Workflow!!

Dataset :: <https://www.kaggle.com/datasets/jsrojas/labeled-network-traffic-flows-114-applications?resource=download>Dataset Attributes present in this dataset is sufficient to work on it and we might need to exclude unnecessary data from it. We don’t have type column where we will detect whether it is normal or anomaly that prediction we will do.

Full Code Implementation for our project which will help us to detect the anomaly probability  
  
import pandas as pd

from sklearn.ensemble import IsolationForest

from sklearn.metrics import precision\_recall\_fscore\_support

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense

# Load the dataset

df = pd.read\_csv("IP\_1.csv")

# Splitting the dataset into features and target

X = df.drop(columns=['category']) # Assuming 'category' column specifies the labels

y = df['category']

# 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)

# Method for anomaly detection using Isolation Forest

def anomaly\_detection\_isolation\_forest(X\_train, X\_test):

# Train Isolation Forest model

model = IsolationForest(contamination=0.1) # Contamination is the expected proportion of anomalies

model.fit(X\_train)

# Predict anomalies

anomaly\_scores\_train = model.decision\_function(X\_train)

anomaly\_scores\_test = model.decision\_function(X\_test)

return anomaly\_scores\_train, anomaly\_scores\_test

# Method for anomaly detection using Autoencoder

def anomaly\_detection\_autoencoder(X\_train, X\_test):

# Define Autoencoder model

input\_dim = X\_train.shape[1]

encoding\_dim = 64 # Adjust as needed

input\_layer = Input(shape=(input\_dim,))

encoded = Dense(encoding\_dim, activation='relu')(input\_layer)

decoded = Dense(input\_dim, activation='sigmoid')(encoded)

autoencoder = Model(input\_layer, decoded)

# Compile and train the model

autoencoder.compile(optimizer='adam', loss='mean\_squared\_error')

autoencoder.fit(X\_train, X\_train, epochs=50, batch\_size=32, shuffle=True, validation\_data=(X\_test, X\_test))

# Predict reconstruction errors

predictions\_train = autoencoder.predict(X\_train)

predictions\_test = autoencoder.predict(X\_test)

mse\_train = np.mean(np.power(X\_train - predictions\_train, 2), axis=1)

mse\_test = np.mean(np.power(X\_test - predictions\_test, 2), axis=1)

return mse\_train, mse\_test

# Call anomaly detection methods

anomaly\_scores\_train\_iso, anomaly\_scores\_test\_iso = anomaly\_detection\_isolation\_forest(X\_train, X\_test)

mse\_train\_auto, mse\_test\_auto = anomaly\_detection\_autoencoder(X\_train, X\_test)

# Convert anomaly scores to binary labels (1 for anomaly, 0 for normal)

y\_pred\_iso\_train = (anomaly\_scores\_train\_iso < 0).astype(int)

y\_pred\_iso\_test = (anomaly\_scores\_test\_iso < 0).astype(int)

y\_pred\_auto\_train = (mse\_train\_auto > threshold\_auto).astype(int) # Adjust threshold\_auto as needed

y\_pred\_auto\_test = (mse\_test\_auto > threshold\_auto).astype(int)

# Calculate precision, recall, and F1-score for each category

metrics\_iso\_train = precision\_recall\_fscore\_support(y\_train, y\_pred\_iso\_train, labels=['Anomaly', 'DNS', 'HTTP', 'SNMP', 'TLS'])

metrics\_iso\_test = precision\_recall\_fscore\_support(y\_test, y\_pred\_iso\_test, labels=['Anomaly', 'DNS', 'HTTP', 'SNMP', 'TLS'])

metrics\_auto\_train = precision\_recall\_fscore\_support(y\_train, y\_pred\_auto\_train, labels=['Anomaly', 'DNS', 'HTTP', 'SNMP', 'TLS'])

metrics\_auto\_test = precision\_recall\_fscore\_support(y\_test, y\_pred\_auto\_test, labels=['Anomaly', 'DNS', 'HTTP', 'SNMP', 'TLS'])

# Create a DataFrame to store the metrics

index = ['Anomaly', 'DNS', 'HTTP', 'SNMP', 'TLS']

columns = ['Precision', 'Recall', 'F1-score']

df\_metrics\_iso\_train = pd.DataFrame(list(metrics\_iso\_train[:-1]), columns=index, index=columns).T

df\_metrics\_iso\_test = pd.DataFrame(list(metrics\_iso\_test[:-1]), columns=index, index=columns).T

df\_metrics\_auto\_train = pd.DataFrame(list(metrics\_auto\_train[:-1]), columns=index, index=columns).T

df\_metrics\_auto\_test = pd.DataFrame(list(metrics\_auto\_test[:-1]), columns=index, index=columns).T

# Display the metrics in tabular format

print("Isolation Forest - Train:")

print(df\_metrics\_iso\_train)

print("\nIsolation Forest - Test:")

print(df\_metrics\_iso\_test)

print("\nAutoencoder - Train:")

print(df\_metrics\_auto\_train)

print("\nAutoencoder - Test:")

print(df\_metrics\_auto\_test)