from sklearn.datasets import make\_classification

import pandas as pd

import numpy as np

from collections import Counter

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras import layers, losses

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

X, y = make\_classification(n\_samples=100000, n\_features=32, n\_informative=32,

n\_redundant=0, n\_repeated=0, n\_classes=2,

n\_clusters\_per\_class=1,

weights=[0.995, 0.005],

class\_sep=0.5, random\_state=0)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print('The number of records in the training dataset is', X\_train.shape[0])

print('The number of records in the test dataset is', X\_test.shape[0])

print(f"The training dataset has {sorted(Counter(y\_train).items())[0][1]} records for the majority class and {sorted(Counter(y\_train).items())[1][1]} records for the minority class.")

X\_train\_normal = X\_train[np.where(y\_train == 0)]

input = tf.keras.layers.Input(shape=(32,))

encoder = tf.keras.Sequential([

layers.Dense(16, activation='relu'),

layers.Dense(8, activation='relu'),

layers.Dense(4, activation='relu')])(input)

decoder = tf.keras.Sequential([

layers.Dense(8, activation="relu"),

layers.Dense(16, activation="relu"),

layers.Dense(32, activation="sigmoid")])(encoder)

autoencoder = tf.keras.Model(inputs=input, outputs=decoder)

autoencoder.compile(optimizer='adam', loss='mae')

history = autoencoder.fit(X\_train\_normal, X\_train\_normal,

epochs=10,

batch\_size=64,

validation\_data=(X\_test, X\_test),

shuffle=True)

prediction = autoencoder.predict(X\_test)

prediction\_loss = tf.keras.losses.mae(prediction, X\_test)

loss\_threshold = np.percentile(prediction\_loss, 98)

print(f'The prediction loss threshold for 2% of outliers is {loss\_threshold:.2f}')

sns.histplot(prediction\_loss, bins=30, alpha=0.8)

plt.axvline(x=loss\_threshold, color='orange')

threshold\_prediction = [0 if i < loss\_threshold else 1 for i in prediction\_loss]

print(classification\_report(y\_test, threshold\_prediction))