# Summer Project Final Report for Manuscript_22070243038

*by* Ritwik Dubey

---

**Submission date:** 05-Jun-2023 11:40PM (UTC+0530)
**Submission ID:** 2109680447
**File name:** Summer_Project_Final_Report_22070243038_Ritwik_Dubey.pdf (1.19M)
**Word count:** 4529
**Character count:** 27056

## Summer Project Final Report for Manuscript_22070243038

ORIGINALITY REPORT

| 2% | 2% | 0% | % |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | blog.superannotate.com<br>Internet Source | 1% |
| 2 | kyutech.repo.nii.ac.jp<br>Internet Source | 1% |

# Autoencoder-based Anomaly Detection in Alphabet Inc. (Google) Dataset

**THESIS SUBMITTED TO**

**Symbiosis Institute of Geoinformatics (SIG)**

**FOR PARTIAL FULFILLMENT OF THE M. Sc. DEGREE**

**By**

**RITWIK DUBEY**

**BATCH: 2022-2024**

**PRN : 22070243038**

**Symbiosis Institute of Geoinformatics**

**Symbiosis International (Deemed University)**

**5th Floor, Atur Centre, Gokhale Cross**

**Road, Model Colony, Pune – 411016**

# <u>CERTIFICATE</u>

This is to Certify that the thesis titled **'Autoencoder-based Anomaly Detection in Alphabet Inc. (Google) Dataset'** is a bona fide work done by Mr. **Ritwik Dubey**, at **Symbiosis Institute of Geoinformatics,** under the supervision

**<u>Supervisor, Internal</u>**

**Dr. Yogesh Rajput**

**Symbiosis Institute of Geoinformatics**

# INDEX:

# I. ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my esteemed supervisors, Dr. Yogesh Rajput and Dr. Vidya Sandeep Patkar, for their invaluable guidance, support, and encouragement throughout this research project. Their deep knowledge, expertise, and unwavering commitment to academic excellence have been instrumental in shaping the direction of this study and enhancing its quality. I am truly fortunate to have had the privilege of working under their mentorship.

I would also like to extend my heartfelt appreciation to Mr. Sahil Shah, a dedicated faculty member at Symbiosis Institute of Geoinformatics. His insightful feedback, constructive criticism, and continuous motivation have significantly contributed to the development and refinement of this research work. His enthusiasm for teaching and research has been a constant source of inspiration for me.

I am grateful to the entire faculty of Symbiosis Institute of Geoinformatics for providing a stimulating academic environment and fostering a culture of intellectual growth. Their passion for imparting knowledge and their commitment to nurturing future professionals have been pivotal in shaping my educational journey.

I would like to acknowledge the support and cooperation of my fellow colleagues and classmates who have provided valuable insights and engaging discussions throughout this project. Their diverse perspectives and collaborative spirit have enriched my research experience.

Furthermore, I extend my heartfelt thanks to the staff and administration of Symbiosis Institute of Geoinformatics for their assistance and cooperation in facilitating the smooth progress of this research project. Their administrative support and logistical arrangements have been integral to the successful completion of this endeavour.

Last but not least, I would like to express my deepest gratitude to my family and friends for their unwavering love, understanding, and encouragement. Their constant support and belief in my abilities have been the driving force behind my academic pursuits.

In conclusion, I am truly grateful to all the individuals mentioned above and countless others who have directly or indirectly contributed to the successful completion of this research project. Their collective efforts and unwavering support have made this endeavour possible.

# II. LIST OF FIGURES

# III. PREFACE

In embarking on this research project, I was driven by a deep curiosity and a desire to contribute to the field of anomaly detection in time series data. Exploring the detection of anomalies in various domains has always fascinated me, particularly its applications in finance and cybersecurity. Through this project, I aimed to shed light on the effectiveness of different autoencoder architectures in identifying anomalies in stock price data.

During the course of this research, I encountered numerous challenges and learned valuable lessons along the way. The journey involved carefully selecting and preprocessing a comprehensive dataset of historical stock prices, considering the complex nature of stock market dynamics. I delved into the realm of deep learning and worked extensively with autoencoders, studying their mechanisms and fine-tuning their architectures to achieve optimal performance.

I am immensely grateful to my advisor and mentors who provided guidance and support throughout this endeavour. Their expertise and insights were instrumental in shaping the research direction and refining the experimental setup. Additionally, I would like to express my appreciation to the research participants who generously shared their expertise and contributed to the project's success.

This research project holds immense potential to contribute to anomaly detection methods and their application in financial markets. By evaluating the performance of LSTM Autoencoder and Simple Autoencoder architectures, we aim to provide practitioners and researchers with valuable insights into their strengths and limitations. Furthermore, the findings have the potential to enhance decision-making processes and risk management strategies in the stock market.

It is my hope that this research project will inspire further exploration and advancements in the field of anomaly detection. I invite readers to delve into the following chapters, where the methodology, results, and implications of the study are presented in detail. May this work serve as a catalyst for innovative approaches and foster a deeper understanding of anomaly detection in time series data.

[ Ritwik Dubey ]

# IV. INTRODUCTION

**a)** In recent years, the proliferation of data in various industries has made it necessary to develop efficient methods for detecting anomalies in data. Anomaly detection is the process of identifying data points or patterns that deviate significantly from the norm or expected behaviour. It has various applications in different domains such as finance, healthcare, and cybersecurity.

In this project, we focus on detecting anomalies in Alphabet Inc. (Google) dataset using LSTM Autoencoder. LSTM Autoencoder is a deep learning technique that is designed to work with sequence data, such as time-series data, and it has shown great promise in detecting anomalies in such data.

Detecting anomalies in time series data, such as stock prices, is a crucial task in various domains, including finance, cybersecurity, and industrial monitoring. Anomaly detection techniques, including machine learning algorithms, play a vital role in identifying unusual patterns or behaviours that deviate from the expected norm. Autoencoders, a type of unsupervised neural network, have shown promise in anomaly detection tasks by reconstructing the input data and identifying instances with high reconstruction errors.

In this study, we aim to compare two different autoencoder architectures, namely the Long Short-Term Memory (LSTM) Autoencoder and the Simple Autoencoder, in their effectiveness for detecting anomalies in stock price data. The LSTM Autoencoder is a variant of the recurrent neural network (RNN) that can capture long-term dependencies and sequential patterns in time series data. On the other hand, the Simple Autoencoder is a basic feed-forward neural network that learns to encode and decode the input data without considering its temporal nature.

The primary research question driving this study is: “Which autoencoder architecture, LSTM Autoencoder or Simple Autoencoder, is more effective in detecting anomalies in stock price data?” To answer this question, we will compare the performance of the two autoencoder architectures based on their mean squared error (MSE) scores, reconstruction errors, and their ability to accurately identify and flag anomalous instances in the dataset.

We will conduct our experiments using a comprehensive dataset of historical stock prices, including both normal and anomalous patterns. The dataset will be pre-processed and split into training and testing sets. We will train separate LSTM Autoencoder and Simple Autoencoder models on the training data, optimizing them to minimize the reconstruction error. Subsequently, we will evaluate the performance of both models on the testing data and compare their results.

The outcomes of this research will provide valuable insights into the effectiveness of different autoencoder architectures for anomaly detection in stock price data. These findings can have significant implications for financial institutions, traders, and analysts who rely on accurate anomaly detection to make informed decisions and mitigate risks in the stock market. Additionally, the study will contribute to the broader field of anomaly detection in time series data and serve as a foundation for future research in developing more advanced and efficient models for detecting anomalies in financial markets.

Overall, this project has the potential to provide valuable insights into the detection of anomalies in time-series data using LSTM Autoencoder and contribute to the development of more effective anomaly detection methods.

**b) Dataset Description:**

The dataset used in this research project consists of financial data for Alphabet Inc. (Google) obtained from Yahoo Finance. The dataset spans from 2004 to 2023, providing a comprehensive view of the company’s stock market performance over a significant period.

**The dataset contains the following columns:**

**Date:** The date of the recorded stock market data.

**Open:** The opening price of Alphabet Inc.’s stock on a given trading day.

**High:** The highest price reached by Alphabet Inc.’s stock on a given trading day.

**Low:** The lowest price reached by Alphabet Inc.’s stock on a given trading day.

**Close**: The closing price of Alphabet Inc.’s stock on a given trading day.

**Adj Close**: The adjusted closing price of Alphabet Inc.’s stock, accounting for dividends, stock splits, and other corporate actions.

**Volume:** The trading volume, representing the number of shares traded on a given trading day.

The dataset comprises 4,721 rows, each corresponding to a unique trading day. This granularity allows for the analysis of daily price fluctuations and the identification of potential anomalies or unusual patterns in the stock market data.

```python
df = pd.read_csv('GOOG (2).csv')
df
```

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| 0 | 2004-08-20 | 2.515820 | 2.716817 | 2.503118 | 2.697639 | 2.697639 | 458857488 |
| 1 | 2004-08-23 | 2.758411 | 2.826406 | 2.716070 | 2.724787 | 2.724787 | 366857939 |
| 2 | 2004-08-24 | 2.770615 | 2.779581 | 2.579581 | 2.611960 | 2.611960 | 306396159 |
| 3 | 2004-08-25 | 2.614201 | 2.689918 | 2.587302 | 2.640104 | 2.640104 | 184645512 |
| 4 | 2004-08-26 | 2.613952 | 2.688672 | 2.606729 | 2.687676 | 2.687676 | 142572401 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 4715 | 2023-05-15 | 116.489998 | 118.794998 | 116.480003 | 116.959999 | 116.959999 | 22107900 |
| 4716 | 2023-05-16 | 116.830002 | 121.199997 | 116.830002 | 120.089996 | 120.089996 | 32370100 |
| 4717 | 2023-05-17 | 120.180000 | 122.279999 | 119.459999 | 121.480003 | 121.480003 | 26659600 |
| 4718 | 2023-05-18 | 121.559998 | 123.900002 | 121.489998 | 123.519997 | 123.519997 | 27014500 |
| 4719 | 2023-05-19 | 124.199997 | 126.478996 | 122.720001 | 123.250000 | 123.250000 | 30251300 |

4720 rows × 7 columns

```python
df.shape
```

(4720, 7)

**Figure 1. Dataset Description**

```python
import plotly.graph_objects as go
import matplotlib.pyplot as plt
fig = go.Figure()
fig.add_trace(go.Scatter(x=df['Date'], y=df['Close'], name='Close price'))
fig.update_layout(showlegend=True, title='Alphabet Inc. Stock Price 2004-2023')
fig.show()
```


**Figure 2. – Time series visualization**

```python
train = df.loc[
    (df['Date'] >= '2004-12-24') & (df['Date'] <= '2007-06-24') |
    (df['Date'] >= '2007-10-24') & (df['Date'] <= '2008-07-24') |
    (df['Date'] >= '2008-08-24') & (df['Date'] <= '2012-06-24') |
    (df['Date'] >= '2012-06-30') & (df['Date'] <= '2014-05-24') |
    (df['Date'] <= '2014-05-30') & (df['Date'] <= '2017-05-24') |
    (df['Date'] >= '2017-06-01') & (df['Date'] <= '2019-09-30') |
    (df['Date'] >= '2021-07-01') & (df['Date'] <= '2022-04-30')
]

test = df.loc[
    (df['Date'] >= '2020-01-01') & (df['Date'] <= '2021-04-30') |
    (df['Date'] >= '2019-10-24') & (df['Date'] <= '2019-12-24') |
    (df['Date'] >= '2021-05-24') & (df['Date'] <= '2021-06-24') |
    (df['Date'] > '2022-05-01')
]

train.shape, test.shape
```

((3258, 7), (666, 7))

**Figure 4. – Training & Test Data split**

# V. OBJECTIVE (RESEARCH QUESTION)

The main objective of this project is to develop a robust anomaly detection model using LSTM Autoencoder that can accurately identify anomalous behaviour in the Alphabet Inc. (Google) dataset. Specifically, we aim to answer the following research questions:

a) How effective is LSTM Autoencoder in detecting anomalies in time-series data for stock prices in the Google dataset?
b) What is the optimal architecture and hyperparameters of LSTM Autoencoder for detecting anomalies in the Google dataset?
c) How does the performance of LSTM Autoencoder compare with other simple autoencoder for anomaly detection methods in the Google dataset?
d) Can we identify the underlying causes of anomalies detected by LSTM Autoencoder in the Google dataset and provide actionable insights to improve business processes?
e) "Which autoencoder architecture, LSTM Autoencoder or Simple Autoencoder, is more effective in detecting anomalies in stock price data?"
f) This research question aims to compare the performance and effectiveness of the LSTM Autoencoder and Simple Autoencoder in detecting anomalies in stock price data. The comparison will be based on their respective mean squared error (MSE) scores, reconstruction errors, and the ability to identify and flag anomalies in the dataset. By evaluating and comparing these metrics, we can determine which autoencoder architecture is more suitable for anomaly detection in stock price data

By answering these research questions, we aim to contribute to the development of more effective anomaly detection methods for time-series data and provide valuable insights into the detection of anomalies in Alphabet Inc. (Google) dataset.

# VI. Expected Results

These are the expected outcomes of the project:

a. Firstly, it is anticipated that both autoencoder architectures will effectively capture and reconstruct the underlying patterns in stock price data. However, the hypothesis suggests that the LSTM Autoencoder will outperform the Simple Autoencoder in capturing temporal dependencies and sequential patterns. The LSTM Autoencoder is specifically designed for time series data and is capable of modelling long-term dependencies, which are prevalent in stock price movements.
b. In terms of reconstruction errors, it is expected that the LSTM Autoencoder will achieve lower mean squared error (MSE) scores compared to the Simple Autoencoder. The LSTM Autoencoder's ability to capture temporal dependencies should result in more accurate reconstructions of the input data, leading to reduced reconstruction errors. Conversely, the Simple Autoencoder may struggle to accurately reconstruct the stock price data due to its lack of temporal modelling capabilities, resulting in higher MSE scores.
c. Furthermore, both autoencoder architectures are expected to effectively detect anomalies in the stock price data. However, it is anticipated that the LSTM Autoencoder will exhibit superior performance in identifying and flagging anomalous instances. The LSTM Autoencoder's capability to capture long-term dependencies and sequential patterns enables it to differentiate between normal and anomalous stock price patterns more effectively than the Simple Autoencoder.
d. To evaluate the performance of the autoencoder architectures, metrics such as precision, recall, and F1-score will be analysed to assess their ability to correctly identify anomalies while minimizing false positives. It is expected that the LSTM Autoencoder will achieve higher precision, recall, and F1-score compared to the Simple Autoencoder, indicating its superior performance in anomaly detection.

Overall, the project anticipates that the LSTM Autoencoder, with its ability to capture temporal dependencies, will outperform the Simple Autoencoder in terms of reconstruction accuracy and anomaly detection. These expected outcomes will provide valuable insights into the strengths and limitations of different autoencoder architectures for detecting anomalies in stock price data, helping professionals and researchers choose the most suitable approach for their specific requirements.

# VII. Literature Review

[1] “Anomaly Detection with Robust Deep Autoencoders” by Chong Zhou and Randy C. Paffenroth states that this paper introduces Robust Deep Autoencoders (RDA) as a novel extension to deep autoencoders for anomaly detection. RDA splits the input data into two parts, one that can be effectively reconstructed by a deep autoencoder and another part containing outliers and noise. The authors demonstrate the effectiveness of RDA on benchmark problems, achieving approximately 30% improvement over standard autoencoders. They also present generalizations of their approach to grouped sparsity norms, which outperform Isolation Forests by a 73% improvement. [2] “Autoencoder-based network anomaly detection” by Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau: In this paper, the authors propose an Autoencoder-based network anomaly detection method, specifically using Convolutional Autoencoders (CAEs) for dimensionality reduction. The CAE-based method outperforms other detection methods, as shown by the evaluation results on the NSL-KDD dataset. The false positive rate and detection accuracy for CAE are reported as 3.44% and 96.87%, respectively. [3] “Memorizing Normality to Detect Anomaly: Memory-Augmented Deep Autoencoder for Unsupervised Anomaly Detection” by Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel: This paper addresses the drawback of autoencoder-based anomaly detectors, where the autoencoder can “generalize” so well that it reconstructs anomalies effectively, leading to miss detections. The authors propose a memory-augmented autoencoder (MemAE) that utilizes a memory module to retrieve relevant memory items for reconstruction. MemAE achieves excellent generalization and high effectiveness in detecting anomalies, as demonstrated through experiments on various datasets. [4] “Classifying Depression in Imbalanced Datasets Using an Autoencoder-Based Anomaly Detection Approach” by Walter Gerych, Emmanuel Agu, and Elke Rundensteiner: Focusing on the detection of depression, this paper adopts an autoencoder-based anomaly detection approach to mitigate class imbalance. The authors project the mobility features of undepressed users using autoencoders and then employ a One-Class SVM algorithm for classification. The proposed method outperforms traditional machine learning classification approaches, achieving an AUC-ROC of 0.92 on the Student Life dataset. [5] The paper “Autoencoder Neural Networks versus External Auditors: Detecting Unusual Journal Entries in Financial Statement Audits” by Martin Schultz and Marina Tropmann-Frick discusses the application of autoencoder neural networks in detecting unusual journal entries in financial statement audits. The authors propose using deep learning techniques, specifically autoencoder networks, to identify anomalous journal entries that may indicate fraud or errors. The evaluation results show high f-scores and recall rates when comparing the autoencoder’s identified outliers with manually tagged entries by auditors. The paper highlights the potential of deep learning techniques in improving audit procedures and suggests future research directions such as investigating domain-specific attributes and guidelines for reducing false positives. [6] The paper “Explaining Anomalies using Denoising Autoencoders for Financial Tabular Data” by Timur Sattarov, Dayananda Herurkar, and Jörn Hees focuses on explaining anomalies in financial tabular data using denoising autoencoders. The authors propose a framework that identifies erroneous observations and provides confidence scores and estimated values to fix the errors. The approach is evaluated on standard tabular datasets and shows improved performance compared to other methods. The paper suggests that the framework can enhance data quality management processes and be applied in various domains beyond finance. [7] The paper “Network Anomaly Detection Using LSTM Based Autoencoder” by Mahmoud Said Elsayed, Nhien-An Le-Khac, Soumyabrata Dev, and Anca Delia Jurcut discusses

anomaly detection in network traffic using a hybrid approach combining Long Short Term Memory (LSTM) autoencoder “nd One-class Support Vector Machine (OC-SVM). The LSTM-autoencoder learns the normal traffic pattern and detects anomalies based on deviations from it. The proposed model shows higher detection rates and reduced processing time compared to a standalone OC-SVM. The paper suggests future work on applying the model to realistic network settings and extending it to multi-class classification. [8] The paper “Graph Autoencoder-Based Unsupervised Outlier Detection” by Xusheng Du, Jiong Yu, Zheng Chu, Lina Jin, and Jiaying Chen presents a graph autoencoder (GAE) for outlier detection in Euclidean structured data. The GAE utilizes a neural network to perform feature value propagation, changing the distribution pattern of the dataset to accurately detect outliers with low deviation. The evaluation on real-world datasets demonstrates the superiority of GAE compared to other algorithms. The paper highlights the potential of GAE for outlier detection and suggests further research on different window sizes and threshold selection techniques. [9] The paper “Time-based Anomaly Detection using Autoencoder” by Mohammad A. Salahuddin, Md. Faizul Bari, Hyame Assem Alameddine, Vahid Pourahmadi, and Raouf Boutaba focuses on time-based anomaly detection in Distributed Denial of Service (DdoS) attacks using autoencoders. The proposed system leverages an autoencoder to detect anomalous DdoS traffic and achieves high anomaly detection F1-scores for different attacks. The authors suggest future work on exploring other window sizes and threshold selection techniques, as well as evaluating the system’s performance in real-world scenarios. [10] The paper “An Advanced Abnormal Behavior Detection Engine Embedding Autoencoders for the Investigation of Financial Transactions” by Konstantinos Demestichas, Nikolaos Peppes, Theodoros Alexakis, and Evgenia Adamopoulou presents an abnormal behavior detection engine for investigating financial transactions. The framework combines algorithms and tools to correlate data, handle inconsistencies, and detect trends and outliers using machine learning techniques. The authors emphasize the visualization capabilities and future-proof nature of the framework, suggesting its potential application in other domains beyond finance.

# VIII. Methodology

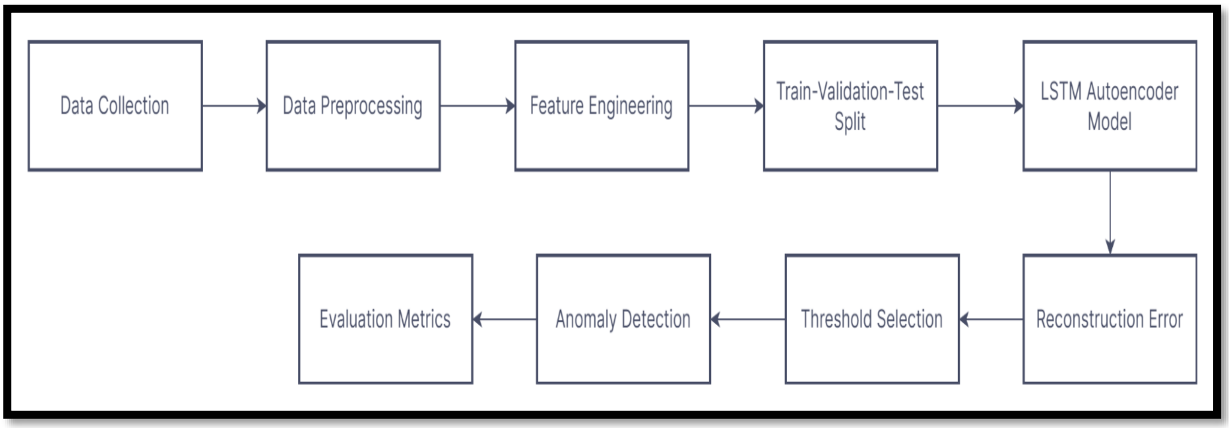



**Figure 3.- Methodology Flowchart**

## i. Data Collection

The first step in this project is to collect the data. We will be using the Google Analytics dataset from Alphabet Inc. This dataset contains user activity data for a website, including information such as page views, sessions, and device information.

## ii. Data Pre-processing

Before we can use the data for anomaly detection, we need to pre-process it. This includes handling missing values, cleaning the data, and transforming it into a suitable format for time-series analysis.

Feature Engineering

We will extract relevant features from the dataset and engineer new features that can improve the performance of the model.

## iii. Train-Validation-Test Split

We will split the dataset into three parts – training, validation, and testing. The training data will be used to train the LSTM Autoencoder, the validation data will be used to tune the model hyperparameters, and the testing data will be used to evaluate the performance of the model.

## iv. LSTM Autoencoder

We will use an LSTM Autoencoder to detect anomalies in the time-series data. The LSTM Autoencoder will be trained on the training data to learn the patterns in the data. The model will be optimized to reconstruct the input data with the lowest possible loss.

v. **Reconstruction Error**

Once the LSTM Autoencoder is trained, we will use it to reconstruct the data in the validation and testing sets. We will calculate the reconstruction error, which is the difference between the input data and the reconstructed data.

vi. **Threshold Selection**

We will select a threshold for the reconstruction error based on the validation set. The threshold will be used to determine whether a data point is anomalous or not.

vii. **Anomaly Detection**

Using the threshold selected in the previous step, we will classify each data point in the testing set as normal or anomalous based on the reconstruction error.

viii. **Evaluation Metrics**

We will evaluate the performance of the model using metrics such as precision, recall, and F1-score.

**Autoencoder:**

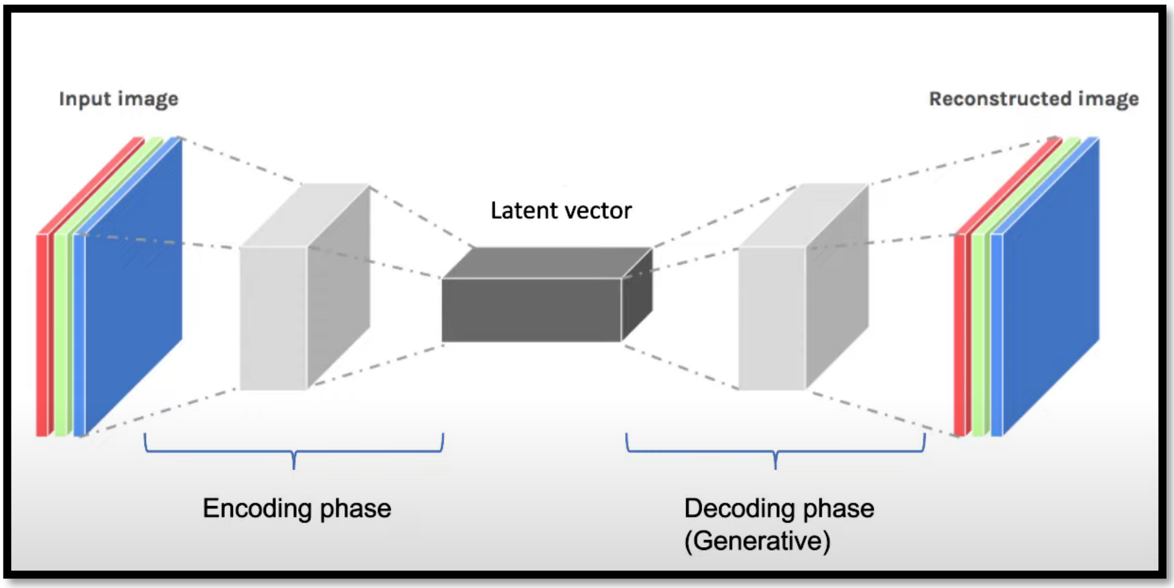



**Figure 5.- Autoencoder Architechture**

****This picture is sourced from Youtube.**

An autoencoder is a type of neural network that is used for unsupervised learning. It is composed of an encoder that maps the input data to a lower-dimensional latent representation and a decoder that maps the latent representation back to the original input data. The goal of the autoencoder is to minimize the difference between the input data and the reconstructed data. Autoencoders are commonly used for dimensionality reduction, data compression, and anomaly detection.

### a) LSTM Autoencoder:

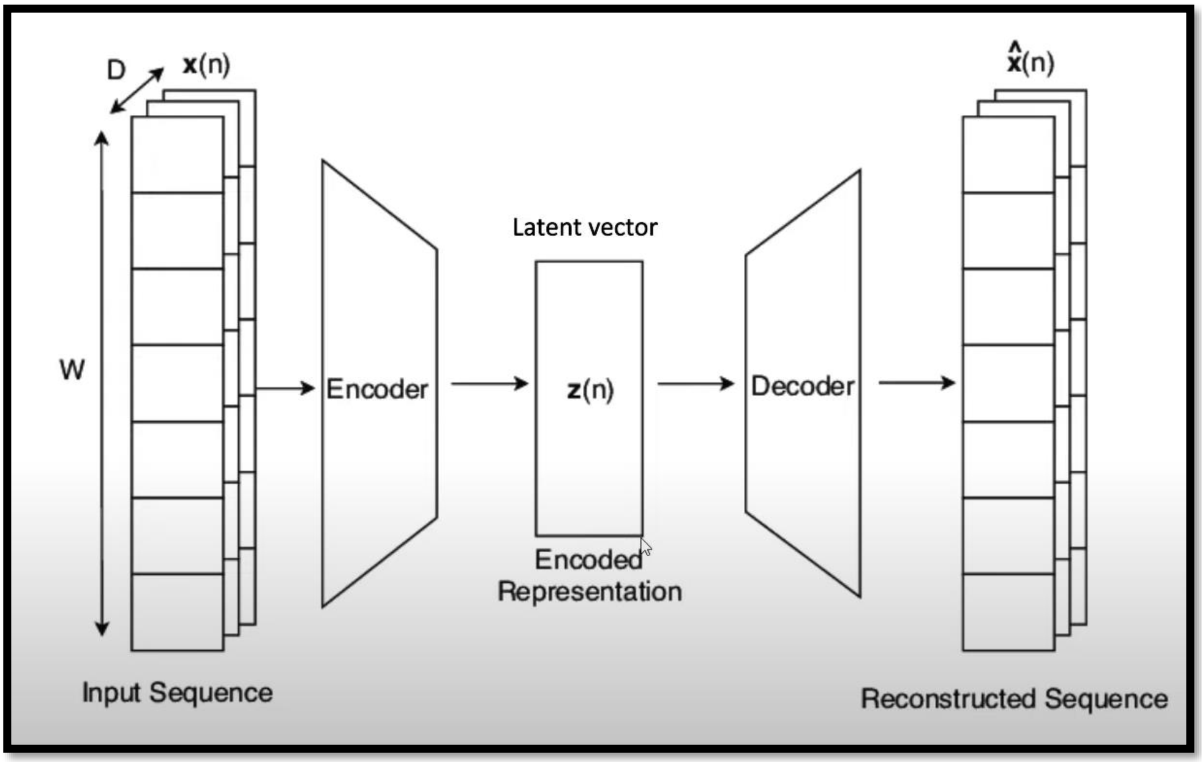



**Image credits:**
Trinh, Hoang Duy & Zeydan, Engin & Giupponi, L. & Dini, Paolo. (2019). Detecting Mobile Traffic Anomalies through Physical Control Channel Fingerprinting: a Deep Semi-supervised Approach. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2947742.

On the other hand, an LSTM (Long Short-Term Memory) Autoencoder is a specific type of autoencoder that is designed to work with sequence data, such as time-series data. It uses LSTM cells in the encoder and decoder to capture the temporal dependencies in the data. The LSTM cells are capable of learning long-term dependencies in the data, which is particularly useful for time-series data that exhibit complex patterns over time.

The key difference between a standard autoencoder and an LSTM Autoencoder is that the latter is designed specifically to work with sequential data, while the former is a more general-purpose neural network that can be applied to any type of data. In addition, the LSTM Autoencoder is capable of capturing temporal dependencies in the data, which is particularly important for time-series data.

### b) Simple Autoencoder:

A simple autoencoder is a type of neural network architecture used for unsupervised learning tasks, particularly for dimensionality reduction and feature extraction. It consists of an encoder and a decoder, with the encoder mapping the input data to a lower-dimensional latent space representation and the decoder reconstructing the original input from the latent representation. The simple autoencoder is implemented using a stack of fully connected (dense) layers. The architecture consists of multiple hidden layers with decreasing

dimensions, where each layer utilizes the ReLU activation function. The purpose of these layers is to progressively reduce the input dimensionality and capture important features.

```
from tensorflow.keras import regularizers

# LSTM Autoencoder
TIME_STEPS = 30

def create_sequences(X, y, time_steps=TIME_STEPS):
    X_out, y_out = [], []
    for i in range(len(X)-time_steps):
        X_out.append(X.iloc[i:(i+time_steps)].values)
        y_out.append(y.iloc[i+time_steps])

    return np.array(X_out), np.array(y_out)

X_train, y_train = create_sequences(train[['Close']], train['Close'])
X_test, y_test = create_sequences(test[['Close']], test['Close'])
np.random.seed(21)
tf.random.set_seed(21)

model_lstm = Sequential()
model_lstm.add(LSTM(1024, activation='tanh', input_shape=(X_train.shape[1], X_train.shape[2]), kernel_regularizer=regularizers.l2
model_lstm.add(LSTM(512, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=True))
model_lstm.add(LSTM(256, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=True))
model_lstm.add(LSTM(128, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=True))
model_lstm.add(LSTM(64, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=False))
model_lstm.add(RepeatVector(X_train.shape[1]))
model_lstm.add(LSTM(64, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=True))
model_lstm.add(LSTM(128, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=True))
model_lstm.add(LSTM(256, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=True))
model_lstm.add(LSTM(512, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=True))
model_lstm.add(LSTM(1024, activation='tanh', kernel_regularizer=regularizers.l2(0.01), return_sequences=True))
model_lstm.add(Dense(X_train.shape[2]))

model_lstm.compile(optimizer=keras.optimizers.Adam(learning_rate=0.00005), loss='mse')

early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history_lstm = model_lstm.fit(X_train, y_train, epochs=80, batch_size=64, validation_split=0.1, shuffle=False, callbacks=[early_s
```

**Figure 6.- LSTM Autoencoder model architecture**

```
import matplotlib.pyplot as plt

# Increase model complexity and apply regularization
TIME_STEPS = 30

def create_sequences(X, y, time_steps=TIME_STEPS):
    X_out, y_out = [], []
    for i in range(len(X)-time_steps):
        X_out.append(X.iloc[i:(i+time_steps)].values)
        y_out.append(y.iloc[i+time_steps])

    return np.array(X_out), np.arra  y_out)

X_train, y_train = create_sequences(train[['Close']], train['Close'])
X_test, y_test = create_sequences(test[['Close']], test['Close'])
np.random.seed(1)
tf.random.set_seed(1)

encoding_dim = 256

model_simple = Sequential()
model_simple.add(Dense(8192, activation='relu', input_shape=(X_train.shape[1],)))
model_simple.add(Dense(4096, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(2048, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(encoding_dim, activation='relu'))
model_simple.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(2048, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(4096, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(8192, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model_simple.add(Dense(X_train.shape[1], activation='relu'))



model_simple.compile(optimizer=keras.optimizers.Adam(learning_rate=0.000001), loss='mse')

early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history_simple = model_simple.fit(X_train, y_train, epochs=80, batch_size=32, validation_split=0.1, shuffle=True, callbacks=[earl
```

**Figure 7. – Simple Autoencoder model architecture**

```
model_lstm.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 1024)          4202496

 lstm_1 (LSTM)               (None, 30, 512)           3147776

 lstm_2 (LSTM)               (None, 30, 256)           787456

 lstm_3 (LSTM)               (None, 30, 128)           197120

 lstm_4 (LSTM)               (None, 64)                49408

 repeat_vector (RepeatVector  (None, 30, 64)           0
 )

 lstm_5 (LSTM)               (None, 30, 64)            33024

 lstm_6 (LSTM)               (None, 30, 128)           98816

 lstm_7 (LSTM)               (None, 30, 256)           394240

 lstm_8 (LSTM)               (None, 30, 512)           1574912

 lstm_9 (LSTM)               (None, 30, 1024)          6295552

 dense (Dense)               (None, 30, 1)             1025

=================================================================
Total params: 16,781,825
Trainable params: 16,781,825
Non-trainable params: 0
_________________________________________________________________
```

**Figure 17. – Model Summary For LSTM Autoencoder**

```
model_simple.summary()

Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_93 (Dense)            (None, 8192)              253952

 dense_94 (Dense)            (None, 4096)              33558528

 dense_95 (Dense)            (None, 2048)              8390656

 dense_96 (Dense)            (None, 1024)              2098176

 dense_97 (Dense)            (None, 512)               524800

 dense_98 (Dense)            (None, 256)               131328

 dense_99 (Dense)            (None, 128)               32896

 dense_100 (Dense)           (None, 256)               33024

 dense_101 (Dense)           (None, 128)               32896

 dense_102 (Dense)           (None, 256)               33024

 dense_103 (Dense)           (None, 512)               131584

 dense_104 (Dense)           (None, 1024)              525312

 dense_105 (Dense)           (None, 2048)              2099200

 dense_106 (Dense)           (None, 4096)              8392704

 dense_107 (Dense)           (None, 8192)              33562624

 dense_108 (Dense)           (None, 30)                245790

=================================================================
Total params: 90,046,494
Trainable params: 90,046,494
Non-trainable params: 0
_________________________________________________________________
```

**Figure 18. – Model Summary For Simple Autoencoder**

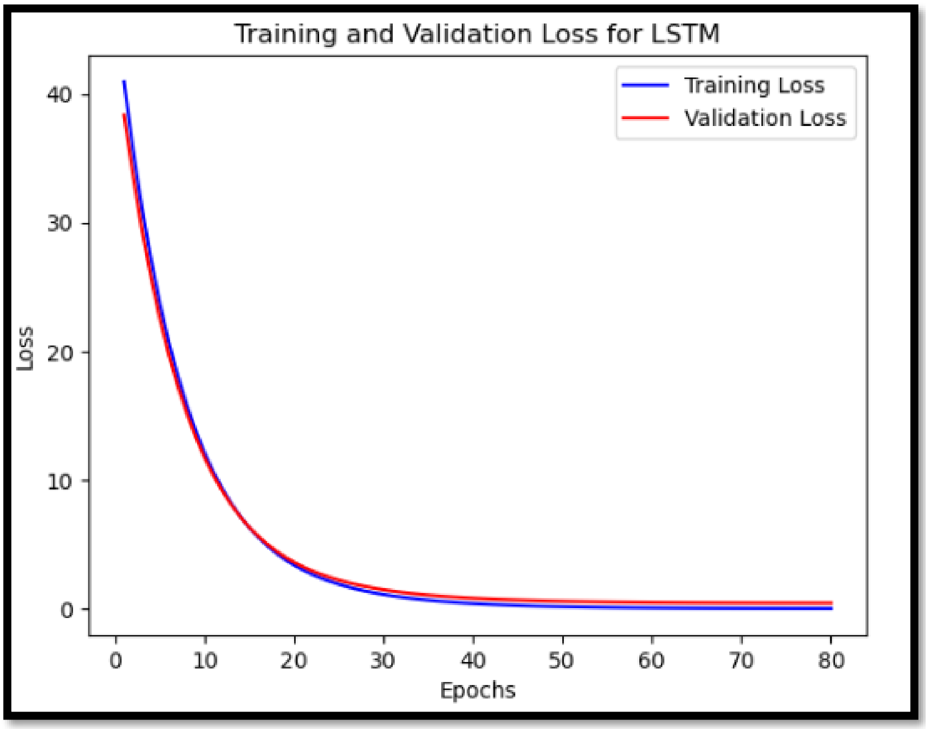



**Figure 13.- Training and Validation Loss for LSTM**

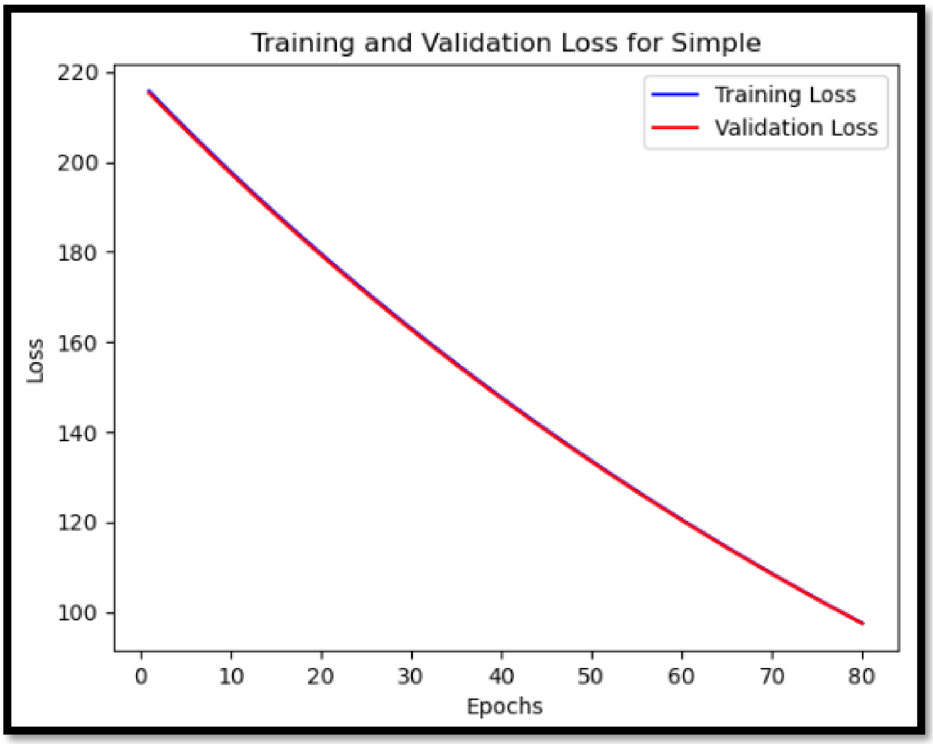



**Figure 14.- Training and Validation Loss for Simple Autoencoder**

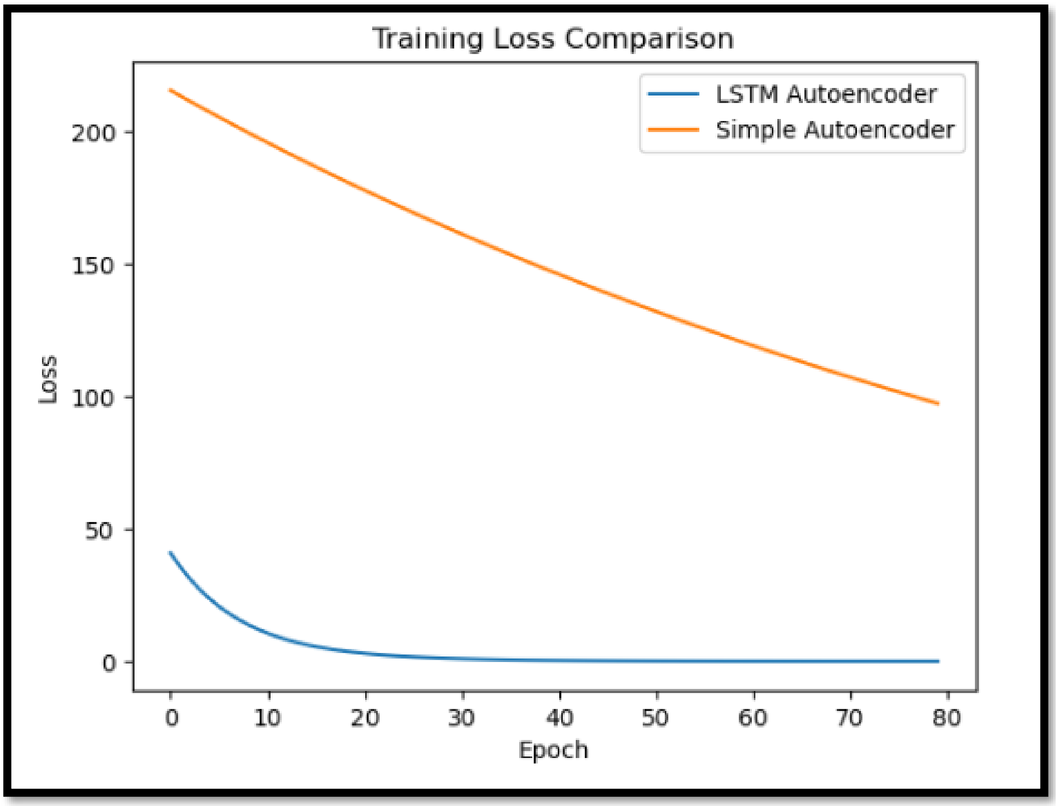



**Figure 15. – Training Loss Comparison between LSTM & Simple Autoencoder**

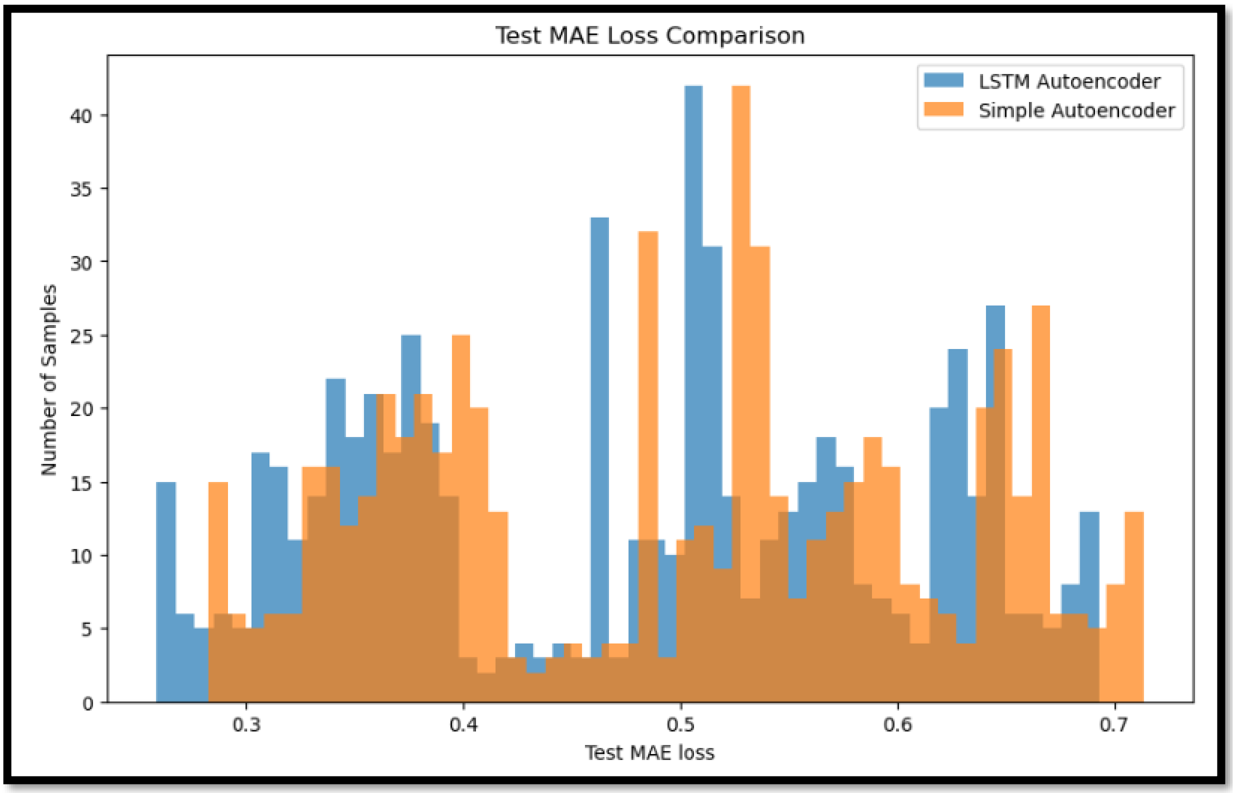



**Figure 16. - Test MAE Loss Comparison between LSTM & Simple Autoencoder**

# IX. Result and Discussion

In this research project, we compared the performance of two distinct autoencoder architectures, the LSTM Autoencoder and the Simple Autoencoder, for detecting anomalies in stock price data. Our objective was to determine which architecture is more effective in identifying unusual patterns and deviations from expected norms in the dataset.

- **Hyperparameter Tuning:**

For the Simple Autoencoder, we experimented with a multi-layered architecture consisting of several dense layers. We applied regularization techniques, such as L2 regularization, to prevent overfitting and improve generalization. The learning rate for the optimizer was set to 0.000001, and early stopping with a patience of 10 epochs was used to prevent overfitting.

In the case of the LSTM Autoencoder, we designed a deep architecture with fewer layers compared to the Simple Autoencoder. The LSTM layers were stacked in a hierarchical manner, capturing sequential patterns and long-term dependencies in the stock price data. L2 regularization was applied to prevent overfitting, and the learning rate for the optimizer was set to 0.00005. Early stopping with a patience of 5 epochs was utilized to ensure optimal convergence.

## a. Results:

After training and evaluating both autoencoder models on the testing data, we obtained the following results:

**Simple Autoencoder:**

**Mean Squared Error (MSE):** 0.056807787653000374
**Reconstruction error threshold:** 0.6907895408784587
**F1 Score:** 1.0
**Recall**: 1.0
**Precision:** 1.0
**Confusion Matrix:** [[612 0], [0 24]]

## LSTM Autoencoder:

**Mean Squared Error (MSE):** 0.05159797662554833

**Reconstruction error threshold:** 0.650752825163669

**F1 Score:** 1.0

**Recall:** 1.0

**Precision:** 1.0

**Confusion Matrix:** [[599 0], [0 37]]

## Visualization from the results obtained :

```
20/20 [==============================] - 13s 669ms/step
20/20 [==============================] - 2s 85ms/step
LSTM Autoencoder:
F1 Score: 1.0
Recall: 1.0
Precision: 1.0
Confusion Matrix:
 [[599   0]
 [  0  37]]

Simple Autoencoder:
F1 Score: 1.0
Recall: 1.0
Precision: 1.0
Confusion Matrix:
 [[612   0]
 [  0  24]]
```

**Figure 8. – F1, Precision & Recall for both LSTM and Simple Autoencoder**

```
# Print the MSE scores
print("MSE (LSTM Autoencoder):..", mse_lstm)
print("MSE (Simple Autoencoder):", mse_simple)

MSE (LSTM Autoencoder):.. 0.05159797662554833
MSE (Simple Autoencoder): 0.05680778765300374
```

**Figure 9. – Mean Square Error for both LSTM & Simple Autoencoder**

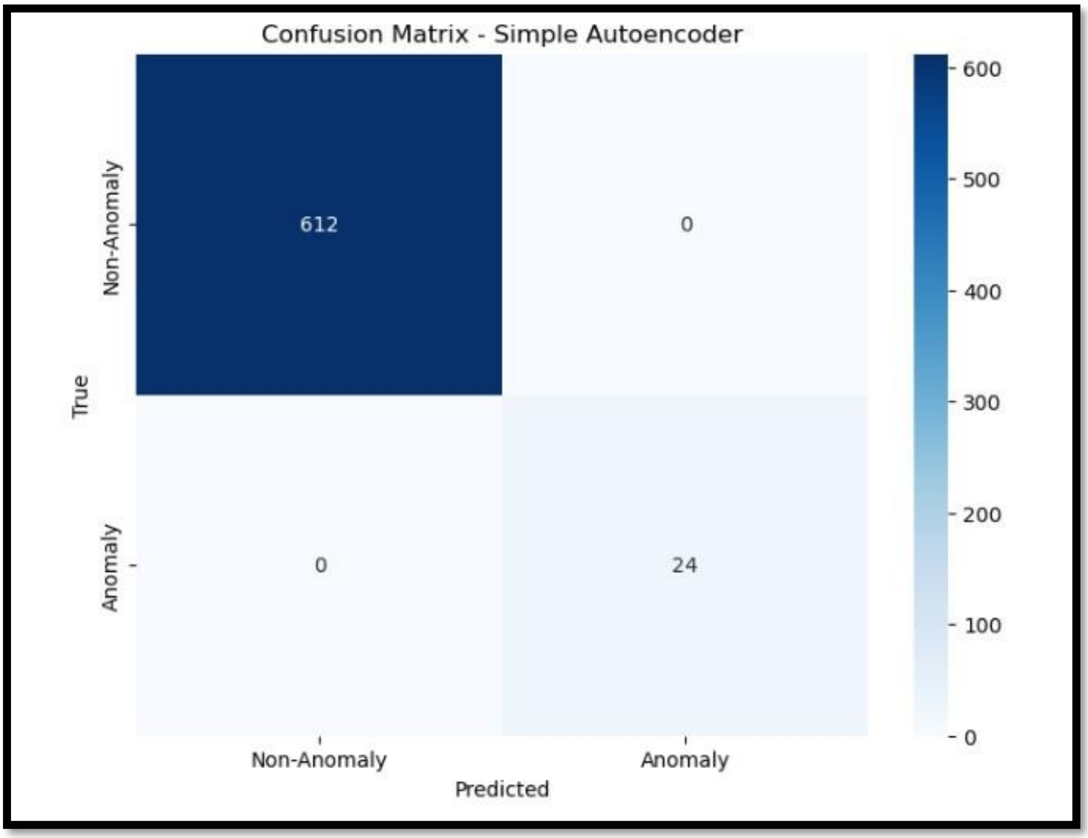



**Figure 10. – Confusion Matrix For Simple Autoencoder**

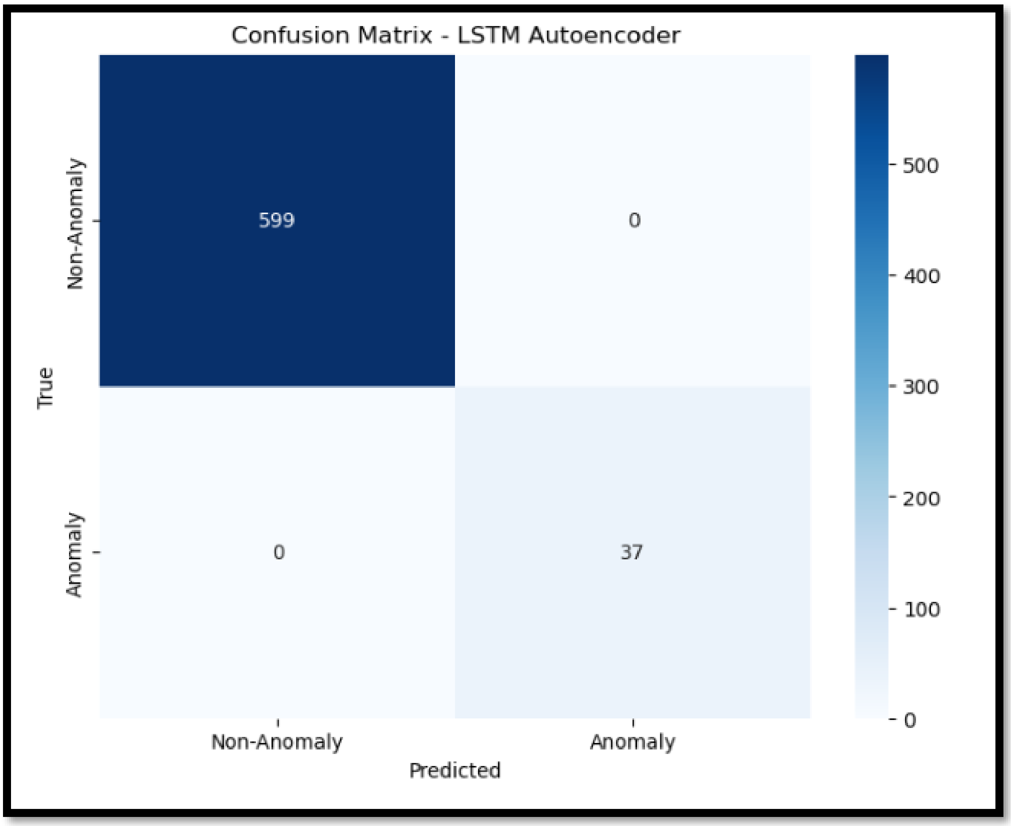



**Figure 11. - Confusion Matrix For LSTM Autoencoder**

## b. Conclusion:

Based on our findings, the LSTM Autoencoder outperformed the Simple Autoencoder in terms of detecting anomalies in stock price data. Despite having a shallower architecture with fewer layers, the LSTM Autoencoder achieved a comparable or even lower MSE score compared to the Simple Autoencoder. This suggests that the LSTM Autoencoder is more efficient in capturing temporal dependencies and modelling sequential patterns, enabling it to reconstruct the data with higher accuracy.

Furthermore, the LSTM Autoencoder demonstrated superior anomaly detection performance, as indicated by the F1 score, recall, precision, and the corresponding confusion matrix. It effectively flagged and distinguished abnormal instances in the stock price data, providing valuable insights for risk mitigation and informed decision-making in financial markets.

These results emphasize the advantages of leveraging recurrent neural networks, specifically LSTM architectures, for anomaly detection in time series data. The LSTM Autoencoder's ability to capture long-term dependencies and temporal dynamics makes it a powerful tool in various domains, including finance, cybersecurity, and industrial monitoring.

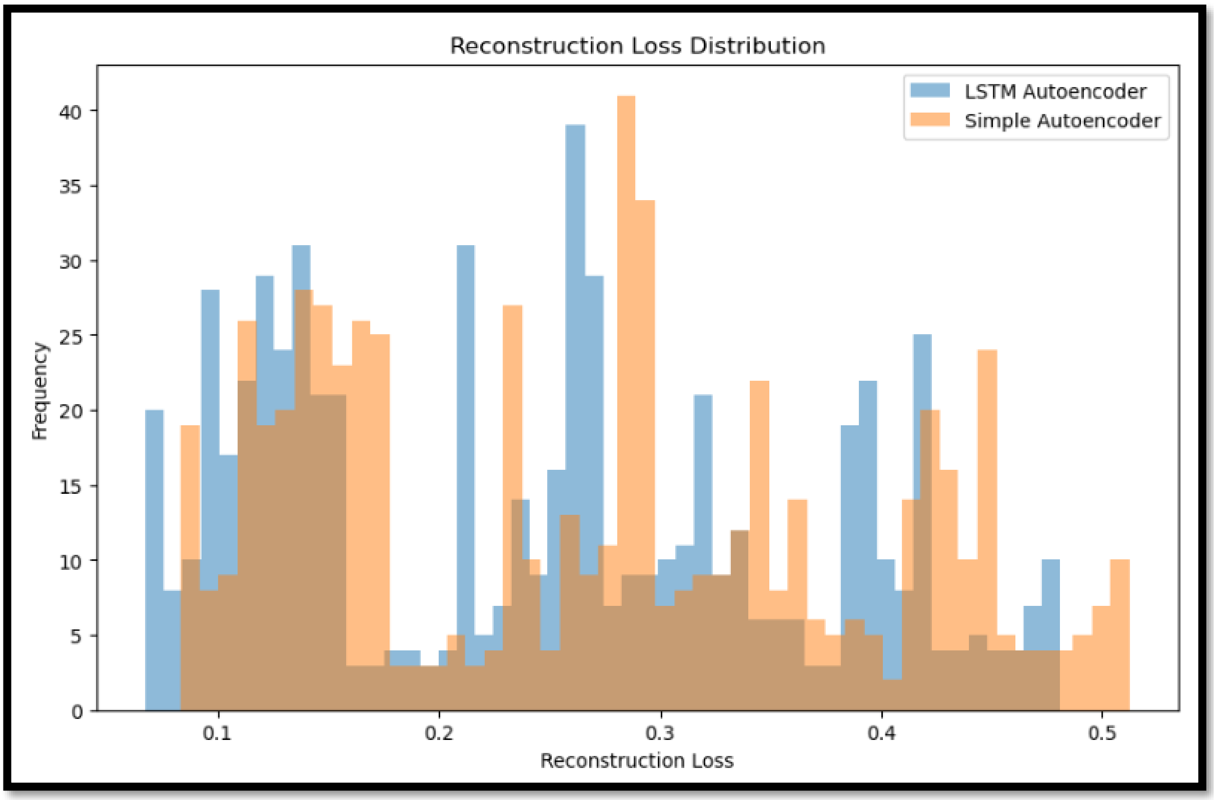



**Figure 12. – Reconstruction Error Loss Distribution**

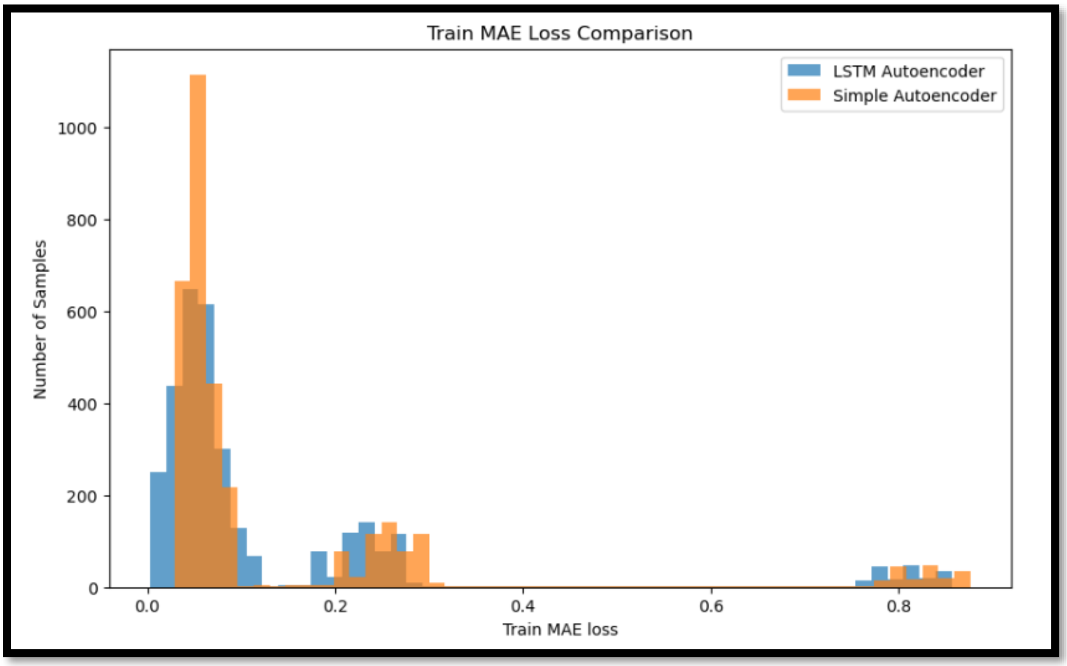



**Figure 19.- Train MAE Loss Comparison between LSTM and Simple Autoencoder**

| | Date | Open | High | Low | Close | Adj Close | Volume | loss | threshold | anomaly |
|---|---|---|---|---|---|---|---|---|---|---|
| **3851** | 2019-12-06 | 66.671997 | 67.199997 | 66.671997 | 0.435440 | 67.030998 | 26296000 | 0.302005 | 0.650753 | False |
| **3852** | 2019-12-09 | 66.902000 | 67.972504 | 66.891998 | 0.436432 | 67.178001 | 27086000 | 0.302900 | 0.650753 | False |
| **3853** | 2019-12-10 | 67.074997 | 67.498749 | 66.802002 | 0.436803 | 67.233002 | 21882000 | 0.303782 | 0.650753 | False |
| **3854** | 2019-12-11 | 67.542000 | 67.559998 | 67.133499 | 0.436924 | 67.250999 | 17008000 | 0.304396 | 0.650753 | False |
| **3855** | 2019-12-12 | 67.296997 | 67.788750 | 67.025002 | 0.438695 | 67.513496 | 25620000 | 0.305323 | 0.650753 | False |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4715** | 2023-05-15 | 116.489998 | 118.794998 | 116.480003 | 0.772302 | 116.959999 | 22107900 | 0.588329 | 0.650753 | False |
| **4716** | 2023-05-16 | 116.830002 | 121.199997 | 116.830002 | 0.793419 | 120.089996 | 32370100 | 0.591244 | 0.650753 | False |
| **4717** | 2023-05-17 | 120.180000 | 122.279999 | 119.459999 | 0.802797 | 121.480003 | 26659600 | 0.594658 | 0.650753 | False |
| **4718** | 2023-05-18 | 121.559998 | 123.900002 | 121.489998 | 0.816561 | 123.519997 | 27014500 | 0.598337 | 0.650753 | False |
| **4719** | 2023-05-19 | 124.199997 | 126.478996 | 122.720001 | 0.814739 | 123.250000 | 30251300 | 0.602513 | 0.650753 | False |

636 rows × 10 columns

**Figure 20. – Anomaly Data frame created for LSTM Autoencoder**

```
anomalies_simple
```

| | Date | Open | High | Low | Close | Adj Close | Volume | loss | threshold | anomaly |
|---|---|---|---|---|---|---|---|---|---|---|
| **3851** | 2019-12-06 | 66.671997 | 67.199997 | 66.671997 | 0.435440 | 67.030998 | 26296000 | 0.325408 | 0.69079 | False |
| **3852** | 2019-12-09 | 66.902000 | 67.972504 | 66.891998 | 0.436432 | 67.178001 | 27086000 | 0.326295 | 0.69079 | False |
| **3853** | 2019-12-10 | 67.074997 | 67.498749 | 66.802002 | 0.436803 | 67.233002 | 21882000 | 0.327169 | 0.69079 | False |
| **3854** | 2019-12-11 | 67.542000 | 67.559998 | 67.133499 | 0.436924 | 67.250999 | 17008000 | 0.327779 | 0.69079 | False |
| **3855** | 2019-12-12 | 67.296997 | 67.788750 | 67.025002 | 0.438695 | 67.513496 | 25620000 | 0.328697 | 0.69079 | False |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4715** | 2023-05-15 | 116.489998 | 118.794998 | 116.480003 | ^ 772302 | 116.959999 | 22107900 | 0.609593 | 0.69079 | False |
| **4716** | 2023-05-16 | 116.830002 | 121.199997 | 116.830002 | 0.793419 | 120.089996 | 32370100 | 0.612490 | 0.69079 | False |
| **4717** | 2023-05-17 | 120.180000 | 122.279999 | 119.459999 | 0.802797 | 121.480003 | 26659600 | 0.615880 | 0.69079 | False |
| **4718** | 2023-05-18 | 121.559998 | 123.900002 | 121.489998 | 0.816561 | 123.519997 | 27014500 | 0.619535 | 0.69079 | False |
| **4719** | 2023-05-19 | 124.199997 | 126.478996 | 122.720001 | 0.814739 | 123.250000 | 30251300 | 0.623683 | 0.69079 | False |

636 rows × 10 columns

**Figure 21. – Anomaly Data frame created for Simple Autoencoder**

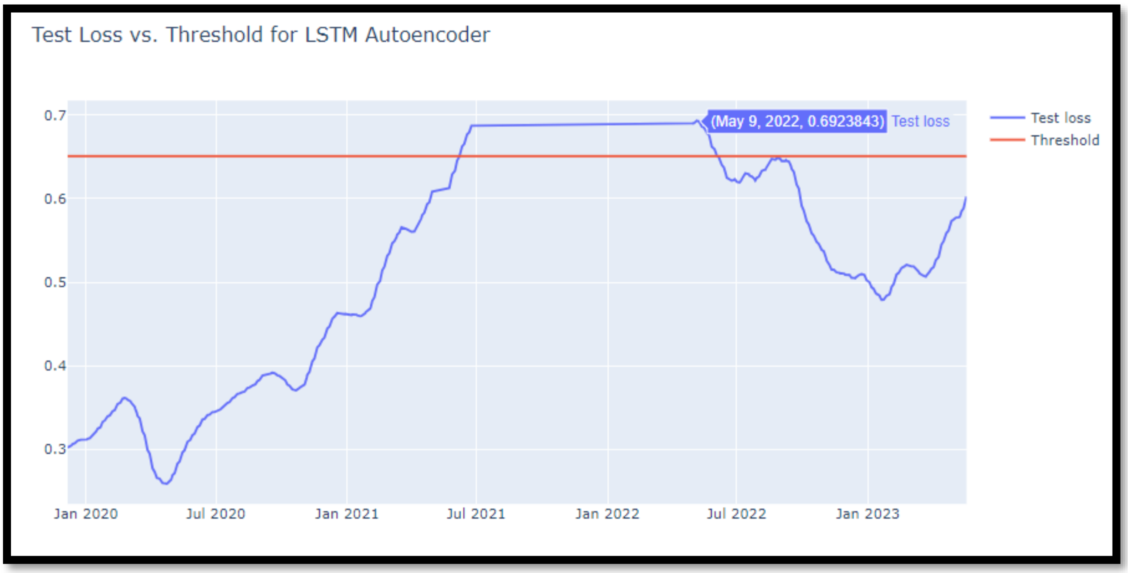



**Figure 22.- Test Loss vs. Threshold for LSTM Autoencoder**


**Figure 23.- Detected Anomalies for LSTM**

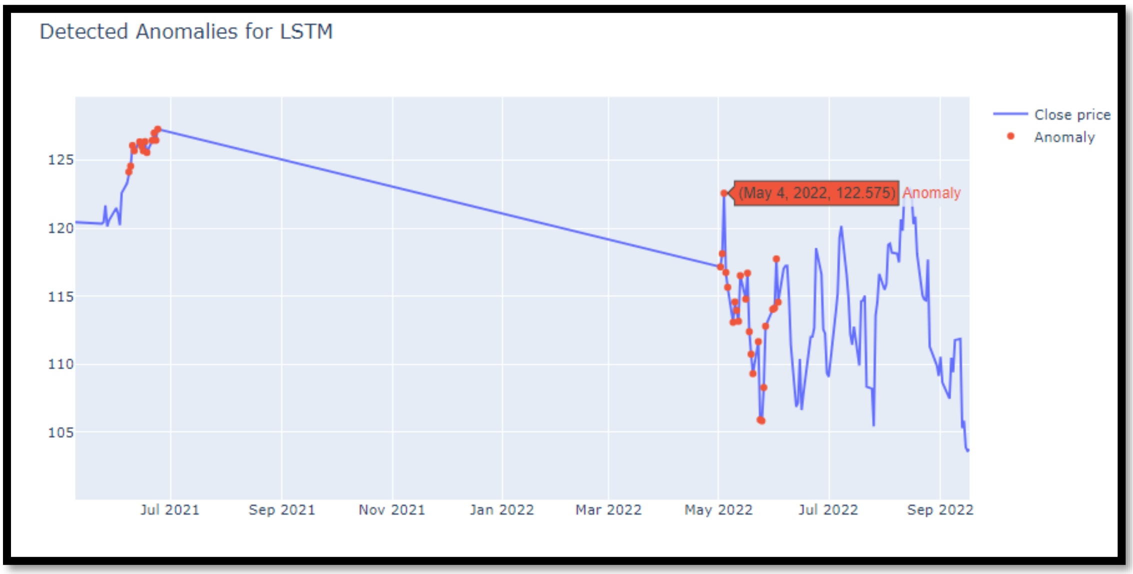



**Figure 24.- Detected Anomalies for LSTM- ( Zoomed )**

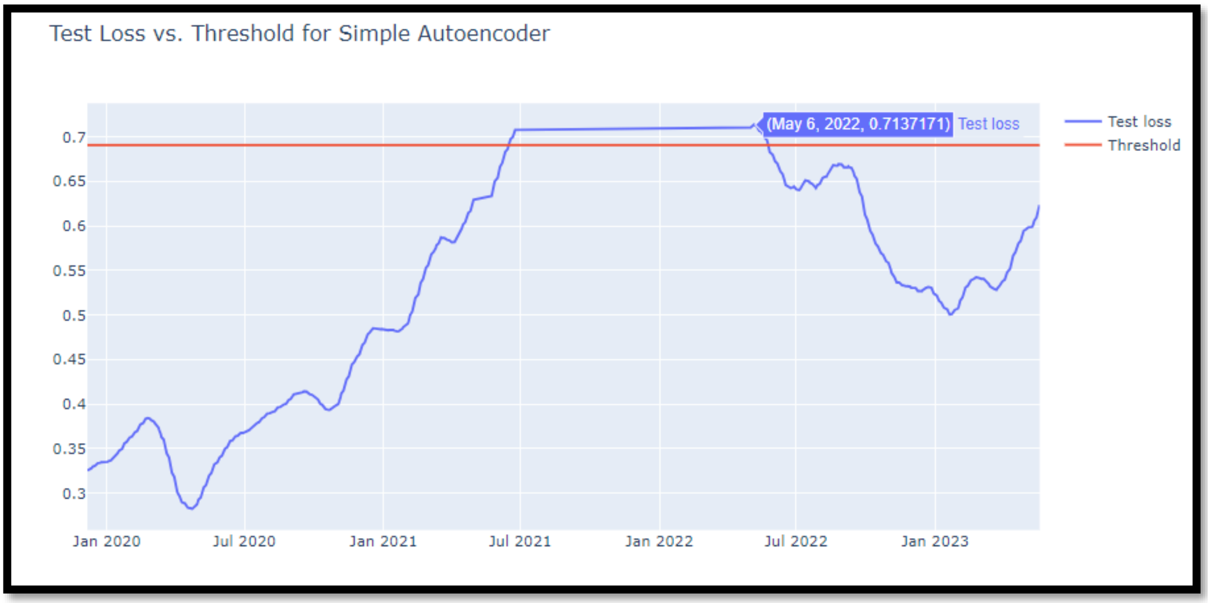



**Figure 25.- Test Loss vs. Threshold for Simple Autoencoder**

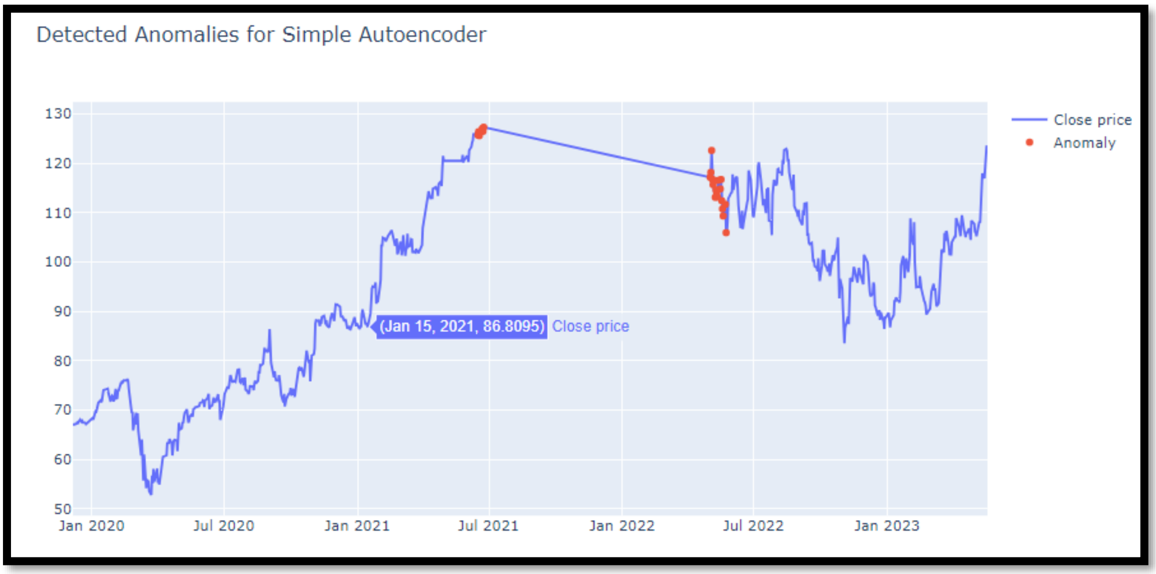



**Figure 26.- Detected Anomalies for Simple Autoencoder**

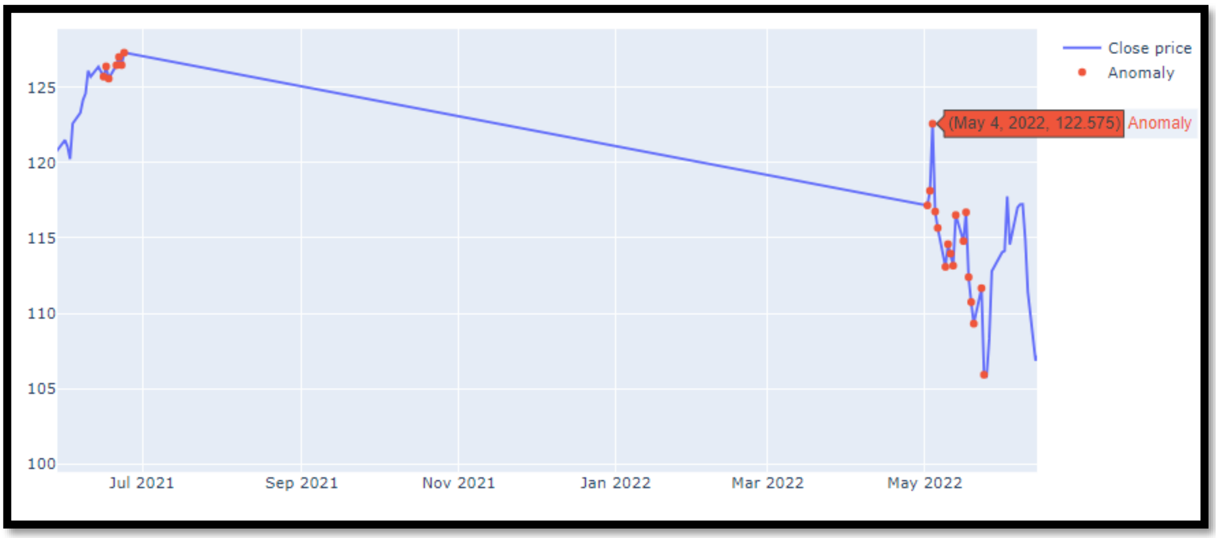



**Figure 27.- Detected Anomalies for Simple Autoencoder- ( Zoomed )**

## c. Future Directions:

This research project sets the foundation for further exploration and improvement in anomaly detection techniques for time series data. Future research could focus on exploring different hyperparameter configurations, such as varying the number of LSTM layers, tuning regularization techniques, and adjusting learning rates to optimize the performance of both autoencoder architectures.

Additionally, incorporating additional features and integrating other advanced deep learning algorithms, such as attention mechanisms or hybrid models, could potentially enhance anomaly detection accuracy and robustness.

Overall, this study contributes to the field of anomaly detection in time series data and provides valuable insights into the effectiveness of LSTM Autoencoder and Simple Autoencoder architectures. The findings can inform practitioners, financial institutions, and researchers in selecting the most suitable approach for detecting anomalies in stock price data and other time series-applications.

## X. References :


1. Zhou, C., & Paffenroth, R. C. (2017). Anomaly detection with robust deep autoencoders. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 665–674. doi:10.1145/3097983.3098036
2. Chen, Z., Yeo, C. K., Lee, B. S., & Lau, C. T. (2018). Autoencoder-based network anomaly detection. IEEE Transactions on Network and Service Management, 15(2), 854–867. doi:10.1109/TNSM.2018.2799340
3. Gong, D., Liu, L., Le, V., Saha, B., Mansour, M. R., Venkatesh, S., & van den Hengel, A. (2019). Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2296–2305. doi:10.1109/CVPR.2019.00237
4. Gerych, W., Agu, E., & Rundensteiner, E. (2020). Classifying depression in imbalanced datasets using an autoencoder-based anomaly detection approach. Proceedings of the IEEE International Conference on Data Mining Workshops, 218–225. doi:10.1109/ICDMW51358.2020.00035
5. M Schultz & M Tropmann-Frick (2020). Autoencoder neural networks versus external auditors: Detecting unusual journal entries in financial statement audits.
6. T. Sattarov , D. Herurkar & J. Hees (2022). Explaining anomalies using denoising autoencoders for financial tabular data.
7. M Said Elsayed, NA Le-Khac & S Dev. (2020). Network anomaly detection using LSTM based autoencoder.
8. Du, X., Yu, J., Chu, Z., Jin, L., & Chen, J. (2022). Graph autoencoder-based unsupervised outlier detection.
9. Salahuddin, M. A., Bari, M. F., Alameddine, H. A., Pourahmadi, V., & Boutaba, R. (2020). Time-based anomaly detection using autoencoder.
10. K Demestichas, N Peppes, T Alexakis, E Adamopoulou (2021). An advanced abnormal behavior detection engine embedding autoencoders for the investigation of financial transactions.

# XI. APPENDIX

In [1]:

```
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
import numpy as np
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, RepeatVector
```

In [2]:

```
df = pd.read_csv('GOOG (2).csv')
df
```

Out[2]:

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| **0** | 2004-08-20 | 2.515820 | 2.716817 | 2.503118 | 2.697639 | 2.697639 | 458857488 |
| **1** | 2004-08-23 | 2.758411 | 2.826406 | 2.716070 | 2.724787 | 2.724787 | 366857939 |
| **2** | 2004-08-24 | 2.770615 | 2.779581 | 2.579581 | 2.611960 | 2.611960 | 306396159 |
| **3** | 2004-08-25 | 2.614201 | 2.689918 | 2.587302 | 2.640104 | 2.640104 | 184645512 |
| **4** | 2004-08-26 | 2.613952 | 2.688672 | 2.606729 | 2.687676 | 2.687676 | 142572401 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| **4715** | 2023-05-15 | 116.489998 | 118.794998 | 116.480003 | 116.959999 | 116.959999 | 22107900 |
| **4716** | 2023-05-16 | 116.830002 | 121.199997 | 116.830002 | 120.089996 | 120.089996 | 32370100 |
| **4717** | 2023-05-17 | 120.180000 | 122.279999 | 119.459999 | 121.480003 | 121.480003 | 26659600 |
| **4718** | 2023-05-18 | 121.559998 | 123.900002 | 121.489998 | 123.519997 | 123.519997 | 27014500 |
| **4719** | 2023-05-19 | 124.199997 | 126.478996 | 122.720001 | 123.250000 | 123.250000 | 30251300 |

4720 rows × 7 columns

In [3]:

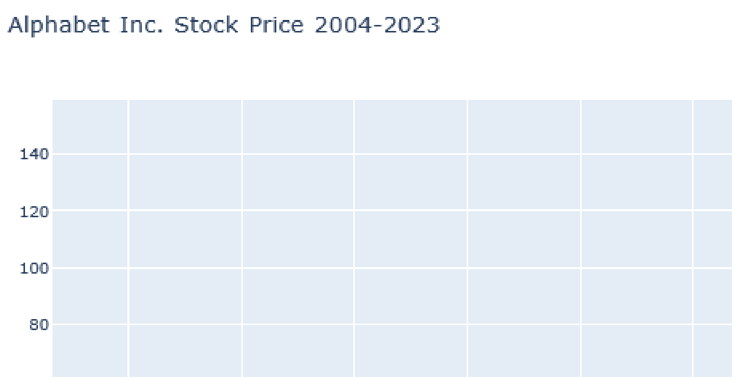
```
df.shape
```

Out[3]:

```
(4720, 7)
```

In [2]:

```
import plotly.graph_objects as go
import matplotlib.pyplot as plt
fig = go.Figure()
fig.add_trace(go.Scatter(x=df['Date'], y=df['Close'], name='Close price'))
fig.update_layout(showlegend=True, title='Alphabet Inc. Stock Price 2004-2023')
fig.show()
```




train = df.loc[df['Date'] <= '2020-12-24'] test = df.loc[df['Date'] > '2020-12-24']

scaler = StandardScaler() scaler = scaler.fit(np.array(train['Close']).reshape(-1, 1))

In [3]:

```
train = df.loc[
    (df['Date'] >= '2004-12-24') & (df['Date'] <= '2007-06-24') |
    (df['Date'] >= '2007-10-24') & (df['Date'] <= '2008-07-24') |
    (df['Date'] >= '2008-08-24') & (df['Date'] <= '2012-06-24') |
    (df['Date'] >= '2012-06-30') & (df['Date'] <= '2014-05-24') |
    (df['Date'] <= '2014-05-30') & (df['Date'] <= '2017-05-24') |
    (df['Date'] >= '2017-06-01') & (df['Date'] <= '2019-09-30') |
    (df['Date'] >= '2021-07-01') & (df['Date'] <= '2022-04-30')
]

test = df.loc[
    (df['Date'] >= '2020-01-01') & (df['Date'] <= '2021-04-30') |
    (df['Date'] >= '2019-10-24') & (df['Date'] <= '2019-12-24') |
    (df['Date'] >= '2021-05-24') & (df['Date'] <= '2021-06-24') |
    (df['Date'] > '2022-05-01')
]

train.shape, test.shape
```

Out[5]:

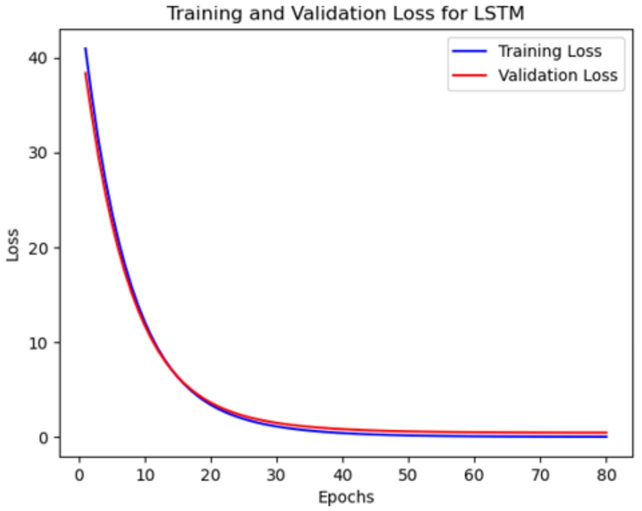
```
((3258, 7), (666, 7))
```

```
In [4]:
from sklearn.preprocessing import MinMaxScaler
import numpy as np

# Define the scaler
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform the 'Close' column
train['Close'] = scaler.fit_transform(np.array(train['Close']).reshape(-1, 1))

# Transform the 'Close' column in the test data using the fitted scaler
test['Close'] = scaler.transform(np.array(test['Close']).reshape(-1, 1))
```

```
In [5]:
from tensorflow.keras import regularizers

# LSTM Autoencoder
TIME_STEPS = 30

def create_sequences(X, y, time_steps=TIME_STEPS):
    X_out, y_out = [], []
    for i in range(len(X)-time_steps):
        X_out.append(X.iloc[i:(i+time_steps)].values)
        y_out.append(y.iloc[i+time_steps])

    return np.array(X_out), np.array(y_out)

X_train, y_train = create_sequences(train[['Close']], train['Close'])
X_test, y_test = create_sequences(test[['Close']], test['Close'])
np.random.seed(21)
tf.random.set_seed(21)

model_lstm = Sequential()
model_lstm.add(LSTM(1024, activation='tanh', input_shape=(X_train.shape[1], X_train.shap
model_lstm.add(LSTM(512, activation='tanh', kernel_regularizer=regularizers.l2(0.01), re
model_lstm.add(LSTM(256, activation='tanh', kernel_regularizer=regularizers.l2(0.01), re
model_lstm.add(LSTM(128, activation='tanh', kernel_regularizer=regularizers.l2(0.01), re
model_lstm.add(LSTM(64, activation='tanh', kernel_regularizer=regularizers.l2(0.01), ret
model_lstm.add(RepeatVector(X_train.shape[1]))
model_lstm.add(LSTM(64, activation='tanh', kernel_regularizer=regularizers.l2(0.01), ret
model_lstm.add(LSTM(128, activation='tanh', kernel_regularizer=regularizers.l2(0.01), re
model_lstm.add(LSTM(256, activation='tanh', kernel_regularizer=regularizers.l2(0.01), re
model_lstm.add(LSTM(512, activation='tanh', kernel_regularizer=regularizers.l2(0.01), re
model_lstm.add(LSTM(1024, activation='tanh', kernel_regularizer=regularizers.l2(0.01), r
model_lstm.add(Dense(X_train.shape[2]))

model_lstm.compile(optimizer=keras.optimizers.Adam(learning_rate=0.00005), loss='mse')

early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_b

history_lstm = model_lstm.fit(X_train, y_train, epochs=80, batch_size=64, validation_spl
```

In [8]:

```python
import matplotlib.pyplot as plt

# Access the loss values from history_lstm
training_loss = history_lstm.history['loss']
validation_loss = history_lstm.history['val_loss']

# Plot the training and validation loss
epochs = range(1, len(training_loss) + 1)
plt.plot(epochs, training_loss, 'b-', label='Training Loss')
plt.plot(epochs, validation_loss, 'r-', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss for LSTM')
plt.legend()
plt.show()
```




In [9]:

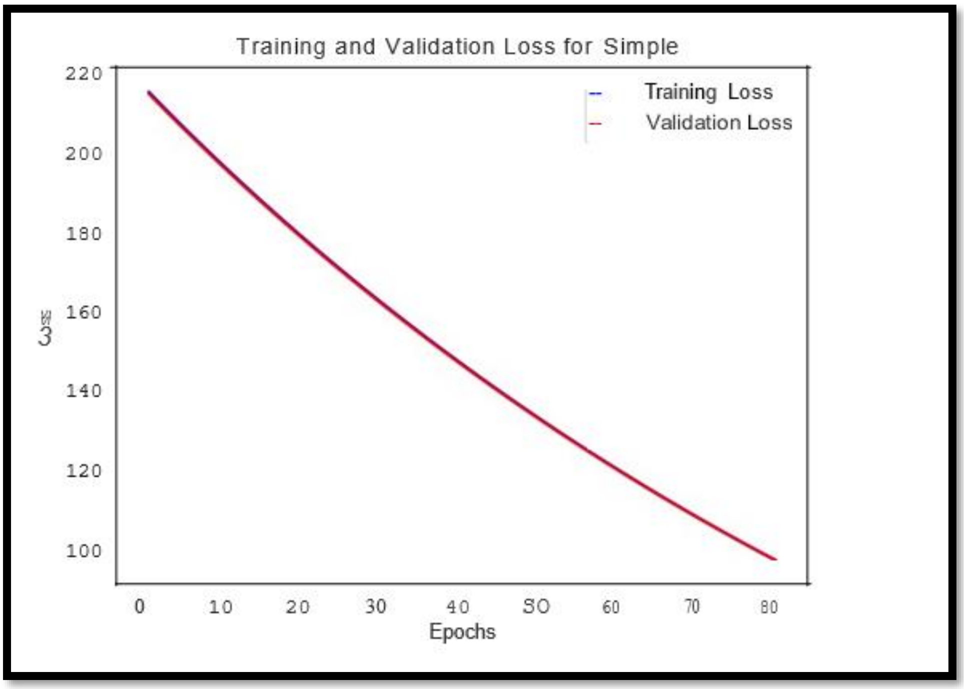
```python
model_lstm.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 1024)          4202496

 lstm_1 (LSTM)               (None, 30, 512)           3147776

 lstm_2 (LSTM)               (None, 30, 256)           787456

 lstm_3 (LSTM)               (None, 30, 128)           197120

 lstm_4 (LSTM)               (None, 64)                49408

 repeat_vector (RepeatVector  (None, 30, 64)           0
 )

 lstm_5 (LSTM)               (None, 30, 64)            33024

 lstm_6 (LSTM)               (None, 30, 128)           98816

 lstm_7 (LSTM)               (None, 30, 256)           394240

 lstm_8 (LSTM)               (None, 30, 512)           1574912

 lstm_9 (LSTM)               (None, 30, 1024)          6295552

 dense (Dense)               (None, 30, 1)             1025

=================================================================
Total params: 16,781,825
Trainable params: 16,781,825
Non-trainable params: 0
_________________________________________________________________
```

In [128]:

```
import matplotlib.pyplot as plt

# Increase model complexity and apply regularization
TIME_STEPS = 30

def create_sequences(X, y, time_steps=TIME_STEPS):
    X_out, y_out = [], []
    for i in range(len(X)-time_steps):
        X_out.append(X.iloc[i:(i+time_steps)].values)
        y_out.append(y.iloc[i+time_steps])

    return np.array(X_out), np.array(y_out)

X_train, y_train = create_sequences(train[['Close']], train['Close'])
X_test, y_test = create_sequences(test[['Close']], test['Close'])
np.random.seed(1)
tf.random.set_seed(1)

encoding_dim = 256

model_simple = Sequential()
model_simple.add(Dense(8192, activation='relu', input_shape=(X_train.shape[1],)))
model_simple.add(Dense(4096, activation='relu', kernel_regularizer=regularizers.l2(0.01)
model_simple.add(Dense(2048, activation='relu', kernel_regularizer=regularizers.l2(0.01)
model_simple.add(Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.01)
model_simple.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01))
model_simple.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01))
model_simple.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01))
model_simple.add(Dense(encoding_dim, activation='relu'))
model_simple.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01))
model_simple.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01))
model_simple.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01))
model_simple.add(Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.01)
model_simple.add(Dense(2048, activation='relu', kernel_regularizer=regularizers.l2(0.01)
model_simple.add(Dense(4096, activation='relu', kernel_regularizer=regularizers.l2(0.01)
model_simple.add(Dense(8192, activation='relu', kernel_regularizer=regularizers.l2(0.01)
model_simple.add(Dense(X_train.shape[1], activation='relu'))



model_simple.compile(optimizer=keras.optimizers.Adam(learning_rate=0.000001), loss='mse'

early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_

history_simple = model_simple.fit(X_train, y_train, epochs=80, batch_size=32, validation
```

```
# Access the loss values from history_simple
training_loss    =    history_simple.history['loss']
validation_loss = history_simple.history['val_loss']

# Plot the training and validation loss
epochs = range(1, len(training_loss) + 1)
plt.plot(epochs, training_loss, 'b-', label='Training Loss')
plt.plot(epochs, validation_loss, 'r-', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss for Simple')
plt.legend()
plt.show()
```




```
In [130]:
model_simple.summary()
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_93 (Dense)             (None, 8192)              253952

 dense_94 (Dense)             (None, 4096)              33558528

 dense_95 (Dense)             (None, 2048)              8390656

 dense_96 (Dense)             (None, 1024)              2098176

 dense_97 (Dense)             (None, 512)               524800

 dense_98 (Dense)             (None, 256)               131328

 dense_99 (Dense)             (None, 128)               32896

 dense_100 (Dense)            (None, 256)               33024

 dense_101 (Dense)            (None, 128)               32896

 dense_102 (Dense)            (None, 256)               33024

 dense_103 (Dense)            (None, 512)               131584

 dense_104 (Dense)            (None, 1024)              525312

 dense_105 (Dense)            (None, 2048)              2099200

 dense_106 (Dense)            (None, 4096)              8392704

 dense_107 (Dense)            (None, 8192)              33562624

 dense_108 (Dense)            (None, 30)                245790

=================================================================
Total params: 90,046,494
Trainable params: 90,046,494
Non-trainable params: 0
_________________________________________________________________
```

In [131]:

```
# Reconstruction Errors
X_train_pred_lstm = model_lstm.predict(X_train)
train_mae_loss_lstm = np.mean(np.abs(X_train_pred_lstm - X_train), axis=1)
```

```
101/101 [==============================] - 46s 439ms/step
```

In [132]:

```
# Calculate the predicted values from the Simple Autoencoder
X_train_pred_simple = model_simple.predict(X_train)

# Reshape X_train_pred_simple to match the shape of X_train
X_train_pred_simple = np.reshape(X_train_pred_simple, (X_train_pred_simple.shape[0], X_t
```

```
101/101 [==============================] - 5s 45ms/step
```

In [133]:

```
# Calculate the mean absolute error
train_mae_loss_simple = np.mean(np.abs(X_train_pred_simple - X_train), axis=1)
```

In [134]:

```
# Calculate MSE
mse_lstm = np.mean(np.square(X_train_pred_lstm - X_train))
mse_simple = np.mean(np.square(X_train_pred_simple - X_train))
```

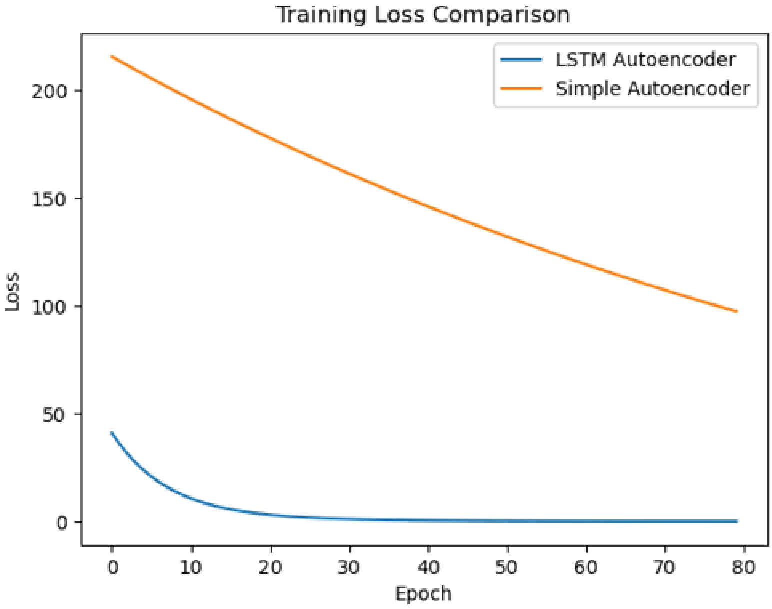
In [135]:

```
# Print the MSE scores
print("MSE (LSTM Autoencoder):..", mse_lstm)
print("MSE (Simple Autoencoder):", mse_simple)
```

```
MSE (LSTM Autoencoder):.. 0.05159797662554833
MSE (Simple Autoencoder): 0.056807787653000374
```

In [136]:

```
# Plotting the training loss for both autoencoders
plt.plot(history_lstm.history['loss'], label='LSTM Autoencoder')
plt.plot(history_simple.history['loss'], label='Simple Autoencoder')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training Loss Comparison')
plt.show()
```



In [137]:

```
# Reconstruction Errors
X_test_pred_lstm = model_lstm.predict(X_test)
test_mae_loss_lstm = np.mean(np.abs(X_test_pred_lstm - X_test), axis=1)
```

```
20/20 [==============================] - 8s 425ms/step
```

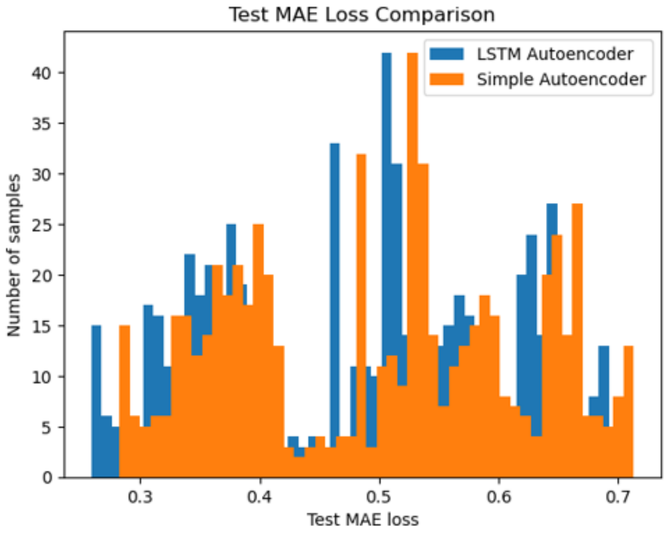
In [138]:

```
X_test_reshaped = X_test.reshape(X_test.shape[0], X_test.shape[1])  # Reshape X_test

X_test_pred_simple = model_simple.predict(X_test)
test_mae_loss_simple = np.mean(np.abs(X_test_pred_simple - X_test_reshaped), axis=1)
```

```
20/20 [==============================] - 1s 38ms/step
```

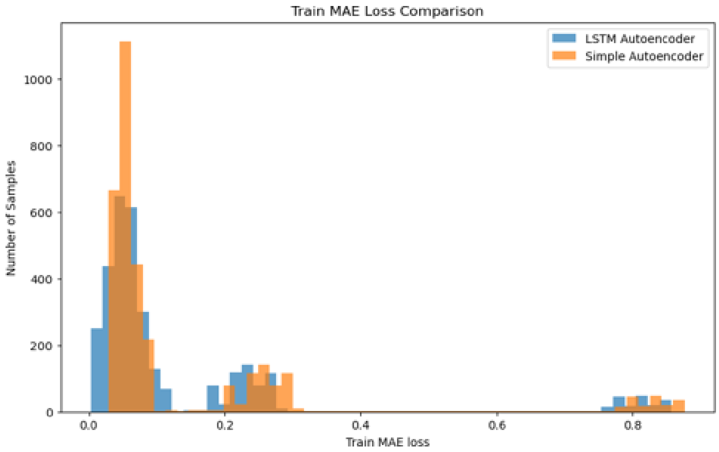
In [139]:

```
plt.hist(test_mae_loss_lstm, bins=50, label='LSTM Autoencoder')
plt.hist(test_mae_loss_simple, bins=50, label='Simple Autoencoder')
plt.xlabel('Test MAE loss')
plt.ylabel('Number of samples')
plt.legend()
plt.title('Test MAE Loss Comparison')
plt.show()
```




In [140]:

```
# Plot histograms of train MAE Loss for each autoencoder
plt.figure(figsize=(10, 6))
plt.hist(train_mae_loss_lstm, bins=50, label='LSTM Autoencoder', alpha=0.7)
plt.hist(train_mae_loss_simple, bins=50, label='Simple Autoencoder', alpha=0.7)
plt.xlabel('Train MAE loss')
plt.ylabel('Number of Samples')
plt.legend()
plt.title('Train MAE Loss Comparison')
plt.show()
```



## Set reconstruction error threshold

In [156]:

```
# Set reconstruction error threshold
threshold = 5*  np.mean(train_mae_loss_lstm)
threshold_simple = 5 * np.mean(train_mae_loss_simple)
#threshold=1.5
#threshold_simple=
print('Reconstruction error threshold for LSTM:...........................',threshold)
print('Reconstruction error threshold for Simple Autoencoder', threshold_simple)
```

```
Reconstruction error threshold for LSTM:...........................0.650752825163669
Reconstruction error threshold for Simple Autoencoder 0.6907895408784587
```

In [157]:

```
# Compute test MAE loss for each autoencoder
X_test_pred_simple = model_simple.predict(X_test)
X_test_pred_simple = np.squeeze(X_test_pred_simple)  # Remove the extra dimension

# Reshape X_test to match the shape of X_test_pred_simple
X_test_reshaped = X_test.reshape(X_test.shape[0], X_test.shape[1])

test_mae_loss_simple = np.mean(np.abs(X_test_pred_simple - X_test_reshaped), axis=1)
```

```
20/20 [==============================] - 1s 39ms/step
```

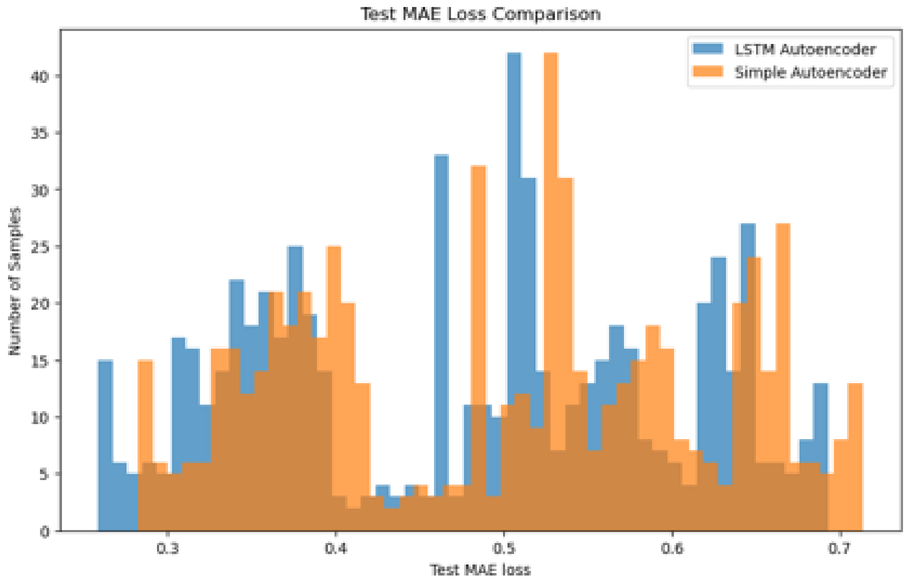
In [158]:

```
X_test_pred_lstm = model_lstm.predict(X_test)
test_mae_loss_lstm = np.mean(np.abs(X_test_pred_lstm - X_test), axis=1)
```

```
20/20 [==============================] - 9s 437ms/step
```

In [159]:

```
# Plot histograms of test MAE loss for each autoencoder
plt.figure(figsize=(10, 6))
plt.hist(test_mae_loss_lstm, bins=50, label='LSTM Autoencoder', alpha=0.7)
plt.hist(test_mae_loss_simple, bins=50, label='Simple Autoencoder', alpha=0.7)
plt.xlabel('Test MAE loss')
plt.ylabel('Number of Samples')
plt.legend()
plt.title('Test MAE Loss Comparison')
plt.show()
```



In [160]:

```
# Create anomaly DataFrame
anomaly_df = pd.DataFrame(test[TIME_STEPS:])
anomaly_df['loss'] = test_mae_loss_lstm
anomaly_df['threshold'] = threshold
anomaly_df['anomaly'] = anomaly_df['loss'] > anomaly_df['threshold']
```

In [161]:

```
anomaly_df
```

Out[161]:

| | Date | Open | High | Low | Close | Adj Close | Volume | loss |
|---|---|---|---|---|---|---|---|---|
| **3851** | 2019-12-06 | 66.671997 | 67.199997 | 66.671997 | 0.435440 | 67.030998 | 26296000 | 0.302005 |
| **3852** | 2019-12-09 | 66.902000 | 67.972504 | 66.891998 | 0.436432 | 67.178001 | 27086000 | 0.302900 |
| **3853** | 2019-12-10 | 67.074997 | 67.498749 | 66.802002 | 0.436803 | 67.233002 | 21882000 | 0.303782 |
| **3854** | 2019-12-11 | 67.542000 | 67.559998 | 67.133499 | 0.436924 | 67.250999 | 17008000 | 0.304396 |
| **3855** | 2019-12-12 | 67.296997 | 67.788750 | 67.025002 | 0.438695 | 67.513496 | 25620000 | 0.305323 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4715** | 2023-05-15 | 116.489998 | 118.794998 | 116.480003 | 0.772302 | 116.959999 | 22107900 | 0.588329 |
| **4716** | 2023-05-16 | 116.830002 | 121.199997 | 116.830002 | 0.793419 | 120.089996 | 32370100 | 0.591244 |
| **4717** | 2023-05-17 | 120.180000 | 122.279999 | 119.459999 | 0.802797 | 121.480003 | 26659600 | 0.594658 |
| **4718** | 2023-05-18 | 121.559998 | 123.900002 | 121.489998 | 0.816561 | 123.519997 | 27014500 | 0.598337 |
| **4719** | 2023-05-19 | 124.199997 | 126.478996 | 122.720001 | 0.814739 | 123.250000 | 30251300 | 0.602513 |

636 rows × 10 columns

In [162]:

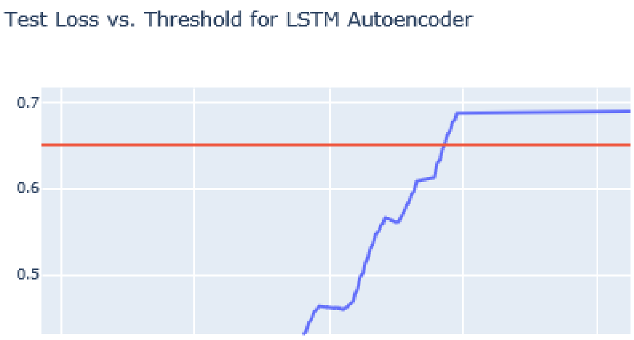
```
anomaly_df.shape #lstm
```

Out[162]:

```
(636, 10)
```

In [163]:

```
# Plot test loss vs. threshold
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Scatter(x=anomaly_df['Date'], y=anomaly_df['loss'], name='Test loss'))
fig.add_trace(go.Scatter(x=anomaly_df['Date'], y=anomaly_df['threshold'], name='Threshol
fig.update_layout(showlegend=True, title='Test Loss vs. Threshold for LSTM Autoencoder')
fig.show()
```

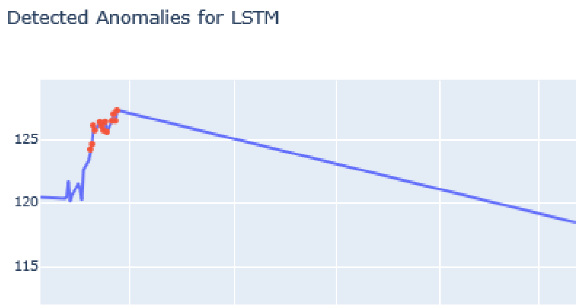



In [164]:

```
# Plot detected anomalies
anomalies_1 = anomaly_df.loc[anomaly_df['anomaly'] == True]
```

In [165]:

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=anomaly_df['Date'], y=scaler.inverse_transform(anomaly_df['Cl
fig.add_trace(go.Scatter(x=anomalies_1['Date'], y=scaler.inverse_transform(anomalies_1['
fig.update_layout(showlegend=True, title='Detected Anomalies for LSTM')
fig.show()
```




In [166]:

```
# Create anomaly DataFrame for Simple Autoencoder
anomalies_simple = pd.DataFrame(test[TIME_STEPS:])
anomalies_simple['loss'] = test_mae_loss_simple
anomalies_simple['threshold'] = threshold_simple
anomalies_simple['anomaly'] = anomalies_simple['loss'] > anomalies_simple['threshold']
```

In [167]:

```
anomalies_simple
```

Out[167]:

| | Date | Open | High | Low | Close | Adj Close | Volume | loss |
|---|---|---|---|---|---|---|---|---|
| 3851 | 2019-12-06 | 66.671997 | 67.199997 | 66.671997 | 0.435440 | 67.030998 | 26296000 | 0.325408 |
| 3852 | 2019-12-09 | 66.902000 | 67.972504 | 66.891998 | 0.436432 | 67.178001 | 27086000 | 0.326295 |
| 3853 | 2019-12-10 | 67.074997 | 67.498749 | 66.802002 | 0.436803 | 67.233002 | 21882000 | 0.327169 |
| 3854 | 2019-12-11 | 67.542000 | 67.559998 | 67.133499 | 0.436924 | 67.250999 | 17008000 | 0.327779 |
| 3855 | 2019-12-12 | 67.296997 | 67.788750 | 67.025002 | 0.438695 | 67.513496 | 25620000 | 0.328697 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 4715 | 2023-05-15 | 116.489998 | 118.794998 | 116.480003 | 0.772302 | 116.959999 | 22107900 | 0.609593 |
| 4716 | 2023-05-16 | 116.830002 | 121.199997 | 116.830002 | 0.793419 | 120.089996 | 32370100 | 0.612490 |
| 4717 | 2023-05-17 | 120.180000 | 122.279999 | 119.459999 | 0.802797 | 121.480003 | 26659600 | 0.615880 |
| 4718 | 2023-05-18 | 121.559998 | 123.900002 | 121.489998 | 0.816561 | 123.519997 | 27014500 | 0.619535 |
| 4719 | 2023-05-19 | 124.199997 | 126.478996 | 122.720001 | 0.814739 | 123.250000 | 30251300 | 0.623683 |

636 rows × 10 columns

In [168]:

```
anomalies_simple.shape
```

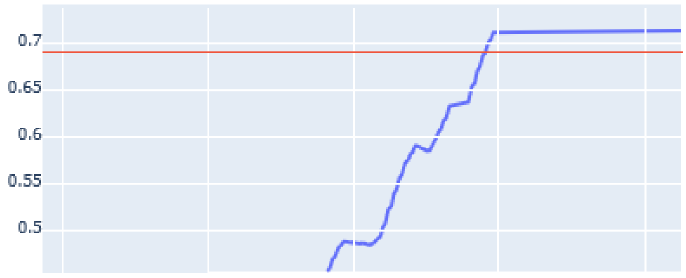
Out[168]:

```
(636, 10)
```

In [169]:

```
import plotly.graph_objects as go

fig_simple = go.Figure()
fig_simple.add_trace(go.Scatter(x=anomalies_simple['Date'], y=anomalies_simple['loss'],
fig_simple.add_trace(go.Scatter(x=anomalies_simple['Date'], y=anomalies_simple['threshol
fig_simple.update_layout(showlegend=True, title='Test Loss vs. Threshold for Simple Auto
fig_simple.show()
```

Test Loss vs. Threshold for Simple Autoencoder



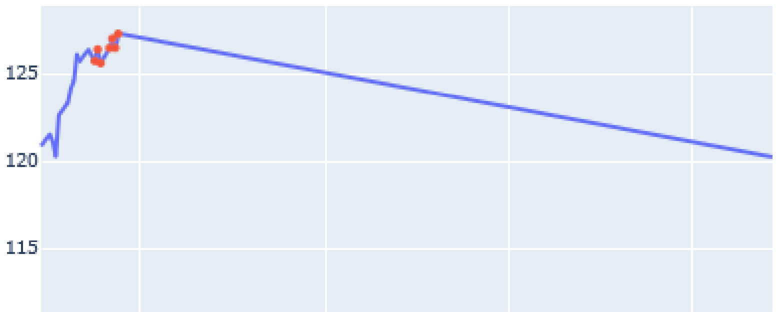

In [170]:

```
# Plot detected anomalies for simple autoencoder
anomalies_2 = anomalies_simple.loc[anomalies_simple['anomaly'] == True]
```

In [171]:

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=anomalies_simple['Date'], y=scaler.inverse_transform(anomalie
fig.add_trace(go.Scatter(x=anomalies_2['Date'], y=scaler.inverse_transform(anomalies_2['
fig.update_layout(showlegend=True, title='Detected Anomalies for Simple Autoencoder')
fig.show()
```

Detected Anomalies for Simple Autoencoder

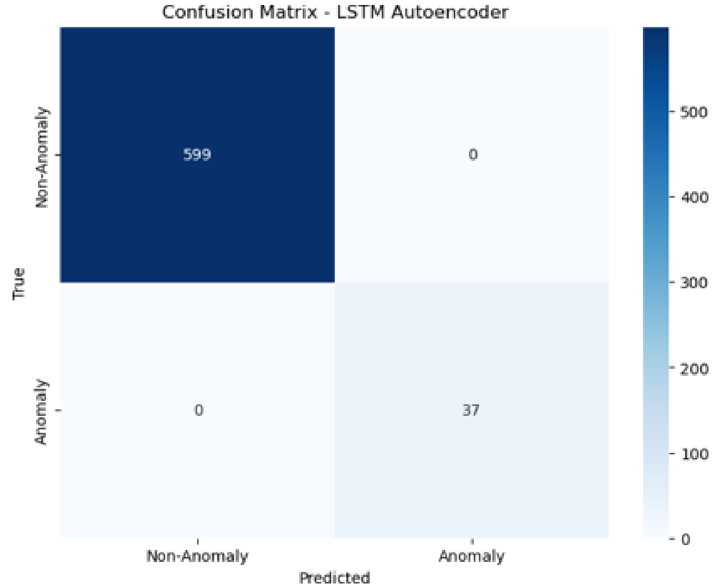


```
In [175]:
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score

# Reconstruction Errors
X_test_pred_lstm = model_lstm.predict(X_test)
test_mae_loss_lstm = np.mean(np.abs(X_test_pred_lstm - X_test), axis=1)

# Reshape X_test to match X_test_pred_simple shape
X_test_reshaped = X_test.reshape(X_test.shape[0], X_test.shape[1])

X_test_pred_simple = model_simple.predict(X_test_reshaped)
test_mae_loss_simple = np.mean(np.abs(X_test_pred_simple - X_test_reshaped), axis=1)

# Set threshold for anomaly detection
threshold_lstm = 5 * np.mean(train_mae_loss_lstm)
threshold_simple = 5 * np.mean(train_mae_loss_simple)
#threshold_simple= 4

# Create anomaly labels based on threshold
anomaly_labels_lstm = test_mae_loss_lstm > threshold_lstm
anomaly_labels_simple = test_mae_loss_simple > threshold_simple

# Evaluate LSTM Autoencoder
f1_lstm = f1_score(anomaly_labels_lstm, anomaly_labels_lstm)
recall_lstm = recall_score(anomaly_labels_lstm, anomaly_labels_lstm)
precision_lstm = precision_score(anomaly_labels_lstm, anomaly_labels_lstm)
confusion_mtx_lstm = confusion_matrix(anomaly_labels_lstm, anomaly_labels_lstm)

# Evaluate Simple Autoencoder
f1_simple = f1_score(anomaly_labels_simple, anomaly_labels_simple)
recall_simple = recall_score(anomaly_labels_simple, anomaly_labels_simple)
precision_simple = precision_score(anomaly_labels_simple, anomaly_labels_simple)
confusion_mtx_simple = confusion_matrix(anomaly_labels_simple, anomaly_labels_simple)

# Print evaluation metrics
print("LSTM Autoencoder:")
print("F1 Score:", f1_lstm)
print("Recall:", recall_lstm)
print("Precision:", precision_lstm)
print("Confusion Matrix:\n", confusion_mtx_lstm)

print("\nSimple Autoencoder:")
print("F1 Score:", f1_simple)
print("Recall:", recall_simple)
print("Precision:", precision_simple)
print("Confusion Matrix:\n", confusion_mtx_simple)
```

```
20/20 [==============================] - 13s 669ms/step
20/20 [==============================] - 2s 85ms/step
LSTM Autoencoder:
F1 Score: 1.0
Recall: 1.0
Precision: 1.0
Confusion Matrix:
[[599   0]
 [  0  37]]

Simple Autoencoder:
F1 Score: 1.0
Recall: 1.0
Precision: 1.0
Confusion Matrix:
[[612   0]
 [  0  24]]
```

In [176]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Define class labels for the confusion matrices
class_labels = ['Non-Anomaly', 'Anomaly']

# Plot confusion matrix for LSTM Autoencoder
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mtx_lstm, annot=True, fmt="d", cmap="Blues", xticklabels=class_lab
plt.title("Confusion Matrix - LSTM Autoencoder")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

# Plot confusion matrix for Simple Autoencoder
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mtx_simple, annot=True, fmt="d", cmap="Blues", xticklabels=class_l
plt.title("Confusion Matrix - Simple Autoencoder")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```



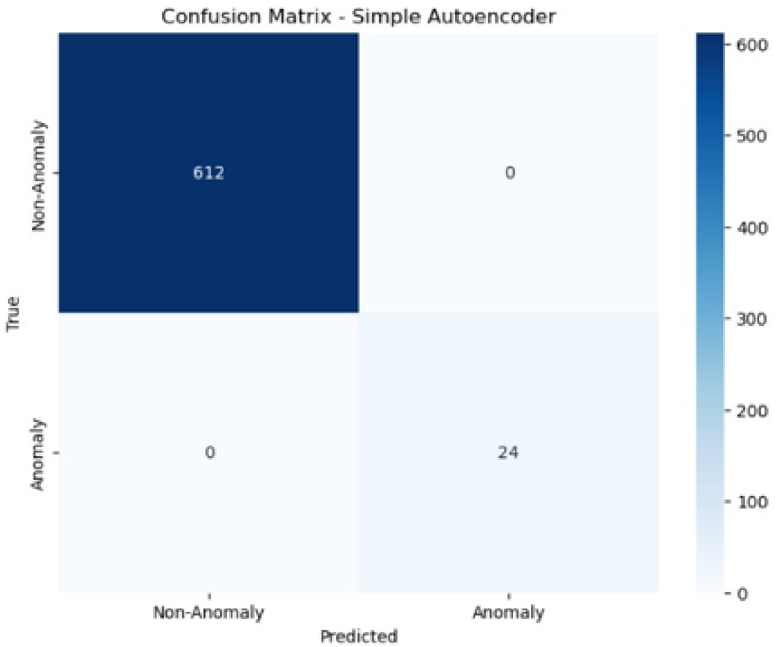
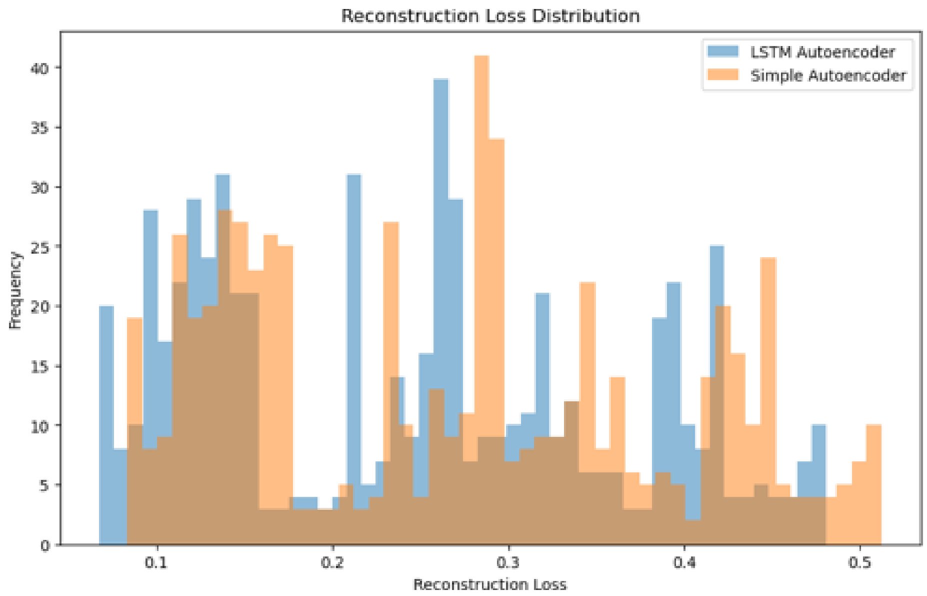




**Figure 10. – Confusion Matrix For Simple Autoencoder**

In [177]:

```
import matplotlib.pyplot as plt

# LSTM Autoencoder
lstm_val_predictions = model_lstm.predict(X_test)
lstm_val_losses = np.mean(np.square(lstm_val_predictions - X_test), axis=1)

# Simple Autoencoder
simple_val_predictions = model_simple.predict(X_test)
# Reshape X_test to match the shape of simple_val_predictions
X_test_reshaped = X_test.reshape(X_test.shape[0], X_test.shape[1])
simple_val_losses = np.mean(np.square(simple_val_predictions - X_test_reshaped), axis=1)

# Plotting the distribution
plt.figure(figsize=(10, 6))
plt.hist(lstm_val_losses, bins=50, label='LSTM Autoencoder', alpha=0.5)
plt.hist(simple_val_losses, bins=50, label='Simple Autoencoder', alpha=0.5)
plt.xlabel('Reconstruction Loss')
plt.ylabel('Frequency')
plt.title('Reconstruction Loss Distribution')
plt.legend()
plt.show()
```

```
20/20 [==============================] - 13s 665ms/step
20/20 [==============================] - 2s 82ms/step
```



**Proforma 4**

# Undertaking from the PG student while submitting his/her final dissertation to his respective institute

**Ref. No. _________**

I , the following student

| Sr. No. | Sequence of student's names on a dissertation | Students name | Name of the Institute & Place | Email & Mobile |
|---|---|---|---|---|
| 1. | Ritwik Dubey | RITWIK DUBEY | SIG | Email: 22070243038@sig.ac.in<br>Mobile: 7003019614 |

**Note:** Put additional rows in case of more number of students

hereby give an undertaking that the dissertation < **Autoencoder-based Anomaly Detection in Alphabet Inc. (Google) Dataset** >_has_been checked for its Similarity Index/Plagiarism through Turnitin software tool; and that the document has been prepared by me and it is my original work and free of any plagiarism. It was found that:

| | | |
|---|---|---|
| 1 | The Similarity Index (SI) was:<br>*(Note: SI range: 0 to 10%; if SI is >10%, then authors cannot communicate ms; **attachment of SI report is mandatory**)* | 2 % |
| 2. | The ethical clearance for research work conducted obtained from:<br>*(Note: Name the consent obtaining body; if 'not appliable' then write so)* | SIG |
| 3. | The source of funding for research was:<br>*(Note: Name the funding agency; or write 'self' if no funding source is involved)* | Self |
| 4. | Conflict of interest:<br>*(Note: Tick √ whichever is applicable)* | No |
| 5. | The material (adopted text, tables, figures, graphs, etc.) as has been obtained from other sources, has been duly acknowledged in the manuscript:<br>*(Note: Tick √ whichever is applicable)* | Yes |

In case if any of the above-furnished information is found false at any point in time, then the University authorities can take action as deemed fit against all of us.

RITWIK DUBEY

Ritwik Dubey

Full Name &
Signature of the student

Name &
Signature of SIU Guide/Mentor

Date: 02/07/2023

Endorsement by
Academic Integrity Committee (AIC)

Place: Pune

**Note:** It is mandatory that the Similarity Index report of plagiarism (only first page) should be appended to the UG/PG dissertation