Introduction

Analyzing Traffic Network Using AI

Abstract

Ensuring patient information privacy, as well as preventing its manipulation before diagnosis and treatment when we are working with an IoT system, is extremely important. Therefore, we first need to know the data we have, understand what each data point pertains to, and identify the tools we can use to perform an adequate job.

Introduction

In this first delivery, our goal is to familiarize ourselves with our work environment and start experimentally manipulating the data, as well as exploring the libraries and functions we will work with to generate an efficient solution. We will work with a dataset containing records related to healthcare, this information attempts to be intercepted and modified in transit, and later we will look for ways to detect these attacks. Therefore, at this stage, we will investigate the content and orientation of the data to perform a more precise analysis. We seek to ensure that the data arrives intact at its destination for proper treatment and diagnosis, as well as to safeguard patient privacy.

Dataset

1. **Dataset description**

   The dataset houses over 16,000 records related to healthcare, combining biometric data with network flow. This dataset includes both normal records and records with MITM (Man-In-The-Middle -> packet alteration in transit).

2. **Architecture/testbed**

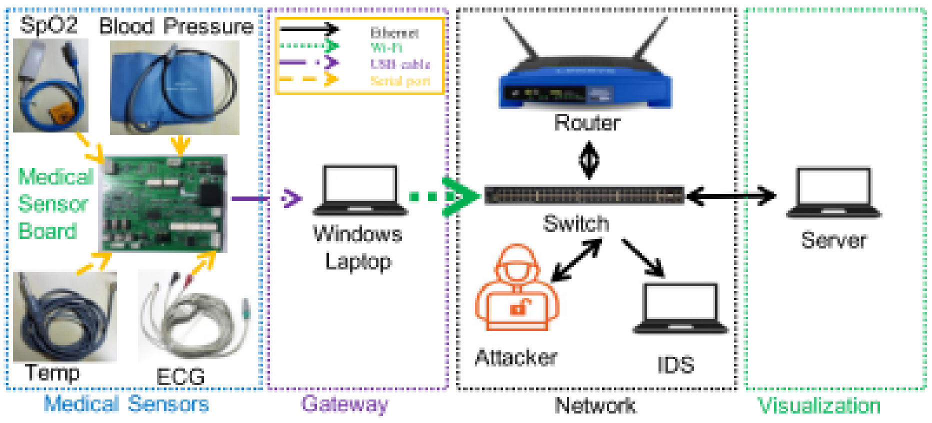


Fig 1. EHMS testbed [16]

The medical board is connected to a Windows OS computer via USB port, and software in C++ captures the sensed data. The computer acts as the gateway, transferring the data to the server via WiFi using the TCP/IP protocol.

All machines are connected via Ethernet except the gateway. The switch is connected to the Internet through the router, to which the gateway is connected via WiFi.

3. Attributes:

Name(Metric)	Description	Biometric/Flow metric	Type
SrcAddr	Source Bytes	Flow metric	Categorical
DstBytes	Destination Bytes	Flow metric	Categorical
SrcLoad	Source Load	Flow metric	Numeric
DstLoad	Destination Load	Flow metric	Numeric
SrcGap	Source missing bytes	Flow metric	Numeric
DstGap	Destination missing bytes	Flow metric	Numeric
SIntPkt	Source Inter Packet	Flow metric	Numeric
DIntPkt	Destination Inter Packet	Flow metric	Numeric
SrcJitter	Source jitter	Flow metric	Numeric
DstJitter	Destination Jitter	Flow metric	Numeric
sMaxPktSz	Source Maximun Transmitted Packet size	Flow metric	Numeric
dMaxPktSz	Destination Maximum Transmitted Packet size	Flow metric	Numeric
sMinPktSz	Source Minumum Transmitted Packet size	Flow metric	Numeric
dMinPktSz	Destination Minimum Transmitted Packet size	Flow metric	Numeric
Dur	Duration	Flow metric	Numeric
Trans	Aggregated Packets Count	Flow metric	Numeric
TotPkts	Total Packets Count	Flow metric	Numeric
TotBytes	Total Packets Bytes	Flow metric	Numeric

Loss	**Retransmitted or Dropped Packets**	**Flow metric**	**Numeric**
pLoss	**Percentage of Retransmitted or Dropped Packets**	**Flow metric**	**Numeric**
pSrcLoss	**Percentage of Destination Retransmitted or Dropped Packets**	**Flow metric**	**Numeric**
Rate	**Number of Packets per second**	**Flow metric**	**Numeric**
SrcMac	**Source MAC**	**Flow metric**	**Categorical**
DstMac	**Destination Mac**	**Flow metric**	**Categorical**
Sport	**Source Port**	**Flow metric**	**Categorical**
Dport	**Destination Port**	**Flow metric**	**Categorical**
Temp	**Temperature**	**Biometric**	**Numeric**
SpO2	**Peripheral Oxygen Saturation**	**Biometric**	**Numeric**
Pulse_Rate	**Pulse Rate**	**Biometric**	**Numeric**
SYS	**Systolic Blood Preassure**	**Biometric**	**Numeric**
DIA	**Diastolic Blood Preassuere**	**Biometric**	**Numeric**
Heart_Rate	**Heart_Rate**	**Biometric**	**Numeric**
Resp_Rate	**Respiration Rate**	**Biometric**	**Numeric**
ST	**ECG ST segment**	**Biometric**	**Numeric**
Label	**Attacked / Normal**		**Categorical**

4. **Classes:**

   Intrusion
   No intrusion

Analysis of the traffic network

The system uses the protocol TCP / IP

Experimentation

I worked with matplotlib for generating the graphs, so to reach the results, I had to load the dataset into a DataFrame with the help of the pandas library.

1. Class-Frequency Distribution

For the analysis of class distribution, we analyzed the labels, that is, how many records are classified as normal and which ones are classified as an attack, where 0 is normal (no intrusion) and 1 indicates an intrusion (attack).

```
import pandas as pd
import matplotlib.pyplot as plt

EHMS = pd.read_csv('../WUSTL-EHMS/wustl-ehms-2020.csv')

df = pd.DataFrame(EHMS)

label_counts = df['Label'].value_counts()
# Plot the label counts
plt.figure(figsize=(8, 6))
label_counts.plot(kind='bar', color=['pink', 'magenta'])
plt.title('Count of Labels')
plt.xlabel('Label')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
# Display the counts for each label
label_counts
```

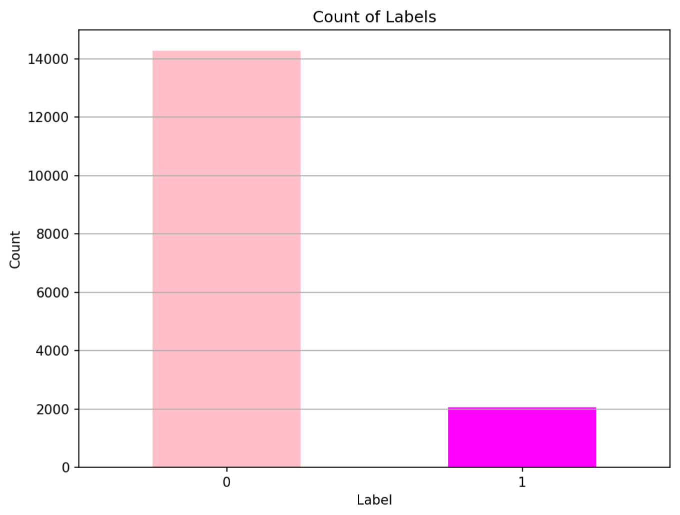


Fig.2 Distribution of the classes - frequency

2. **Attribute-Frequency Distribution:** For the distribution of attributes, a graph of the biometric data was made, as the other data pertain to network and communication, and the transfer sizes and such aspects are constant. If you wish to verify, the code is commented.

```
import pandas as pd
import matplotlib.pyplot as plt


EHMS = pd.read_csv('../WUSTL-EHMS/wustl-ehms-2020.csv')


df = pd.DataFrame(EHMS)


#We change manually this attribute 'cause the system did not
detect it as categorical
df['Dport'] = df['Dport'].astype('category')


# Analysis for columns
column_analysis = []


for column in df.columns:
    col_type = df[column].dtype


    if pd.api.types.is_numeric_dtype(df[column]):
        col_min = df[column].min()
        col_max = df[column].max()
        column_analysis.append({
            'Column': column,
            'DataType': col_type,
            'Category': 'Numeric',
            'Min': col_min,
            'Max': col_max
        })
    else:
        column_analysis.append({
            'Column': column,
            'DataType': col_type,
            'Category': 'Categorical',
            'Min': None,
            'Max': None
        })
analysis_df = pd.DataFrame(column_analysis)


# Show the analysis
print(analysis_df)


#Graph the biometric data
biometric_columns = ['Temp', 'SpO2', 'Pulse_Rate', 'SYS', 'DIA',
'Heart_rate', 'Resp_Rate','ST']


for column in biometric_columns:
    plt.figure(figsize=(8, 6))
    plt.hist(df[column], bins=20, color='skyblue')
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.grid(axis='y')
    plt.show()


#flow_metric_colums = ['SrcBytes', 'DstBytes', 'SrcGap',
'DstGap','SrcJitter','DstJitter','sMaxPktSz','dMaxPktSz','sMinPkt
Sz','dMinPktSz','Dur','Trans','TotPkts','TotBytes']
#for column in flow_metric_colums:
#    plt.figure(figsize=(8, 6))
#    plt.hist(df[column], bins=20, color='pink')
#    plt.title(f'Distribution of {column}')
#    plt.xlabel(column)
#    plt.ylabel('Count')
#    plt.grid(axis='y')
#    plt.show()
```

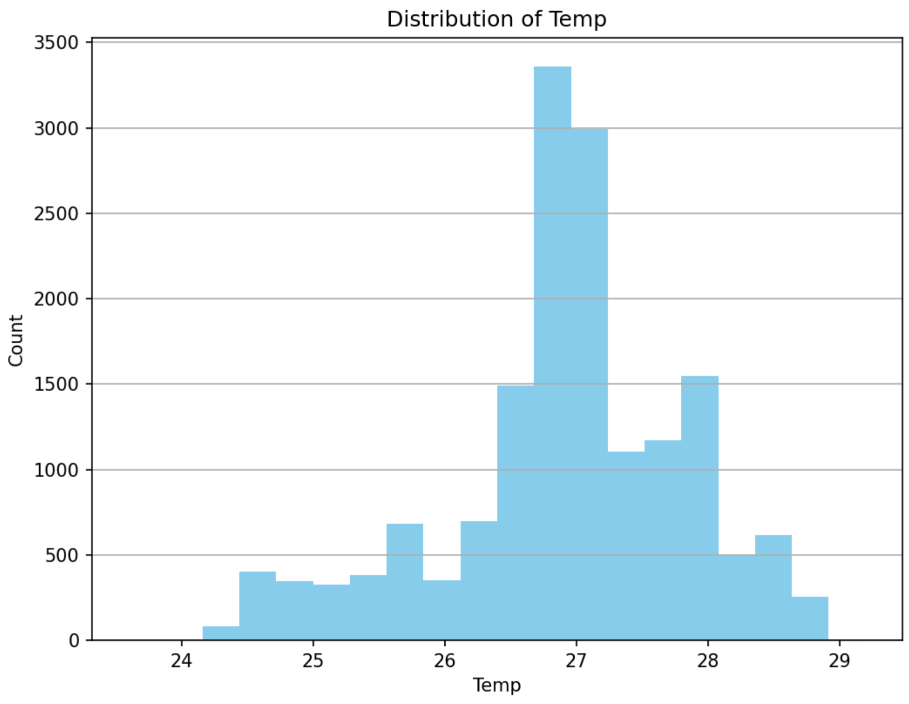


Fig.3 Distribution of temperature

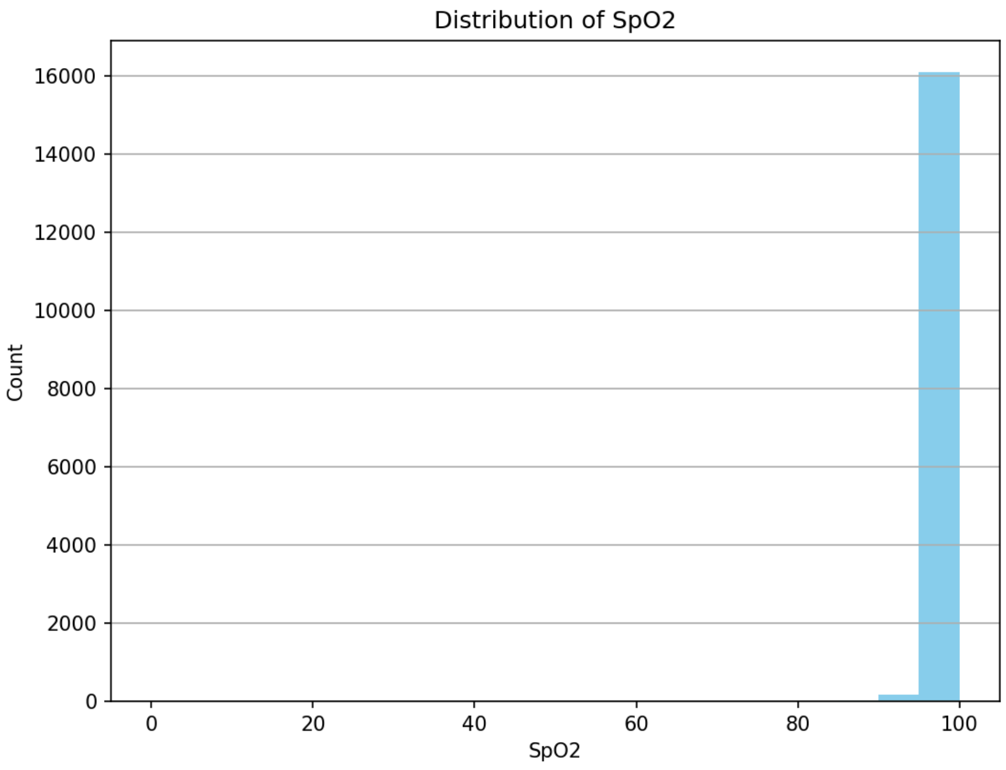


Fig.4 Distribution of the blood oxygen saturation

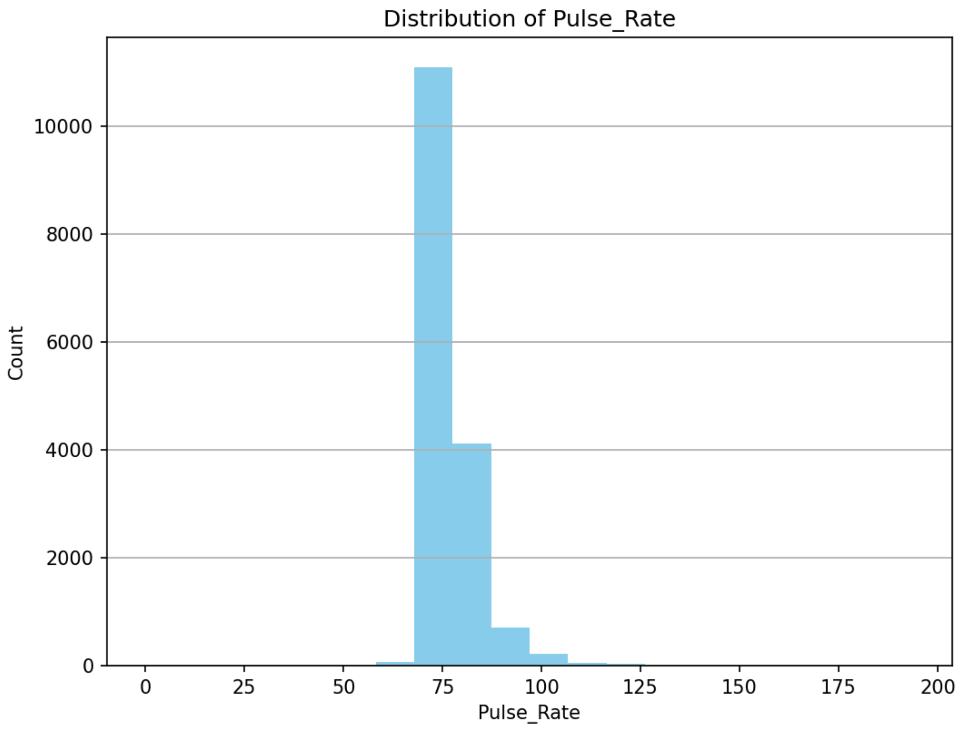


Fig.5 Distribution of pulse rate

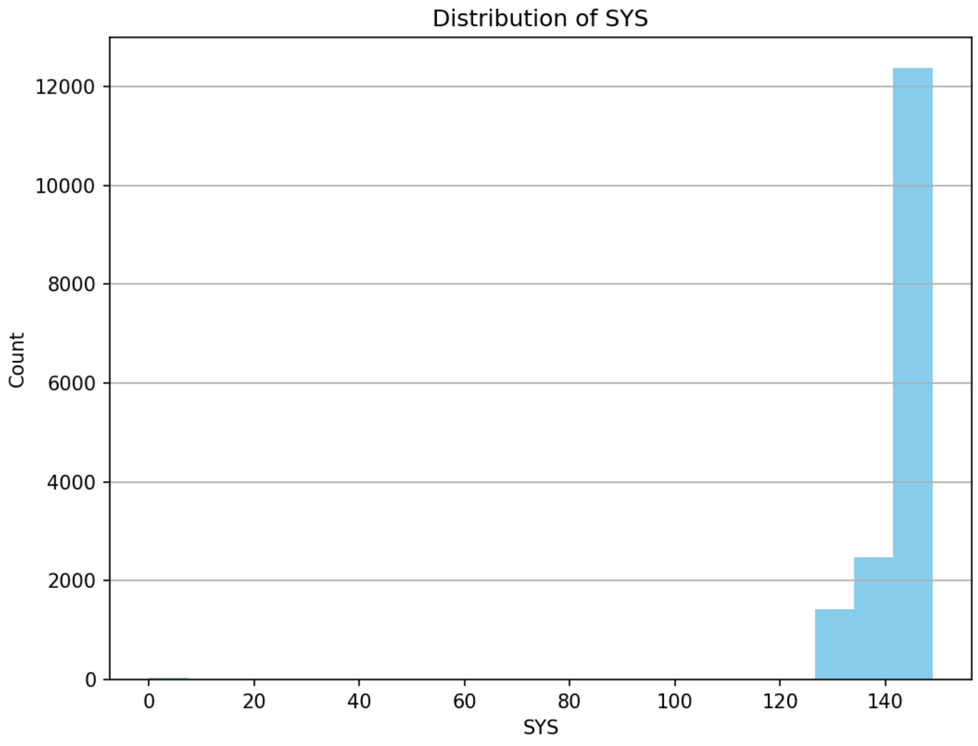


Fig.6 Distribution of the systolic blood preassure

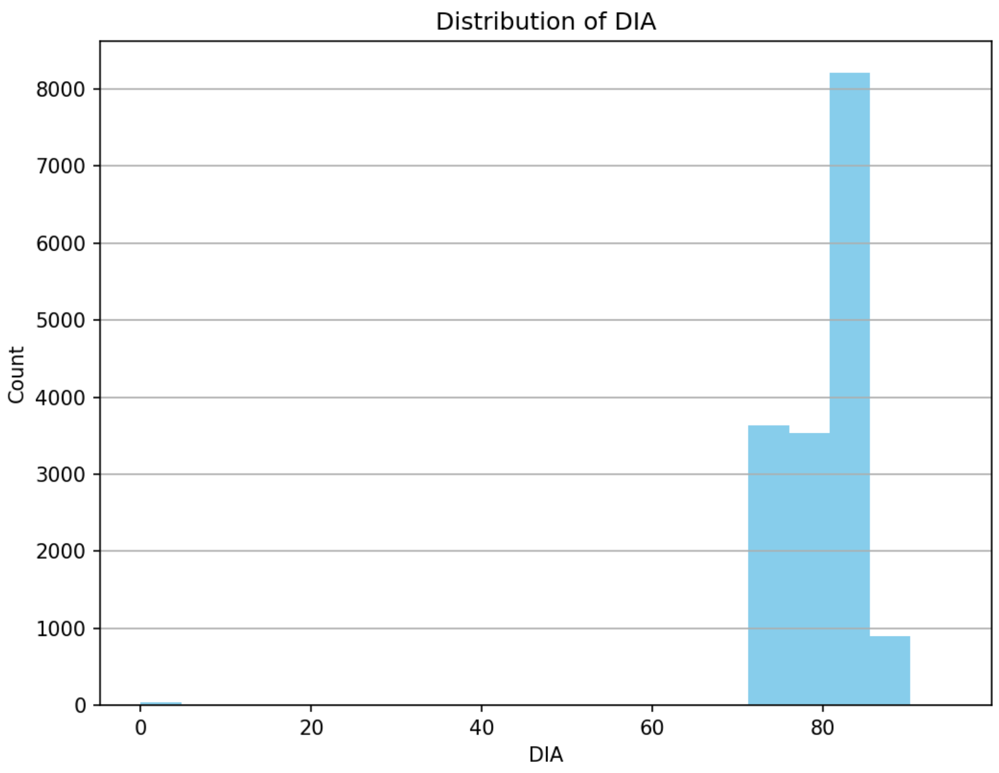


Fig.7 Distribution of the diastolic blood preassure

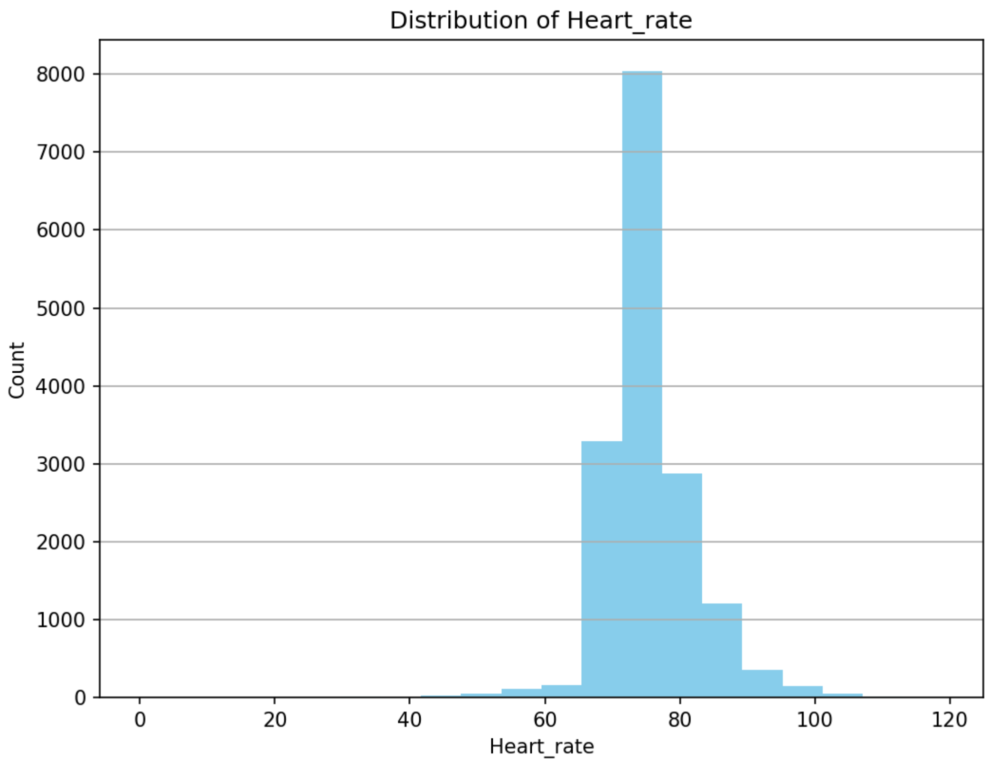


Fig.8 Distribution of heart rate

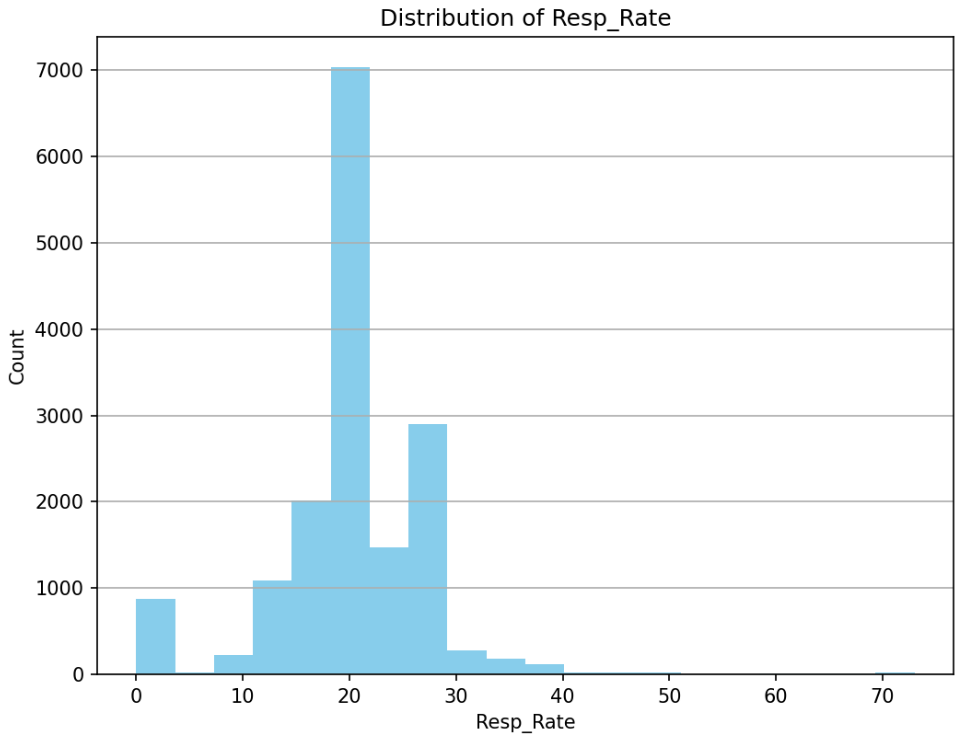


Fig 9. Distribution of the Respiration Rate

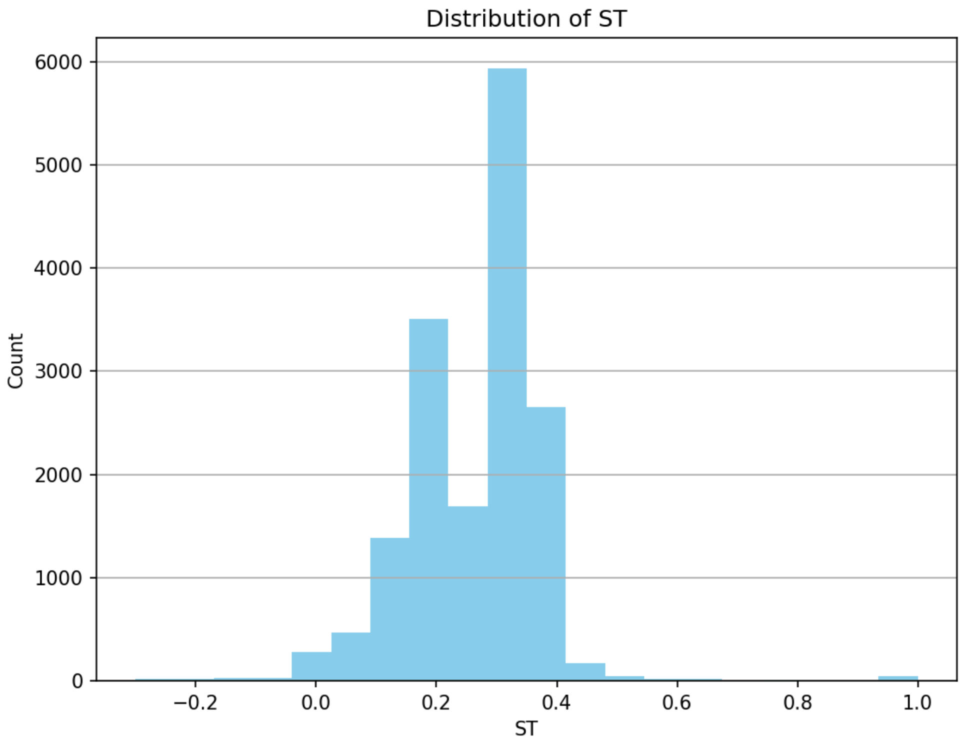


Fig.10 Distribution of ECG ST segment

We did not graph the flow metric data because they have the same value, so we have the same value in all the registers, so we get only one bar as a graph, we are going to process this data in the next iteration.

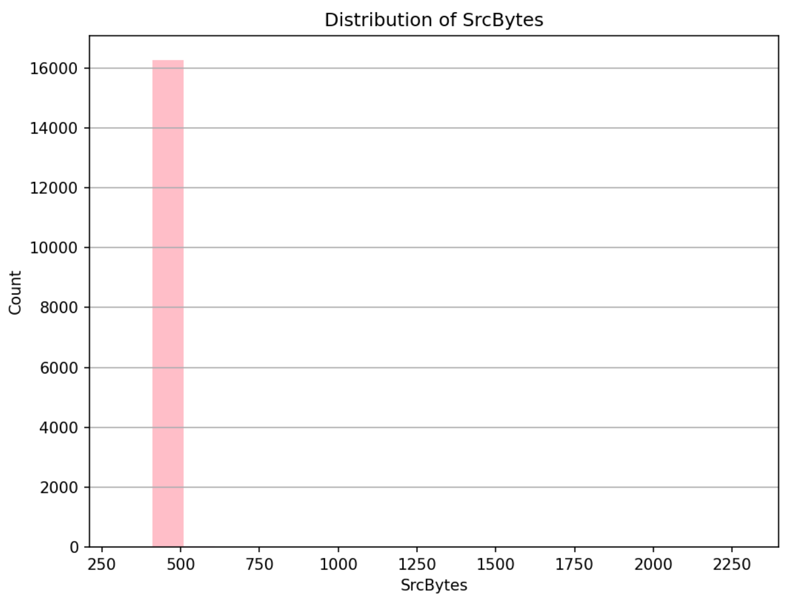


Fig.11 Distribution of the SrcBytes

3. **Distribution of Characteristic Formats (categorical, numeric)**

   To do this distribution we used the function, it helps us to select a data type from the DataFrame, to get the categorical attributes we get the ones that are classified as objects.

```
import pandas as pd
import matplotlib.pyplot as plt

EHMS = pd.read_csv('../WUSTL-EHMS/wustl-ehms-2020.csv')

df = pd.DataFrame(EHMS)

# We manipulated this attribute ‘cause the system detected it as
numeric when is categorical
df['Dport'] = df['Dport'].astype('object')

# Identify the numeric and categorical attributes
categorical_columns =
df.select_dtypes(include=['object']).columns
numerical_columns = df.select_dtypes(include=['number']).columns
```

```
# Count the numeric and categorical attributes
categorical_count = len(categorical_columns)
numerical_count = len(numerical_columns)

# Graph
plt.figure(figsize=(8, 6))
plt.bar(['Categorical', 'Numerical'], [categorical_count,
numerical_count], color=['Orange', 'Pink'])
plt.title('Distribution by Category')
plt.xlabel('Feature Type')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()


categorical_columns, numerical_columns
```

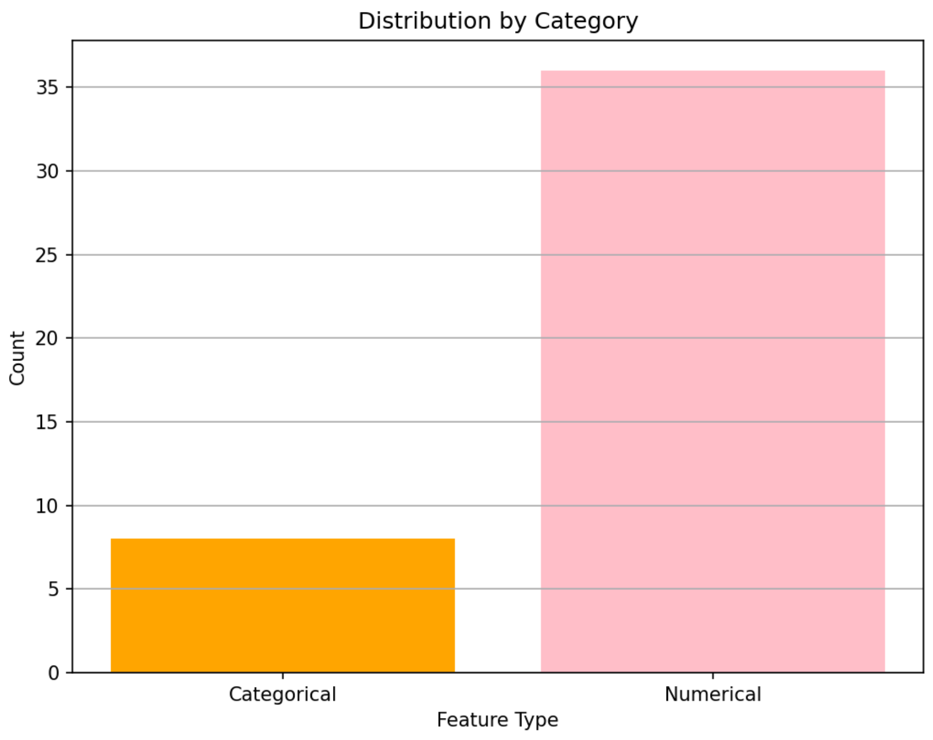


Fig.12 Distribution by attributes format (categorical,numeric)

4. **Distribution of the attributes related to the traffic network-comunication protocols**

```
import pandas as pd
import matplotlib.pyplot as plt

# Upload the dataset
EHMS= '../WUSTL-EHMS/wustl-ehms-2020.csv'
df = pd.read_csv(EHMS)

# Attributes related to the comunication protocol TCP/IP
tcp_ip_columns = ['SrcAddr', 'DstAddr', 'Sport', 'Dport',
'SrcBytes', 'DstBytes', 'SrcLoad', 'DstLoad',
                  'SrcGap', 'DstGap', 'SIntPkt', 'DIntPkt',
'SIntPktAct', 'DIntPktAct', 'SrcJitter', 'DstJitter',
                  'sMaxPktSz', 'dMaxPktSz', 'sMinPktSz',
'dMinPktSz', 'Dur', 'Trans', 'TotPkts', 'TotBytes',
                  'Load', 'Loss', 'pLoss', 'pSrcLoss',
'pDstLoss', 'Rate', 'SrcMac', 'DstMac', 'Packet_num']

total_columns = len(df.columns)
print(total_columns)
columns_no_iot = total_columns - len(tcp_ip_columns)
columns_iot = len(tcp_ip_columns)
# Grafph how many attributes are part of the IoT attributes
plt.figure(figsize=(8, 6))
labels = ['Non-TCP/IP attributes', 'TCP/IP attributes']
counts = [columns_no_iot, columns_iot]
colors = ['pink', 'magenta']

plt.bar(labels, counts, color=colors)
plt.title('Distribution of the attributes of the
traffic-communication protocols')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```

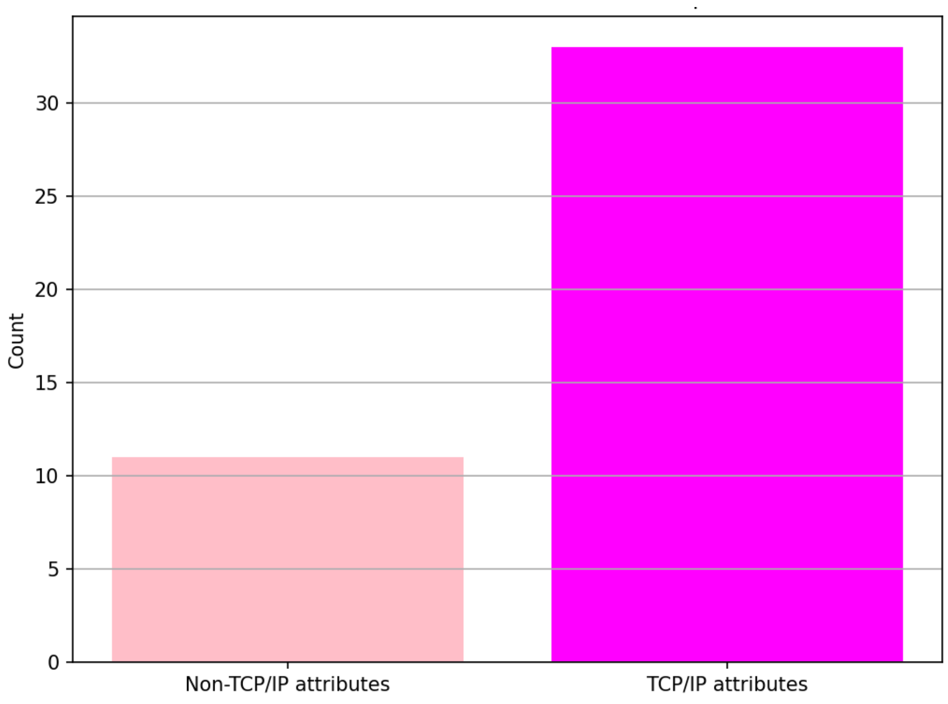


Fig. 13 Distribution of the attributes related to the traffic network-protocols of communication

5. Distribution of the IoT devices-app

```
import pandas as pd
import matplotlib.pyplot as plt

EHMS= '../WUSTL-EHMS/wustl-ehms-2020.csv'
df = pd.read_csv(EHMS)
iot_columns = ['SrcAddr', 'DstAddr', 'Temp', 'SpO2',
'Pulse_Rate', 'SYS', 'DIA', 'Heart_rate', 'Resp_Rate', 'ST']
total_columns = len(df.columns)
columns_no_iot = total_columns - len(iot_columns)
columns_iot = len(iot_columns)

# Graph how many attributes are related to IoT
plt.figure(figsize=(8, 6))
labels = ['Non-IoT attributes', 'IoT attributes']
counts = [columns_no_iot, columns_iot]
colors = ['pink', 'magenta']
```

```
plt.bar(labels, counts, color=colors)
plt.title('Distribution of the IoT devices-app')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```

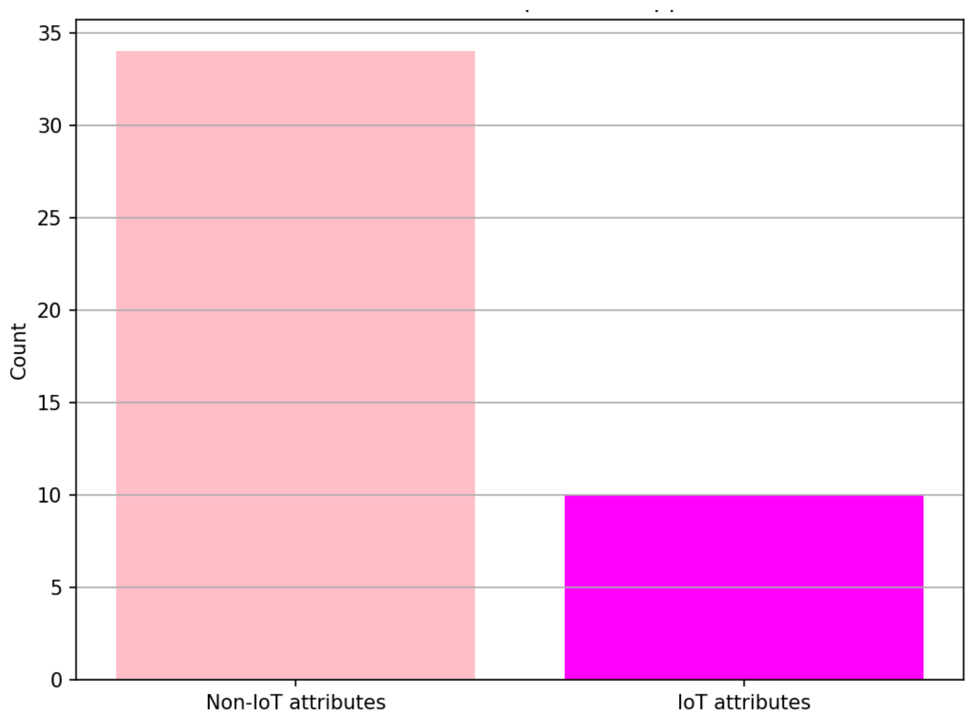


Fig. 14 Distribution of the IoT devices-app

We also did some tests with the libraries, I used some code from Github repositories and examples on blogs

```
import pandas as pd
import numpy as np
import math

midwest = pd.read_csv('midwest.csv')

def calc_information_gain(data, split_name, target_name):
    """
    Calculate information gain given a data set, column to split
on, and target
    """
    # Calculate the original entropy
    original_entropy = calc_entropy(data[target_name])

    #Find the unique values in the column
    values = data[split_name].unique()

    # Make two subsets of the data, based on the unique values
    left_split = data[data[split_name] == values[0]]
    right_split = data[data[split_name] == values[1]]

    # Loop through the splits and calculate the subset entropies
    to_subtract = 0
    for subset in [left_split, right_split]:
        prob = (subset.shape[0] / data.shape[0])
        to_subtract += prob * calc_entropy(subset[target_name])

    # Return information gain
    return original_entropy - to_subtract

def calc_entropy(column):
    """
    Calculate entropy given a pandas series, list, or numpy
array.
    """
    # Compute the counts of each unique value in the column
    counts = np.bincount(column)
    # Divide by the total column length to get a probability
    probabilities = counts / len(column)

    # Initialize the entropy to 0
    entropy = 0
    # Loop through the probabilities, and add each one to the
total entropy
    for prob in probabilities:
        if prob > 0:
            # use log from math and set base to 2
            entropy += prob * math.log(prob, 2)

    return -entropy

# Calculate the entropy of the Label column in midwest
print(calc_entropy(midwest['age']))
```

Also start to work with the scikit-learn library

```
from sklearn.datasets import fetch_openml

X_adult, y_adult = fetch_openml("adult", version=2,
return_X_y=True)

# Remove redundant and non-feature columns
X_adult = X_adult.drop(["education-num", "fnlwgt"],
axis="columns")

print(X_adult.dtypes)
```

6. Pre-Processing

In this dataset preprocessing stage, we had four main objectives: remove duplicate samples, replace null or empty values and substitute them with the average value of the attribute, apply label encoding, and perform min-max scaling.

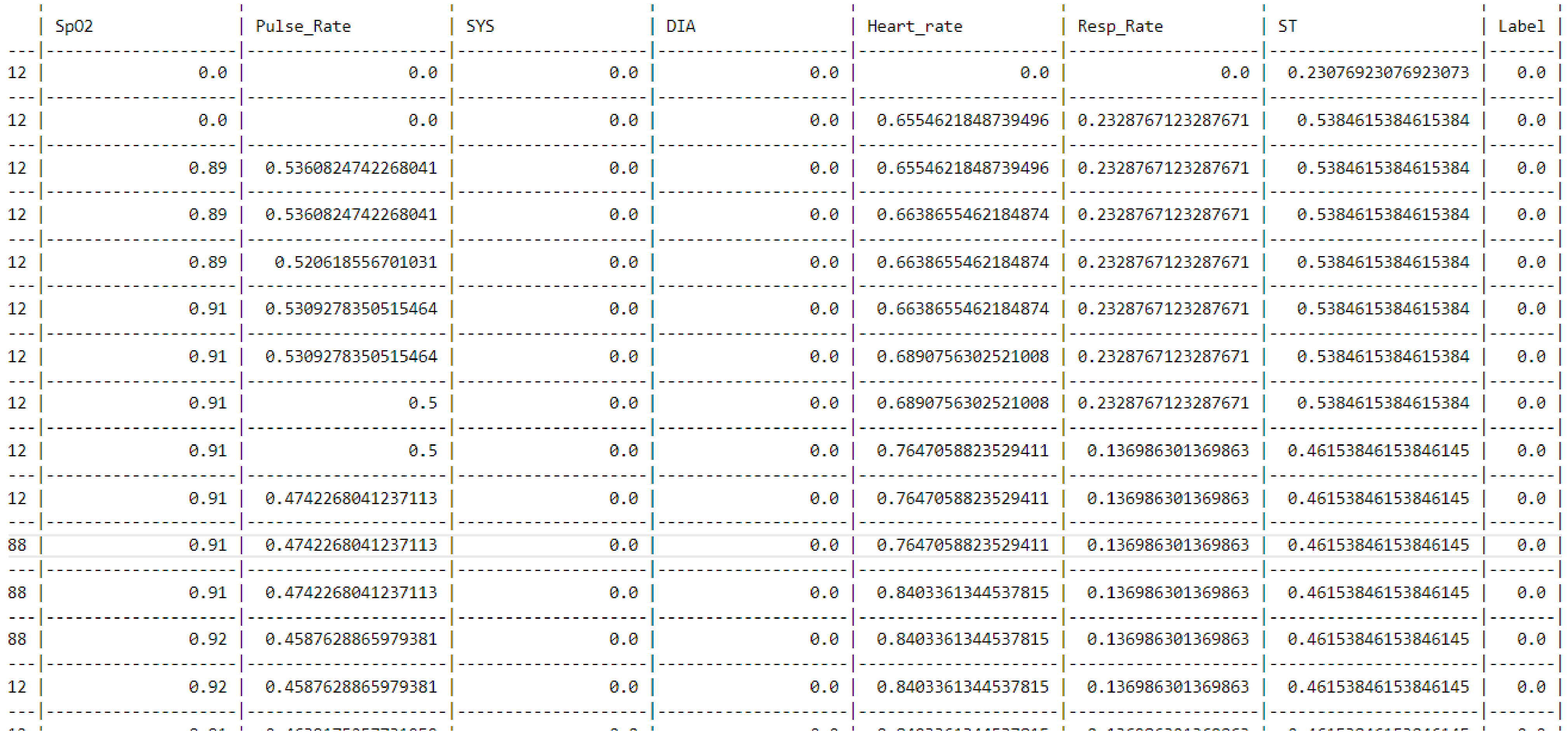
To remove duplicate samples, we used Pandas drop_duplicates method. This method allows us to eliminate rows that contain the same information, leaving only a single sample.

Regarding the removal of null or empty values and their replacement with the average value of the attribute, we used Pandas fillna method. However, our dataset does not have attributes with empty values, so we omitted this processing step.

To transform categorical data, we used label encoding with the LabelEncoder function from Scikit-learn. This function encodes target labels with values between 0 and n_classes-1. We used the fit_transform method of LabelEncoder, which takes the fit label encoder as a parameter and returns encoded labels. We applied this method to each column and reassigned the values to the column.

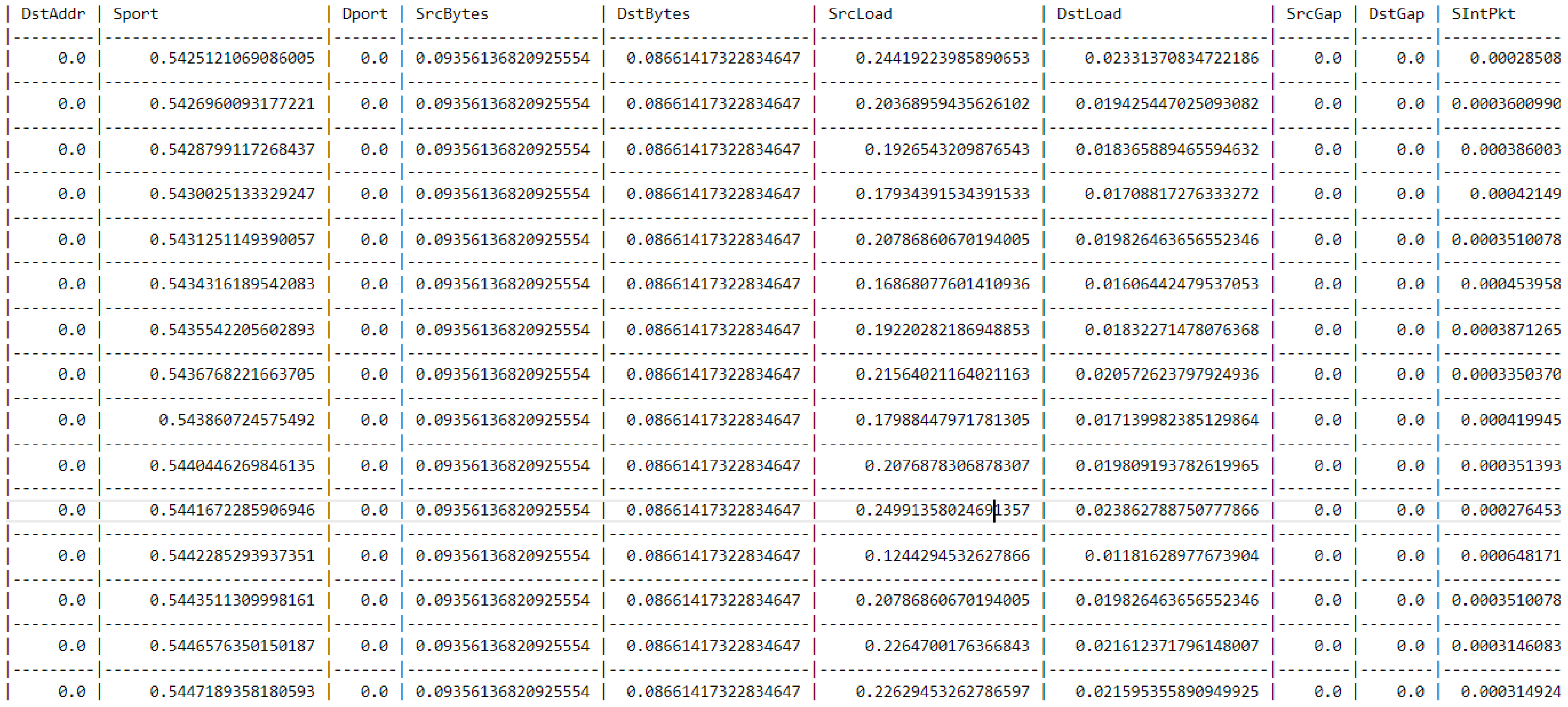
Finally, to normalize the data, we performed Min-Max scaling using the MinMaxScaler function from Scikit-learn.

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, MinMaxScaler

EHMS = pd.read_csv('../WUSTL-EHMS/wustl-ehms-2020.csv')
df = pd.DataFrame(EHMS)

#We delete the irrelevant columns, in this case
df = df.drop(['Dir', 'Flgs'], axis=1)

# Delete duplicates
df = df.drop_duplicates()

#Identify the categorical columns to apply the LabelEncoder
#We change manually this attribute ‘cause the system did not detect it
as categorical
df['Dport'] = df['Dport'].astype('object')
categorical_columns = df.select_dtypes(include=['object']).columns

# We apply the LabelEncoder in the categorical_columns
label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform( ( df[col] ))
    label_encoders[col] = le


# Aply the Min-Max Scaling to all the columns
scaler = MinMaxScaler()
df[df.columns] = scaler.fit_transform(df[df.columns])

#Show the dataframe
print(df)

#We save the new dataset
df.to_csv('dataset_pre_processed.csv', index=False)
```



```
    | SpO2               | Pulse_Rate         | SYS                | DIA                | Heart_rate         | Resp_Rate          | ST                  | Label |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |                0.0 |                0.0 |                0.0 |                0.0 |                0.0 |                0.0 | 0.23076923076923073 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |                0.0 |                0.0 |                0.0 |                0.0 | 0.6554621848739496 | 0.2328767123287671 |  0.5384615384615384 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.89 | 0.5360824742268041 |                0.0 |                0.0 | 0.6554621848739496 | 0.2328767123287671 |  0.5384615384615384 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.89 | 0.5360824742268041 |                0.0 |                0.0 | 0.6638655462184874 | 0.2328767123287671 |  0.5384615384615384 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.89 |  0.520618556701031 |                0.0 |                0.0 | 0.6638655462184874 | 0.2328767123287671 |  0.5384615384615384 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.91 | 0.5309278350515464 |                0.0 |                0.0 | 0.6638655462184874 | 0.2328767123287671 |  0.5384615384615384 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.91 | 0.5309278350515464 |                0.0 |                0.0 | 0.6890756302521008 | 0.2328767123287671 |  0.5384615384615384 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.91 |                0.5 |                0.0 |                0.0 | 0.6890756302521008 | 0.2328767123287671 |  0.5384615384615384 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.91 |                0.5 |                0.0 |                0.0 | 0.7647058823529411 |  0.136986301369863 | 0.46153846153846145 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.91 | 0.4742268041237113 |                0.0 |                0.0 | 0.7647058823529411 |  0.136986301369863 | 0.46153846153846145 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
88  |               0.91 | 0.4742268041237113 |                0.0 |                0.0 | 0.7647058823529411 |  0.136986301369863 | 0.46153846153846145 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
88  |               0.91 | 0.4742268041237113 |                0.0 |                0.0 | 0.8403361344537815 |  0.136986301369863 | 0.46153846153846145 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
88  |               0.92 | 0.4587628865979381 |                0.0 |                0.0 | 0.8403361344537815 |  0.136986301369863 | 0.46153846153846145 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
12  |               0.92 | 0.4587628865979381 |                0.0 |                0.0 | 0.8403361344537815 |  0.136986301369863 | 0.46153846153846145 |   0.0 |
----|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|---------------------|-------|
```

Fig. 15 Part of the results of the dataset preprocessing



```
| DstAddr  | Sport                | Dport | SrcBytes            | DstBytes            | SrcLoad             | DstLoad              | SrcGap | DstGap | SIntPkt
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5425121069086005 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.24419223985890653 |  0.02331370834722186 |    0.0 |    0.0 |    0.00028508
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5426960093177221 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.20368959435626102 | 0.019425447025093082 |    0.0 |    0.0 | 0.0003600990
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |    0.542879117268437 |   0.0 | 0.09356136820925554 | 0.08661417322834647 |  0.1926543209876543 | 0.018365889465594632 |    0.0 |    0.0 |   0.000386003
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5430025133329247 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.17934391534391533 |  0.01708817276333272 |    0.0 |    0.0 |    0.00042149
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5431251149390057 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.20786860670194005 | 0.019826463656552346 |    0.0 |    0.0 | 0.0003510078
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5434316189542083 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.16868077601410936 |  0.01606442479537053 |    0.0 |    0.0 |   0.000453958
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5435542205602893 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.19220282186948853 |  0.01832271478076368 |    0.0 |    0.0 | 0.0003871265
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5436768221663705 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.21564021164021163 | 0.020572623797924936 |    0.0 |    0.0 | 0.0003350370
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |    0.543860724575492 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.17988447971781305 | 0.017139982385129864 |    0.0 |    0.0 |   0.000419945
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5440446269846135 |   0.0 | 0.09356136820925554 | 0.08661417322834647 |  0.2076878306878307 | 0.019809193782619965 |    0.0 |    0.0 |   0.000351393
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5441672285906946 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.24991358024691357 | 0.023862788750777866 |    0.0 |    0.0 |   0.000276453
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5442285293937351 |   0.0 | 0.09356136820925554 | 0.08661417322834647 |  0.1244294532627866 |  0.01181628977673904 |    0.0 |    0.0 |   0.000648171
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5443511309998161 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.20786860670194005 | 0.019826463656552346 |    0.0 |    0.0 | 0.0003510078
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5446576350150187 |   0.0 | 0.09356136820925554 | 0.08661417322834647 |  0.2264700176366843 | 0.021612371796148007 |    0.0 |    0.0 | 0.0003146083
|----------|----------------------|-------|---------------------|---------------------|---------------------|----------------------|--------|--------|--------------
|      0.0 |   0.5447189358180593 |   0.0 | 0.09356136820925554 | 0.08661417322834647 | 0.22629453262786597 | 0.021595355890949925 |    0.0 |    0.0 |   0.000314924
```

Fig. 16 Part of the results of the dataset preprocessing

Related Work

Year	Study	Solution Proposal	Dataset	Algorithms of ML	Advantages	Disadvantages
2020	Intrusion Detection System for Healthcare Systems Using Medical and Network Data: A Comparison Study [2]	Combine the network flow metrics along with the patient's biometrics as features to enhance the system performance	wustl-ehms - 2020	Random Forest(RF) k-nearest-neighbor algorithm (KNN) Support Vector Machines (SVM) Artificial Neural Network(ANN)	The AUC could be enhanced by up to 25% by combining the flow metrics and biometrics data ANN shows the highest performance compared to other methods with an AUC score of 92.98%.	Requires hyperparameter tuning, prediction time is high, feature scaling was not performed. It only considers Man-In-The Middle (MITM) attacks on the system. The ANN model need much more training time than the others
2022	A tree classifier based network intrusion detection model for Internet of Medical Things [14]	The work aimed to design a lightweight intrusion detection model that deploys the feature scaling technique with the random forest model.	wustl-ehms - 2020	Random Forest(RF) Logistic Regression (LR) Decision Tree (DT) Extra Tree	The model achieved a significant reduction in classification time through processes like scaling, feature elimination, encoding, and feature selection The model reported a high accuracy.	The work has scope for including more attack types and exploring lightweight deep neural network models to potentially improve the detection rate and AUC score The model employs hyperparameter tuning to obtain the best estimator for classifying anomalies and normal samples.
2023	Machine Learning Algorithms for the Detection of Threats in IoT Healthcare [15]	In the context of IoT healthcare, this study mainly examined machine learning techniques for threat identification	wustl-ehms - 2020	Logistic Regression (LR) Random forest learning (RF) Gradual Boosting (GB) Support Vector Machines (SVM) Naive Bayes (NB) k-nearest-neighbor algorithm (KNN) Decision Tree (DT)	Random Forest performed the best; it had a precision percentage of 99.80%	While individual decision trees are easy to interpret, a random forest is more complex.

Analysis of the traffic network

1. Principal Component Analysis (PCA)

The PCA calculates the principal components based on the covariance structure of the scaled data. Two tests were performed: one using min-max scaling and another using standard scaling. It is necessary to standardize the data because PCA is based on the variance of the data, and standardizing ensures that all features contribute equally to the analysis.

After that, the selection of the number of principal components: **pca.explained_variance_ratio_** contains the proportion of the total variance explained by each principal component. We also use, **np.cumsum()** ‘cause it calculates the cumulative sum of these proportions, which gives us the total variance explained by including more principal components. It’s important to say that **np.argmax()** is used to find the first index where 95% of explained variance is reached or exceeded.

As the next step, PCA was initialized with the selected number of principal components. With the **fit_transform(df)** method, PCA is adjusted to the DataFrame data and transforms the data into the reduced principal components space **(X_pca)**.

We have to restructure the DataFrame with **pd.DataFrame()** a new DataFrame df_pca is created using the transformed data **X_pca.columns=[f'PC{i+1}' for i in range(n_components)]** makes the columns of the df_pca DataFrame be named as PC1, PC2, ..., PCn according to the number of principal components.

Finally, we print the principal components (Eigenvectors) obtained from PCA. Each row corresponds to a principal component, and each column to an original feature. We print the eigenvalues corresponding to each principal component, which indicate the amount of variance explained by each component. Lastly, we print the proportion of variance explained by each principal component, which was already used to determine the number of necessary principal components.

```
import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder, StandardScaler,
MinMaxScaler

from sklearn.decomposition import PCA



EHMS = pd.read_csv('../WUSTL-EHMS/wustl-ehms-2020.csv')

df = pd.DataFrame(EHMS)
df = df.drop(['Dir', 'Flgs'], axis=1)

df = df.drop_duplicates()


df['Dport'] = df['Dport'].astype('object')

categorical_columns = df.select_dtypes(include=['object']).columns


label_encoders = {}

for col in categorical_columns:

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le


#Apply the Min-Max Scaling to all the columns

#scaler = MinMaxScaler()

#df[df.columns] = scaler.fit_transform(df[df.columns])


# Apply StandardScaler to all the columns

scaler = StandardScaler()

df[df.columns] = scaler.fit_transform(df[df.columns])


# Apply PCA

pca = PCA()


# Select the number of principal components to explain the 95% of
variance
```

```python
explained_variance = np.cumsum(pca.explained_variance_ratio_)

n_components = np.argmax(explained_variance >= 0.95) + 1


print(f'Number of the principal components to explain the 95% of
variance: {n_components}')


# Inicialize the PCA with the number of selected components

pca = PCA(n_components=n_components)

X_pca = pca.fit_transform(df)


# Create a new Dataframe

df_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in
range(n_components)])

# Show the new Dataframe

print(df_pca)

# Save the new dataset

df_pca.to_csv('dataset_PCA.csv', index=False)


# Show details of the results of the PCA

print("Principal Components (Eigenvectors):\n", pca.components_)

print("\nVariance (Eigenvalues):\n", pca.explained_variance_)

print("\nProportion of Variance:\n", pca.explained_variance_ratio_)
```

Results

Regarding data exploration, we identified the distribution of normal records and those affected by MITM attacks. We observed an 80 to 20 ratio, as well as the identification of classes, attributes, data types, and the purpose of each piece of data.

In terms of graphical data visualization, using Matplotlib, we generated graphs representing the distribution of the most important variables. These graphs helped us visualize patterns and trends in the data. We approached the suggested tool Power BI, and we are still in an initial phase, seeking to improve visualization with this new tool to obtain more interactive and detailed graphs, expanding visualization possibilities.

Speaking of libraries, for data processing, we used numpy and pandas. I should mention that I had previously used numpy as well as sympy to implement algorithms for the numerical methods course. Pandas helps a lot in sectioning, cleaning, and transforming data. Using Scikit-learn, we applied basic codes from the tutorial provided by the documentation but managed to understand that this library will help us carry out regression, classification, and clustering algorithms. Regarding Hyperopt, I understand that it seeks to improve model performance, but I need to experiment with it more. Therefore, I believe we laid an important foundation for the subsequent stages.

Finally, I realized the importance of data quality for obtaining accurate results. We identified and handled missing and anomalous data, improving the reliability of our analyses.

Conclusions

In this stage, we delved into the world of data and its processing. We familiarized ourselves with essential libraries like Numpy, Pandas, Scikit-learn, and Hyperopt, and began exploring Power BI to enhance graph visualization, although we initially used Matplotlib. We learned the fundamental terminology and theory, applying them in practice with the help of these libraries. This allowed us to execute many of the necessary operations and formulas to process data effectively. Additionally, the article we reviewed provided us with valuable context, paving the way for developing solid solutions in the upcoming stages.

References

1. *Hyperopt Documentation*. (s. f.). https://hyperopt.github.io/hyperopt/

2. Hady, A. A., Ghubaish, A., Salman, T., Unal, D., & Jain, R. (2020). Intrusion Detection System for Healthcare Systems Using Medical and Network Data: A Comparison Study. *IEEE Access*, *8*, 106576-106584. https://doi.org/10.1109/access.2020.3000421

3. *User Guide*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/user_guide.html

4. *Ganancia de entropía e información en árboles de decisión*. (2020, 4 diciembre). ICHI.PRO. https://ichi.pro/es/ganancia-de-entropia-e-informacion-en-arboles-de-decision-219745150542483
5. *NumPy: the absolute basics for beginners — NumPy v2.0 Manual*. (s. f.). https://numpy.org/doc/stable/user/absolute_beginners.html

6. *Release Highlights for scikit-learn 1.4*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/auto_examples/release_highlights/plot_release_highlights_1_4_0.html#sphx-glr-auto-examples-release-highlights-plot-release-highlights-1-4-0-py
7. *LabelEncoder — scikit-learn 1.5.1 documentation*. (s. f.). Scikit-learn.

   https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

8. MinMaxScaler — scikit-learn 1.5.1 documentation. (s. f.). Scikit-learn.

   https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler.set_output

9. *¿Qué es un bosque aleatorio? | IBM*. (s. f.).

   https://www.ibm.com/mx-es/topics/random-forest

10. *¿Qué es KNN? | IBM*. (s. f.). https://www.ibm.com/mx-es/topics/knn

11. Liu, C. (s. f.). *A top machine learning algorithm explained: Support Vector Machines (SVM) - KDnuggets*. KDnuggets. https://www.kdnuggets.com/2020/03/machine-learning-algorithm-svm-explained.html
12. Logunova, I. (2022, 18 octubre). *Support Vector Machine Algorithm*. Guide To Support Vector Machine (SVM) Algorithm. https://serokell.io/blog/support-vector-machine-algorithm
13. Ali, A. (2021, 10 diciembre). Artificial Neural Network (ANN) with Practical Implementation. *Medium*. https://medium.com/machine-learning-researcher/artificial-neural-network-ann-4481fa33d85a
14. Gupta, K., Sharma, D. K., Gupta, K. D., & Kumar, A. (2022). A tree classifier based network intrusion detection model for Internet of Medical Things. *Computers & Electrical Engineering*, *102*, 108158. https://doi.org/10.1016/j.compeleceng.2022.108158
15. V, A., Shenbagavalli, P., T, S., B, M., R, M. T., & Anitha, K. (2023). Machine learning algorithms for the detection of threats in IoT healthcare. *EE International Conference On Emerging Research In Computational Science – ICERCS'23*. https://doi.org/10.1109/icercs57948.2023.10434124
16. Hady, A. A., Ghubaish, A., Salman, T., Unal, D., & Jain, R. (2020). Intrusion Detection System for Healthcare Systems Using Medical and Network Data: A Comparison Study. EHMS testbed [Scheme]. *IEEE Access*, *8*, 106576-106584. https://doi.org/10.1109/access.2020.3000421