Task 1

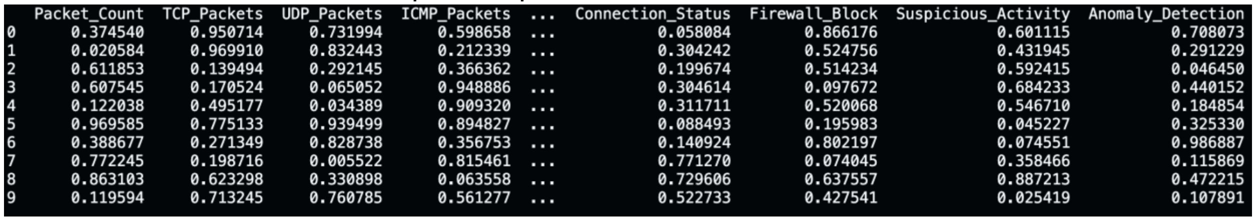
deep learning is a field in machine learning. The idea is to act like a human brain and create neural networks, which are connected by nodes, and every layer process data using a hierarchical model. Its main requirement is to have a lot of data for training. An example of its use is image recognition, where the deep learning model is able to recognize what is presented in an image (human, dog, etc.). It's different from other algorithms where it is able to learn on its own without human interaction. feed forward neural network is an example of deep learning. It has an input, output, and in-between layers. Each layer has nodes that then manipulates this data. The numbers that are found are called weights. The idea is to have the least amount of difference between the predictions and actual labels by using weights. This algorithm optimizes loss, which is the difference between classified output and correct output. So the goal is to minimize this loss function.

Practical example
In this practical example, we will be making a 1000 sample and 10 feature random network traffic dataset where we will be utilizing the feed-forward neural networks to guess the predicted class for network traffic.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# Set random seed for reproducibility
np.random.seed(42)

# Generate random features
X = np.random.rand(1000, 10)  # 1000 samples, 10 features

# Assign labels based on a simple criterion (e.g., sum of features)
y = (X.sum(axis=1) > 5).astype(int)  # Binary classification task

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)

# Display the first 10 rows of the synthetic dataset with cybersecurity-related column
names
column_names = ["Packet_Count", "TCP_Packets", "UDP_Packets", "ICMP_Packets",
"Data_Transfer",
            "Malware_Bytes", "Connection_Status", "Firewall_Block", "Suspicious_Activity",
            "Anomaly_Detection"]
print("First 10 rows of the synthetic dataset:")
print(pd.DataFrame(X_test[:10], columns=column_names))

# Define the FFNN model
model = Sequential([
   Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
   Dense(32, activation='relu'),
   Dense(1, activation='sigmoid')
])

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

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)

# Predictions for the first 10 rows of the test set
predictions = model.predict(X_test[:10])
predicted_classes = (predictions > 0.5).astype("int32")
print("Predicted Classes for the First 10 Rows:")
print(predicted_classes)
```

Here are 10 rows and their respective predicted classes:



```
   Packet_Count  TCP_Packets  UDP_Packets  ICMP_Packets  ...  Connection_Status  Firewall_Block  Suspicious_Activity  Anomaly_Detection
0      0.374540     0.950714     0.731994      0.598658  ...           0.058084        0.866176             0.601115           0.708073
1      0.020584     0.969910     0.832443      0.212339  ...           0.304242        0.524756             0.431945           0.291229
2      0.611853     0.139494     0.292145      0.366362  ...           0.199674        0.514234             0.592415           0.046450
3      0.607545     0.170524     0.065052      0.948886  ...           0.304614        0.097672             0.684233           0.440152
4      0.122038     0.495177     0.034389      0.909320  ...           0.311711        0.520068             0.546710           0.184854
5      0.969585     0.775133     0.939499      0.894827  ...           0.088493        0.195983             0.045227           0.325330
6      0.388677     0.271349     0.828738      0.356753  ...           0.140924        0.802197             0.074551           0.986887
7      0.772245     0.198716     0.005522      0.815461  ...           0.771270        0.074045             0.358466           0.115869
8      0.863103     0.623298     0.330898      0.063558  ...           0.729606        0.637557             0.887213           0.472215
9      0.119594     0.713245     0.760785      0.561277  ...           0.522733        0.427541             0.025419           0.107891
```

```
[[0]
 [0]
 [0]
 [0]
 [0]
 [1]
[1]
[0]
[1]
[1]]
```