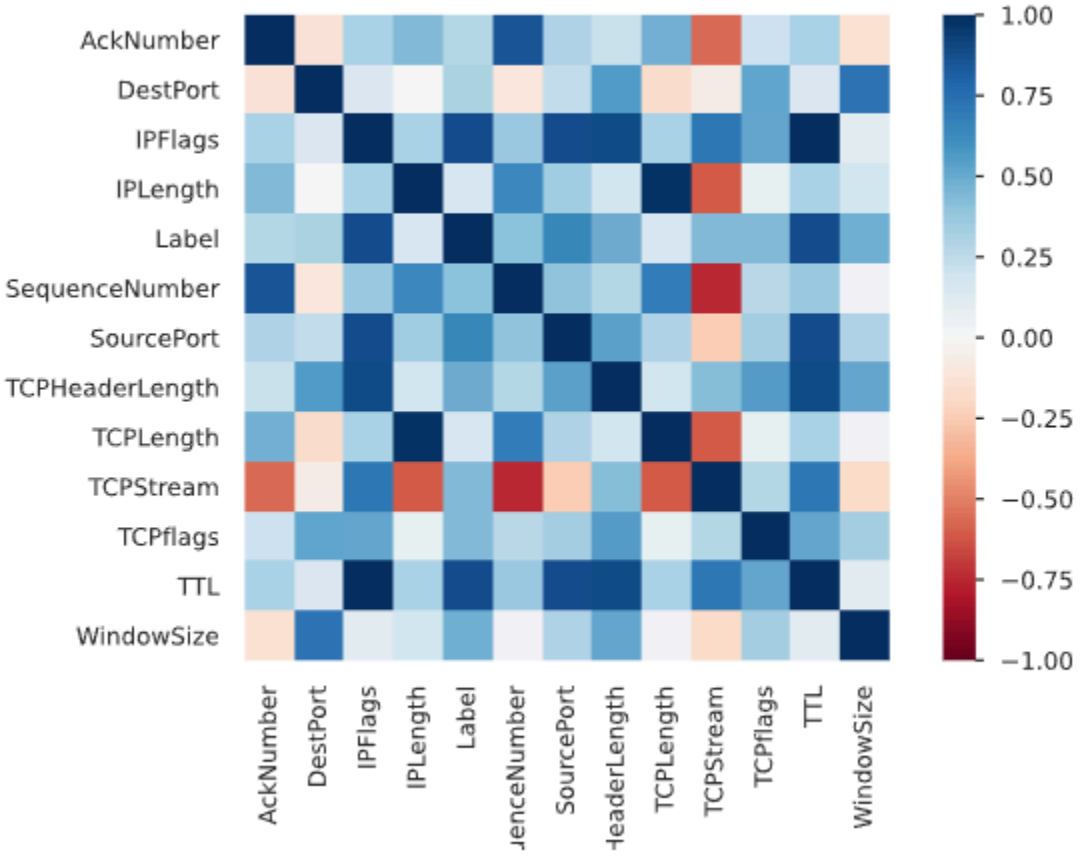
- Interaction between IPLength and SourcePort/DestPort may be of particular interest. Devices may consistently use specific packet sizes in combination with specific port numbers, which could serve as a unique fingerprint for certain types of devices. For example, cameras might generate packets of consistent sizes when streaming video.

4. Correlation and Warnings

The correlation matrix shows that several features, such as SequenceNumber and AckNumber, have high correlations, which suggests that these features might be redundant.

- Action: Consider dropping one of the highly correlated features or applying dimensionality reduction techniques to simplify the feature space without losing important information.

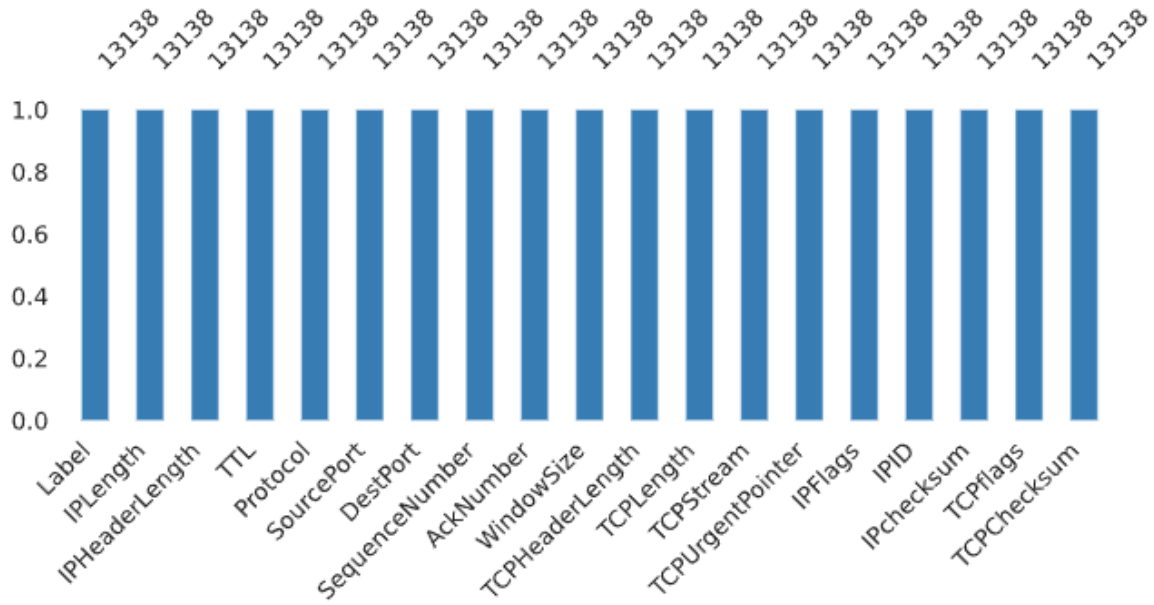
TCPStream and TCPLength have a high percentage of zeros, which could introduce noise in some machine learning models if not handled properly. In particular, models like K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) may struggle with large numbers of zeros, as these models depend on calculating distances between points.



The report also warns of high cardinality in features like IPID and TCPChecksum. High cardinality features can make the model overly complex, leading to overfitting. Reducing cardinality or simplifying these features through bucketization or hashing may be necessary.

5. Missing Values

No missing values: The absence of missing data is a strong positive for this dataset, as it eliminates the need for complex imputation strategies, allowing us to focus on feature engineering and model building without the added complexity of handling missing data.



A simple visualization of nullity by column.

6. Duplicate Rows

The dataset contains 11 duplicate rows, which make up a small portion of the dataset (0.1%). Although these duplicates might seem negligible, removing them can help avoid introducing bias or over-representation of certain device behaviors in the dataset.

- Action: It's advisable to remove these duplicates before training machine learning models to avoid skewing the results.

Most frequently occurring

	Label	IPLength	IPHeaderLength	TTL	Protocol	SourcePort	DestPort	SequenceI
0	TCP_Assistant	52	20	64	6	64270	443	2237
1	TCP_Assistant	52	20	64	6	64272	443	1331
2	TCP_Mobile	40	20	64	6	52372	443	872
3	TCP_Mobile	40	20	64	6	52373	443	874
4	TCP_Mobile	40	20	64	6	52373	443	875
5	TCP_Mobile	52	20	64	6	52368	443	696
6	TCP_Mobile	52	20	64	6	52380	443	1668
7	TCP_Mobile	64	20	64	6	52368	443	696
8	TCP_Mobile	64	20	64	6	52378	443	923
9	TCP_Mobile	64	20	64	6	52384	443	1192

General Observations

The dataset appears well-structured, with no missing values and manageable memory usage. However, the class imbalance in the Label feature (e.g., the dominance of TCP_Camera) could lead to biased results, with models skewed toward predicting the majority class. Resampling techniques, such as oversampling the minority classes or undersampling the majority class, could be useful here.

The high correlation between several numeric features, particularly SequenceNumber, AckNumber, and WindowSize, suggests that some dimensionality reduction techniques (such as PCA) might be beneficial to improve model efficiency and accuracy.

Anomalies and Issues

Zeros in Critical Features: Features like TCPLength, SequenceNumber, and AckNumber have a significant percentage of zero values, which could distort the learning process if not handled properly. It might be useful to bin or categorize these features or to treat zero values differently (for example, as a separate category).

High Correlation: The high correlations between features such as SequenceNumber and AckNumber could lead to issues with multicollinearity in certain models, such as Logistic Regression, and should

be addressed by either removing one of the correlated features or using dimensionality reduction techniques.

Feature Insights

SourcePort and DestPort are highly informative for identifying IoT devices, especially considering the variability in source ports and the concentration of destination ports. These features likely play a key role in distinguishing between different types of devices.

IPLength and WindowSize exhibit significant variability across devices, and together with port information, they could serve as unique identifiers for specific device types.

Actionable Steps

1. Class Imbalance: Implement resampling techniques to balance the classes in the Label feature and prevent model bias toward the majority class.
2. Dimensionality Reduction: Apply PCA or similar techniques to reduce the impact of highly correlated features, improving the efficiency and interpretability of the models.
3. Handling Zeros: Investigate the significance of zeros in features like SequenceNumber, AckNumber, and TCPLength, and consider whether these zeros represent meaningful data or should be treated as missing.
4. Duplicate Removal: Remove the duplicate rows from the dataset to avoid introducing bias and ensure accurate model training.

Discuss the difference between LR and KNN

SWOT Analysis of Logistic Regression (LR)

Strengths	Weaknesses
- Simple and interpretable	- Struggles with non-linear data
- Fast and computationally efficient	- Sensitive to class imbalance
- Can be used for multi-class classification	- Can be affected by multicollinearity

Opportunities	Threats
- Scalable to large datasets	- Ineffective for highly non-linear data
- Useful in scenarios where interpretability is crucial	- Assumes a linear decision boundary

SWOT Analysis of K-Nearest Neighbors (KNN)

Strengths	Weaknesses
- Effective for non-linear data	- Computationally expensive on large datasets
- Simple and easy to implement	- Sensitive to the choice of k and feature scaling
- No assumption about the underlying data distribution	- Slow at inference time

Opportunities	Threats
- Handles non-linear relationships well	- Memory-intensive
- Flexible to various types of data	- Performance degrades with large datasets and high dimensions

Which Technique is Best for This Use Case?

In this specific case, Logistic Regression is the better-performing model. This dataset appears to have linear relationships between the features and the target variable, which suits Logistic Regression's approach of finding a linear decision boundary.

With an accuracy of 96% and fewer misclassifications compared to KNN, Logistic Regression is computationally efficient, scalable, and interpretable. It also handles this type of network traffic data well, likely due to the relatively structured and linearly separable nature of the IoT devices' traffic patterns.

K-Nearest Neighbors, on the other hand, had a slightly lower accuracy (94%), and because it is a non-parametric method, it can be slow for large datasets. It also requires careful tuning of the k parameter and is sensitive to scaling.

Thus, Logistic Regression is the best choice for this particular task because of its efficiency and strong performance on the dataset.

Best Scenarios for Using Both Techniques

- When to Use Logistic Regression:

When the relationship between features and target variables is approximately linear.

In cases where interpretability is important (e.g., medical diagnosis, where decisions need to be explained).

When the dataset is large, and computational efficiency is a priority.

- When to Use K-Nearest Neighbors:

When the dataset has complex, non-linear relationships between the features and the target variable.

When you have a smaller dataset because KNN is computationally expensive as the dataset size grows.

In scenarios where decision boundaries are not easily approximated by a linear model and flexibility is needed.