CLOUD INSIDER THREAT DETECTION USING DEEP LEARNING MODELS

Project work submitted to Avinashilingam Institute for Home Science and Higher Education for Women

MASTEROFSCIENCEININFORMATIONTECHNOLOGY

SUBMITTED BY

DHANYA C J

21PIT001

Under the Guidance of

Dr. D. Shanmugapriya, M.Sc., M.Phil., Ph.D., SET.,

Assistant Professor and Head,

Department of Information Technology

AVINASHILINGAMINSTITUTEFORHOMESCIENCEANDHIGHEREDUCATIONFOR WOMEN

SCHOOLOFPHYSICALSCIENCESANDCOMPUTATIONALSCIENCES

DEPARTMENTOFINFORMATIONTECHNOLOGY

COIMBATORE: 641043

MAY-2023

DECLARATION

I hereby declare that the project entitled **"Cloud Insider Threat Detection Using Deep Learning Models"** is a record of the original work done by DHANYA C J (21PIT001) under the guidance of Dr. D. Shanmugapriya MSC., M. Phil., Ph.D., Assistant Professor and Head, Department of Information Technology, School of Physical Sciences and Computational Sciences, Avinashilingam Institute for Home Science and Higher Education for Women in the partial fulfilment for the award of the degree of Master of Science in Information Technology, and this project work has not formed the basis for any Degree/Diploma/Associates.

Place: Coimbatore

Date: 19.05.2023

Dhanya C J

Signature of the candidate

Countersigned By,

Dr. D. Shanmugapriya M.Sc., M.Phil., Ph.D.

Head of the Department

Assistant Professor

Department of Information Technology,

School of Physical Sciences and Computational Sciences,

CERTIFICATE

Avinashilingam Institute for Home Science and Higher Education for Women

(Deemed to be University under Estd. u/s 3 of UGC Act 1956, Category 'A' by MHRD)

Re-accredited with 'A++' Grade by NAAC. CGPA 3.65/4, Category 1 University by UGC

Coimbatore - 641 043, Tamil Nadu, India

DST - CURIE - AI Sponsored

Centre for Cyber Intelligence

CERTIFICATE OF PROJECT COMPLETION

This is to certify that Ms. Dhanya C J (21PIT001), Master of Information Technology, Avinashilingam Institute for Home Science and Higher Education for Women, has successfully completed the project entitled "Cloud Insider Threat Detection using Deep Learning Models" under Centre for Cyber Intelligence - Centre for Machine Learning and Intelligence - a DST - CURIE - AI facility during December 2022 - May 2023.

Dr. G. Padmavathi
Dean, School of PSCS
CCI - Principal Investigator

Dr. P. Subashini
Project Coordinator - DST - CURIE - AI

Dr. S. Kowsalya
Registrar

CERTIFICATE

This is to certify that this project work entitled "**Cloud Insider Threat Detection Using Deep Learning models**" done by **DHANYA C J**(21PIT001) has been submitted to Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore-641 043 in partial fulfillment of the requirement for the award of the degree of **MASTER OF SCIENCE IN INFORMATION TECHNOLOGY.** This Project has not found the basis for the award of any Degree/Associate/fellowship or similar title to any Candidate of any University. Certified as a Bonafide record of the work submitted for the viva-voce held on

Signature of the Head of the Department

Signature of the supervisor

Signature of the Examiner(s)

ACKNOWLEDGEMENT

I owe my sincere thanks to Lord Almighty and My lovable parents for showering their generous blessing supreme in all endeavors.

I wish to express my gratitude to **Prof. S.P. Thyagarajan**, Chancellor, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing the facilities to conduct this study.

I extend my thanks to **Dr. Bharathi Harishankar**, Ph.D., FRSA., Vice Chancellor, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing flamboyant help towards the completion of the study.

I record my deep sense of gratitude and indebtedness to **Dr. S. Kowsalya**, M.Sc., M.Phil., Ph.D., and Registrar, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing adequate help for the study

I grateful record my sincere thanks to **Dr. G. Padmavathi** M.Sc., M.Phil., Ph.D., Dean and Professor, School of Physical Sciences & Computational of Sciences, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for timely help rendered throughout the course of this work.

I heartily Thank my esteemed project guide **Dr. D. Shanmugapriya** MSC.,M.Phil., Ph.D., , Head of the Department Assistant Professor Department of Information Technology, for imparting tremendous assistance and well-timed support for the triumph four projects.

I express my honorable thanks to our project coordinator Department of Information Technology, for imparting tremendous assistance and well-timed support for the triumph of our project.

I sincerely thank all the staff members of the Department of Information Technology, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for their help and support.

I like to extend my gratitude to Ms.A.Roshini, Technical Assistant –Center of cyber intelligence, Department of Computer Science, For providing Project guidelines and always supporting me and encouraging me with valuable advice and Profound belief in my work and abilities-CURIE

– AI Sponsored Phase II for providing the laboratory facilities to execute my project. I would like to acknowledge the help rendered by Center for Cyber Intelligence, DST

I would like to express my special thanks to my parents, my friends and all my well-wishers for their constant encouragement, support and help in carrying out this work successfully.

Abstract:

Insider threats have become a major concern for organizations worldwide. These threats are perpetrated by employees or insiders who have access to sensitive data and information and can cause significant damage to the organization's operations and reputation. The most damaging cyberattacks typically originate from reputable insiders rather than malevolent outsiders or software. Insiders have a major advantage over outside forces in that they can get past security measures and avoid detection, which could seriously harm the organizational assets. The detection of insider threats using user behavior analysis is the main topic of this research. Based on user activity, user behavior is classified as either genuine or malicious using Deep learning (DL)models

This Project investigates cloud insider threat detection taken in an account using Deep learning models for this study CMUCERT synthetic insider threat dataset r6.1 version Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) DL models are implemented to detect and classify the cloud insider threat in the taken dataset. From the Performance evaluation metrics in the CMC CERT dataset the best accuracy has been obtained the comparison proved that our novel approach produces relatively good accuracy (98.60%), precision (97%), and F1 Score (98%).to access the robustness of the model in detecting malicious insiders. In an outcome, based on significant comparative analysis CNN. and LSTM it is recommended that the CNN performs better with 98.60% accuracy for the malicious insider threat detection

Keywords: CNN, deep learning, Insider threat, LSTM, user behavior.

TABLE OF CONTENT

CHAPTER I

Introduction

1.1 About the Project

Insider threats have shown their great destructive power in information security and financial stability and have received widespread attention from governments and organizations. Traditional intrusion detection systems fail to be effective in insider attacks due to the lack of extensive knowledge of insider behavior patterns. Instead, a more sophisticated method is required to have a deeper understanding of activities that insiders communicate with the information system.

In this project, cloud insider threat detection is being carried out using deep learning models such as Long Short-Term Memory (LSTM). Convolutional Neural Networks(CNN). This study has been done with the CMU CERT synthetic insider threat dataset R6 dataset acquired cert repository

1.2 Motivation and Justification

The threat that insiders pose to businesses, institutions, and governmental organizations continues to be of serious concern. Recent industry surveys and academic literature provide unequivocal evidence to support the significance of this threat and its prevalence. Despite this, however, there is still no unifying framework to fully characterize insider attacks and to facilitate an understanding of the problem, its many components, and how they all fit together. we focus on this challenge and put forward a grounded framework for understanding and reflecting on the threat that insiders pose Deep learning has emerged as a tool to identify potential insider threats by analyzing large amounts of data and identifying patterns that indicate unusual behavior. This can include factors such as accessing files outside of regular hours, unusual login activity, or behavior inconsistent with job roles. The use of deep learning in insider threat detection can potentially prevent costly data breaches and protect organizations from internal threats

1.3 Problem Statement

Insider threat detection is a major threat to an organization where one can act as a genuine person to attempt illegal activities**. To detect insider threat detection using deep learning models**

1.4 Objective

The objective of this project is to Detect Insider Threats using Deep Supervised Learning

1.5 Statistics

According to the Global Report on the Cost of Insider Threat reveals Insider Threats, insider threat incidents have risen 44% over the past two years, with costs per incident up to more than a third of $15.38 million.

Some of the recent facts and statistics about the rise of insider threat in an organizations

- The cost of credential theft to organizations increased 65% from $2.79 million in 2020 to $4.6 million at present.
- The time to contain an insider threat incident increased from 77 days to 85 days, leading organizations to spend the most on containment.
- Incidents that took more than 90 days to contain cost organizations an average of $17.19 million on an annualized basis.

1.6 Some of the Existing study about handling of insider threat

The most detrimental cyber-attacks are usually not originated by malicious outsiders or malware but from trusted insiders. The main advantage of insider attackers has over external elements is their ability to bypass security checks and remain undiscovered; this may cause serious damage to the organizational assets.

This project focuses on insider threat detection through behavioral analysis of users. User behavior is categorized a genuine malicious based on user activity. A series of events and activities are analyzed based on the efficient activities of the user behavior data included here

Since an insider has authorized access to an organization’s asset, therefore they might have a better opportunity to determine the confidentiality, availability or integrity of data than an external attacker. Various primary and secondary elements that may serve as an inspiration for an insider include financial gain or greed, revenge, anger, thrill, pressure, treachery, discontentment, jealousy, and organizational politics.

1.7 Proposed Study:

This study uses CMUCERT synthetic insider threat dataset r6.1 version repository The dataset has two versions, r4.2 and r6. The main difference between the two versions is that r6 includes additional features, such as employee HR data and email content. This additional data provides a more realistic simulation of insider threats and allows for more accurate detection using deep learning algorithms.

The r4.2 version of the dataset includes features such as logon activity, file activity, and email activity, while the r6 version includes these features as well as employee HR data, email content, and web activity. The additional features in r6 provide more context for insider threats and allow for more accurate detection of malicious behavior.

In terms of the size of the dataset, r6 is larger than r4.2. The r6 dataset includes over 2 million events, while the r4.2 dataset includes around 1 million events. The larger dataset size allows for more robust training of deep learning models and better detection of insider threats.

Overall, the r6 version of the CMU CERT synthetic insider threat dataset is a more comprehensive and realistic simulation of insider threats in an organizational setting. The additional features and larger dataset size provide more context and data for deep learning algorithms to analyze and detect malicious behavior.

The algorithm used for detecting insider threats in the CMU CERT synthetic insider threat dataset r6 can vary depending on the specific implementation. The most commonly used a for behavioral-based insider threat detection using deep learning include recurrent neural networks (RNNs), convolutional neural networks (CNNs), and long short-term memory (LSTM) networks. The general approach for using deep learning algorithms to detect insider threats involves training a model on a large dataset of user activity logs and identifying patterns of behavior that are indicative of malicious activity. The model is typically trained on a combination of supervised and unsupervised learning techniques to classify Genuine versus anomalous behavior. One common approach is to use sequence-based models, such as LSTMs, to analyze user activity logs over time and identify sequences of actions that may be indicative of malicious behavior. Another approach is to use feature-based models, such as CNNs, to extract relevant features from user activity logs and use these features to identify anomalous behavior. The specific algorithm used for detecting insider threats in the CMU CERT synthetic insider threat dataset r6 will depend on the specific implementation and the goals of the analysis. The

insider threat detection in CMC CERT dataset is addressed using CNN and LSTM deep learning models

A Review of Insider Detection Techniques Using ML And DL

Insider threat detection techniques using machine and deep learning can be broadly categorized into following types

- User behavior-based detection and
- Graph-based detection.

Both techniques have multiple models.

i. **A user behavior-based insider detection technique**

In user behavior-based detection; user behavior is categorized into two types, that is Genuine and malicious. Each user behavior is logged and is compared with a standard rule set (created by experts). If the behavior deviates from Genuine, it is considered malicious. A supervised time series-based solution using two layers deep auto-encoder can also be used for insider attack detection a technique using the LSTM-CNN algorithm has been shown to identify user anomalous behavior in, by monitoring user activities and extracting temporal features. While in detection algorithm Boost has been used and behavior characteristic features are extracted from audit logs. Technique proposed in extracted features and fields from user behavior logs for behavior auditing, and then these log files are used to train the Improved Hidden Markov Model (IHMM) for detection of malicious behavior. Random forest algorithm can be used for behavior analysis of individual user by analyzing its activities over a period of time to predict and detect insider threat, disturbing psychological patterns of individual users are obtained by analyzing electronic communications in. A state machine system is proposed in that can efficiently integrate policies from rule-based anomaly detection systems in order to create models which are followed by the insiders to launch an attack.

ii. Graph-Based Detection.

While most security projects have focused on fending off attacks coming from outside the organizational boundaries, a real threat has arisen from the people who are inside those perimeter protections. Insider threats have shown their power by hugely affecting national security, financial stability, and the privacy of many thousands of people. What is in the news is the tip of the iceberg, with much more going on under the radar, and some threats never being detected. We propose a

hybrid framework based on graphical analysis and anomaly detection approaches, to combat this severe cyber security threat. Our framework analyzes heterogeneous data in isolating possible malicious users hiding behind others. Empirical results reveal this framework to be effective in distinguishing the majority of users who demonstrate typical behavior

1.8 Significance of Insider Threat Detection in Cloud

Malicious insiders are where things become really interesting. Carnegie Mellon University states that all malicious insider incidents "involve misuse of authorized access to an organization's critical assets, which presents unique security challenges. Perimeter-based security strategies are not adequate to identify and prevent malicious behaviors from insiders. Moreover, insiders know which assets are most critical and how their organization protects them. Static and traditional security models focused on threats from external threat actors, therefore, are ineffective against insider threats."

The way that cloud security is modeled around the shared responsibility principle implies two different malicious insiders:

- Insiders from the cloud customer organization, and
- Insiders from the cloud service provider.

Some of the risks of insider threats in cloud security

Insiders, whether negligent or malicious, present a serious risk to a robust cloud security posture because insider attacks are harder to detect and respond to. Most organizations do not have the tools for monitoring "a Genuine user behavior across their cloud footprints."Besides the complicated detection and prevention of insider attacks, these attacks are dangerous because they "open up avenues for other attacks." In fact, insiders rank as the top cloud security threat facing public clouds. The Cybersecurity Insiders 2020 Cloud Security Report found that organizations ranked misconfiguration of the cloud platform (68%) as the highest threat. Insecure interfaces and APIs (52%) and malicious insiders (36%) were also among the top 10 cloud security threats.

1.9 Types of Insiders in the Cloud

There are three types of insider threats, Compromised users, Careless users, and Malicious users.

a) Compromised Employees or Vendors

Compromised employees or vendors are the most important type of insider threat you'll face. This is because neither of you knows they are compromised. It can happen if an employee grants access to an attacker by clicking on a phishing link in an email. These are the most common types of insider threats.

b) Careless Employees

Careless employees or vendors can become targets for attackers. Leaving a computer or terminal unlocked for a few minutes can be enough for one to gain access.

Granting DBA permissions to regular users (or worse, using software system accounts) to do IT work are also examples of careless insider threats.

c) Malicious Insider

Malicious attackers can take any shape or form. They usually have legitimate user access to the system and willfully extract data or Intellectual Property. Since they are involved with the attack, they can also cover up their tracks. That makes detection even more difficult.

d) Detecting Insider Threats

Most of the security tools used today try to stop legitimate users being compromised. This includes things like firewalls, endpoint scanning, and anti-phishing tools. They are also the most common types of breaches, so it makes sense that so much effort goes into stopping them.

The other two types of profiles aren't that easy to deal with. With careless behavior, knowing what system event was valid or not is almost impossible. Network and security admins probably don't know the context behind an application's behavior, so won't notice anything suspicious before it's too late.

Similarly, with malicious attackers, they will know the ins and outs of your company's security system. Giving them a good chance of getting away without being detected.The most significant issues with detecting insider threats are:

1. Legitimate Users

The nature of the threat is what makes it so hard to prevent. With the actor using their authentic login profiles, there's no immediate warning triggered. Accessing large files or databases infrequently may be a valid part of their day-to-day job requirements.

2. System and Software Context

For the security team to know that something terrible is happening, they need to know what something bad looks like. This isn't easy as. Usually, business units are the experts when it comes to their software. Without the right context, detecting a real insider threat from the security operations center is almost impossible.

3. Post Login Activities

Keeping track of every user's activity after they've logged in to the system is a lot of work. In some cases, raw logs need to be checked, and each event studied. Even with Machine Learning (ML) tools, this can still be a lot of work. It could also lead to many false positives being reported, adding noise to the problem

1.10 Some of the insider threat detection technique involved in ML and DL

The contribution of the project is to develop the suitable deep learning models. To detect Insider Threat accurately. In this project CNN and LSTM based on deep learning models are developed to encounter the problem. The organization of the document followed by the introduction are Chapter 2 discussed the system configuration, chapter 3 entails the Literary study, Chapter 4 details with Methodology carried out to developed the deep learning models, Chapter 5 discussed Experimental result and discussion also Chapter 6 concludes with future scope of the project

CHAPTER II

System Configuration

1.2 Hardware requirement

Processor: intel i7 and an above ram: 8 GB

Hard disk capacity: 1TB

2.2 Software requirement

Operating system: windows10

package: anaconda

Programing platform: python

2.3 About the environment

In this project the following Python environment is used to develop a significant model

The tools used in this project are listed below.

- Python 3.11.3
- Anaconda

2.3.1 Anaconda

Rather than using command lines, Anaconda Navigator's graphical interface can be used to install packages, manage environments, and run standard Python tasks. Moreover, it enables easy management of channels, environments, and packages for Anaconda without the need for command-line input. On the Anaconda Cloud or in a local Anaconda Repository, Navigator can search for packages. Windows, macOS, and Linux are all supported**.**

2.3.2 Jupyter Notebook

An open-source web program called the Jupyter Notebook allows users to create and share documents with live code, equations, visualizations, and text. The folks who work on Project Jupyter maintain Jupyter Notebook. The I-Python project, which once had its own I-Python Notebook project, is where Jupyter Notebooks got their start. Jupyter's name is derived from the three primary programming languages it supports: Julia, Python, and R. There are already more than 100 additional kernels available; however, Jupyter comes with the I-Python kernel, which enables Python programmed writing.

2.33 Some of the Python package and libraries involved in this project to develop the deep learning models

Python Package

a) Scikit-Learn

Scikit-learn, a free Python package that is frequently seen as a direct extension of SciPy, is based on NumPy and SciPy. It is specially made for creating supervised and unsupervised machine learning algorithms and data modeling.

Scikit-learn is user-friendly and beginner-friendly because of its straightforward, intuitive, and consistent interface. Scikit-learn performs admirably by enabling users to alter and exchange data as they need, despite the fact that its utility is constrained because it only excels at data modelling

b) Pandas

Python's Pandas package for data research and analysis enables programmers to create simple, seamless high-level data structures. Pandas, which is based on NumPy, is in charge of getting data sets and data points ready for machine learning. Pandas uses one-dimensional (series) and two-dimensional (Data Frame) data structures. These two types of data structures allow Pandas to be used in a range of industries, from science and statistics to banking and engineering.

Due to its adaptability, the Pandas library can be used with other scientific and numerical libraries. Because they are rapid, compliant, and highly descriptive, their data structures are simple to use. By aggregating, integrating, and re-indexing data with Pandas, one can modify data functionality with a minimum of keystrokes.

c) Matplotlib

Matplotlib is a data visualization library that is used for making plots and graphs. It is an extension of SciPy and is able to handle NumPy data structures as well as complex data models made by Pandas. Although it is expertise is limited to 2D plotting, Matplotlib can produce high-quality and publish-ready diagrams, graphs, plots, histograms, error charts, scatter plots, and bar charts.

Matplotlib is intuitive and easy to use, making it a great choice for beginners. It is even easier to use for people with pre-existing knowledge of various other graph-plotting tools. It offers GUI toolkits support, including wxPython, Tkinter, and Qt.

d) NumPy

Open-source and well-known Python library for numbers, NumPy. It is capable of carrying out a wide range of mathematical operations on matrices and arrays. One of the most popular libraries for scientific computing, it is frequently used by scientists to analyze data.

It is perfect for machine learning and artificial intelligence (AI) projects since it can process multidimensional arrays, handle linear algebra, and perform Fourier transformation. 10

NumPy arrays demand a considerable reduction in storage space when compared to standard Python lists. They are also lot easier to operate and considerably faster than the earlier. One can reshape, transpose, and modify data in matrix form with NumPy. Combining NumPy's capabilities makes it easy to enhance the machine learning model's performance.

e) Seaborn

An open-source Python package for data visualization and graphing is called Seaborn. It uses sophisticated Panda's data structures and is based on the graphing software Matplotlib. Seaborn offers a high-level, feature-rich interface for creating precise, illuminating statistical graphs on its own. Because it can produce logical graphs of learning and execution data, it is employed in machine learning and deep learning applications.

The most beautiful and eye-catching graphs and plots are produced by Seaborn, which makes it ideal for use in publishing and marketing. Seaborn can also save you time and effort because it enables you to build complex graphs with little code and basic instructions.

f) Train-Test Split

The train-test split is used to determine how well machine learning algorithms work when employed with prediction-based methods and applications. To compare the output of one's own machine learning model, one can use this quick and simple procedure.

CHAPTER III

REVIEW OF LITERATURE

This selection describes the some of the pervious researched work carried out to detect insider threat using various ML and DL models are represented in the Table 1.

Recent Literature review

S.no	Tittle Author name and year Pushiest	dataset used	Algorithm applied
1	"Malware Detection in Cloud Computing Infrastructures", International Journal of Recent Trends in Engineering and Research, pp. 223-227, 2018	Techniques allow to detect malware	• The attacker tries to keep the malware survive even the browser is closed
2	"Malware definition – What is it and how to remove it", Malwarebytes, 2018. [Online]. Available: https://www.malwarebytes.com/malware/.	Techniques allow to detect malware	• Review doesn't practically implemented
3	LIU and Y. OU, "An Improved XSS Vulnerability Detection Method Based on Attack Vector", DEStech Transactions on Materials Science and Engineering, no., 2018	Explains about the XSS vulnerability detection on attack vectors	• Malicious code to make it seem like reliable
4	Y. Jang, "Buffer Overflow Vulnerability Instrumentation Technique to Safety Checks of Variables", Korean Society of Technical Education and Training, vol. 23, no. 2, pp. 51-64, 2018	Explains about the vulnerability techniques	• error occurs when the dangling pointer is used after it had been free without allocating new memory location
5	"X-XSS-Protection", MDN Web Docs, 2018. [Online]. Available: https://developer.mozilla.org/enUS/docs/Web/HTTP/Headers/X-XSSProtection.	Explains about the XSS vulnerability detection on attack vectors	• Doesn't explains about the XSS completely

<table>
<tr><th>Sno</th><th>Authors &Year</th><th>Topic</th><th>Journal</th><th>Methods/Algorithm Used</th><th>Result/Observation</th></tr>
<tr><td>6</td><td>Xiaoshuang Sun, Yu Wang, 2021</td><td>Insider Threat Detection Using An Unsupervised Learning Method: COPOD</td><td>2021 IEEE 3rd International Conference on Communications, Information System and Computer Engineering (CISCE 2021)</td><td>→The tree structure method to represent a large number of user activity log data in a tree structure

→The tree structure method to analyze user behavior, form feature sequences, and combine the Copula Based Outlier Detection (COPOD) method to detect
→The difference between feature sequences and identifying abnormal users.

Datasets
→The CERT-R4.2 dataset, which collects artificial attacks from real enterprise environments, records the behavior of thousands of users over a period of a year and a half.
→This dataset is a multi-source dataset with rich data types, including host log, network log, employee psychological evaluation and human resource information</td><td><table><tr><td>Method</td><td>Accuracy</td></tr><tr><td>Isolated Forest</td><td>0.958</td></tr><tr><td>COPOD</td><td>0.975</td></tr></table>
→We adopted the tree structure method to analyze the user behavior and form the feature sequence.
→We superior to the common unsupervised learning algorithm.
→The method has advantages in processing large-scale complex and heterogeneous data.</td></tr>
<tr><td>7</td><td>Pallabi Parveen, Nate McDaniel, Varun S. Hariharan, Bhavani</td><td>Unsupervised Ensemble based Learning for Insider</td><td>2012 ASE/IEEE International Conference on</td><td>→ Tests an unsupervised, ensemble based learning algorithm that maintains a compressed dictionary of repetitive sequences found throughout dynamic data streams of unbounded length to identify anomalies.</td><td>→This results in a classifier exhibiting a substantial increase in classification accuracy for data streams containing insider threat anomalies.</td></tr>
</table>

	Thuraisingham and Latifur Kha 2018	Threat Detection	Social Computing and 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust	→In unsupervised learning, compression-based techniques are used to model common behavior sequences. → Algorithm 1 shows the basic building block for updating the Ensemble → Algorithm 2 shows step by step how a quantized dictionary is generated from LZW dictionary	→This ensemble of classifiers allows the unsupervised approach to outperform traditional static learning approaches and boosts the effectiveness over supervised learning approaches →The approach adopts advantages from both compression & ensemble-based learning. →In particular, compression offered unsupervised learning in a manageable manner and on the other hand ensemble based learning offered adaptive learning →The approach was tested on real command line dataset and shows effectiveness over static approaches in terms of TP and FP

SNO	Author name and year Pushlished	Title	Methods/Algorithm	Result/Observation
8	Xiaoshuang Sun, Yu Wang,	Insider Threat Detection Using An Unsupervised Learning Method: COPOD	→The tree structure method represents a large number of user activity log data in a tree structure →The tree structure method to analyze user behavior, form feature sequences, and combine the Copula Based Outlier Detection (COPOD) method to detect →The difference between feature sequences and identify abGenuine users. Datasets →The CERT-R4.2 dataset, which collects artificial attacks from real enterprise environments, records the behavior of thousands of users over a period of a year and a half.	→We adopted the tree structure method to analyze the user behavior and form the feature sequence. →We superior to the common unsupervised learning algorithm .
9	Pallabi Parveen, Nate McDaniel, Varun S. Hariharan, Bhavani Thuraisingham and Latifur Kha	Unsupervised Ensemble based Learning for Insider Threat Detection	→ Algorithm 1 shows the basic building block for updating the Ensemble → Algorithm 2 shows step by step how a quantized dictionary is generated from LZW dictionary	This results in a classifier exhibiting a substantial increase in classification accuracy for data streams containing insider threat anomalies. approaches

10	William R Claycomb,	Insider Threats to Cloud Computing: Directions for New Research Challenges	The common notion of a cloud insider as a rogue administrator of a service provider is discussed, but we also present two additional cloud related insider risks: the insider who exploits a cloud-related vulnerability to steal information from a cloud system, and the insider who uses cloud systems to carry out an attack on an employer's local resources.	→The cloud-related insider threats should focus on identifying and addressing unique vulnerabilities posed by the use of cloud computing services
11	RIDA NASIR1 , MEHREEN AFZAL 1 , RABIA LATIF 2 , AND WASEEM IQBAL	Behavioral Based Insider Threat Detection Using Deep Learning	The insider threat detection through behavioral analysis of users. →User behavior is categorized as Genuine or malicious based on user activity. →A deep learning based approach is proposed that detects insiders with greater accuracy and low false positive rate.	The comparison proved that our novel approach produces relatively good accuracy (90.60%), precision (97%) and F1 Score (94%).

CHAPTER IV

METHODOLOGY

This chapter explains the step-by-step methodology used to detect insider threat in the CMU CERT synthetic insider threat dataset R6 version is discussed

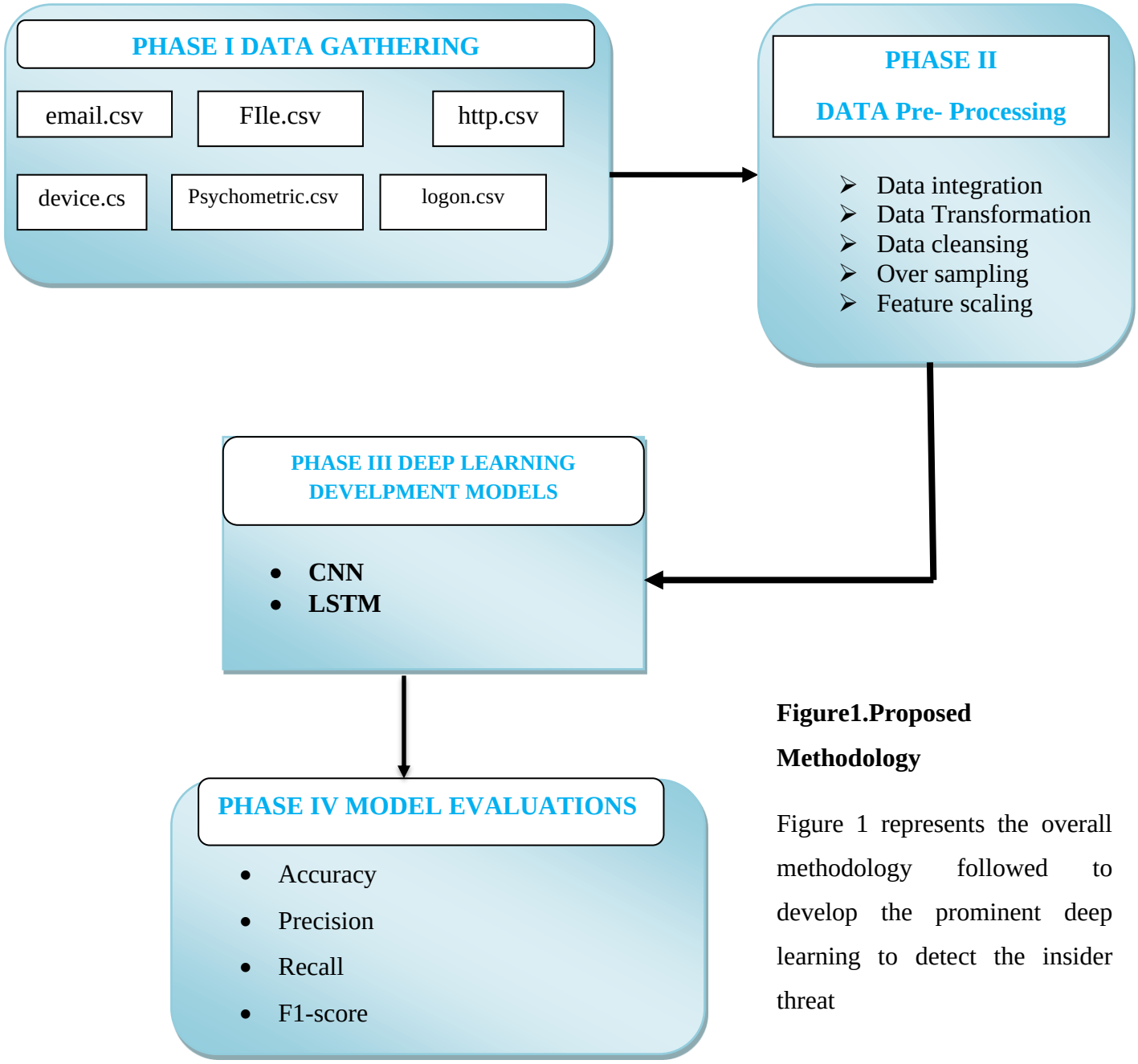


Figure1.Proposed Methodology

Figure 1 represents the overall methodology followed to develop the prominent deep learning to detect the insider threat

Phase I – Data acquisition

Generally, CMU CERT synthetic insider threat dataset consists of different versions

(i.e., r1, r2, r3, r4, r5, r6). This project concentrates on the r6.1 version because this version data consists anomalous count of data with significant user behavioral features

The dataset used is the CMU CERT synthetic insider threat dataset r6.1. The dataset consists of synt hetic data of both Genuine and malicious insiders. To make the approach simple all the csv files are aggregated and the most relevant features are extracted. The dataset consists of 159998 synthetic us ers out of which 11 are malicious insiders. .

In this dataset, a malicious insider user is designed to accomplish one out of the following two scenarios at some point in time

- Use of external hard drives, or work after hours, login activity after office hours by the user who did not have such previous routine, using of a flash drive, uploading data to wikileaks.org and then leaving the organization shortly thereafter.
- User visiting job sites and seeking employment from a competitor. Use of a flash drive (at markedly higher rates than their previous activity) to steal data before leaving the organization

The dataset consists of various csv files in which following are Included:

SNO	R6 CMU CERT DATASET LOG LIFES	SIZE	Description
1	logon.csv	(3533792, 5)	Log of users logging in and out on a computer
2	device.csv	(871043, 6)	Log of users connecting and disconnecting external devices (USB)
3	http.csv	(100000, 7)	Users' browser history
4	email.csv	(30000, 12)	Email logs
5	file.csv	(100000, 9)	Log of user activity on files (coping file to an external device)
6	Psychometric.csv	(4000, 7)	Contains user personality attributes
7	LDAP (Lightweight Directory Access Protocol)	((587, 10)	Set of files describing all users and their assigned job roles

In total the dataset consists of 2 million samples as different log files Table 2 shows the description of r6.1 version datasets

.

PHASE II – DATA PREPROCESSING

Data Pre- Processing Techniques Implemented in the Datasets

4.2 Data Preprocessing: The first step is to preprocess the dataset, which involves cleaning the data, handling missing values, and transforming the data into a format suitable for deep learning models. This may involve feature transformation, where additional features are created based on the available data, and all seven datasets as merged as one file to improve model performance

It raises reliability and accuracy. Pre-processing data can increase the correctness and quality of a dataset, making it more stable by removing missing or inconsistent data values brought on by human or computer mistakes. It ensures consistency in data.

The taken dataset is a benchmark one so that they are suitable for preprocessing techniques as being applied to redefine the data to support deep learning model development

Data preprocessing is a crucial step in behavioral-based insider threat detection using deep learning on the CMU CERT synthetic insider threat dataset R6. It involves cleaning and transforming the raw data into a format suitable for deep learning models. The following are the steps involved in data preprocessing:

- Data integration
- Data transformation
- Data cleaning
- Oversampling
- Features selection

i. Data Integration

The first step involved in data pre-processing is to integrate the seven different CMU CERT log files into a single combined dataset for that a join method under data integration is applied to select the common attributed involved in seven different log files are identified and merged into single csv files

The process of merging data from several sources into a coherent and consistent representation the next step is to select the relevant features for the deep learning models. In the case of the CMU CERT dataset R6, the relevant features may include user activity logs such as login times, file access times, and network activity logs. Other features such as user role, department, and job title may also be relevant

This procedure entails locating and accessing the various data sources, converting the data to a standard format, and resolving any errors or inconsistencies between the sources Data integration aims to facilitate access to and analysis of data that is dispersed across numerous platforms or systems in order to provide a more thorough and accurate view of the data.

- Increased operational efficiency by reducing the need to manually transform and combine data sets
- Better data quality through automated data transformations that apply business rules to data
- More valuable insight development through a holistic view of data that can be more easily analyzed

The seven log files of the CMU CERT dataset are integrated based on the common attributes included in each file (‘id’, ‘date’,’user_id’, ‘pc’, ‘activity’)

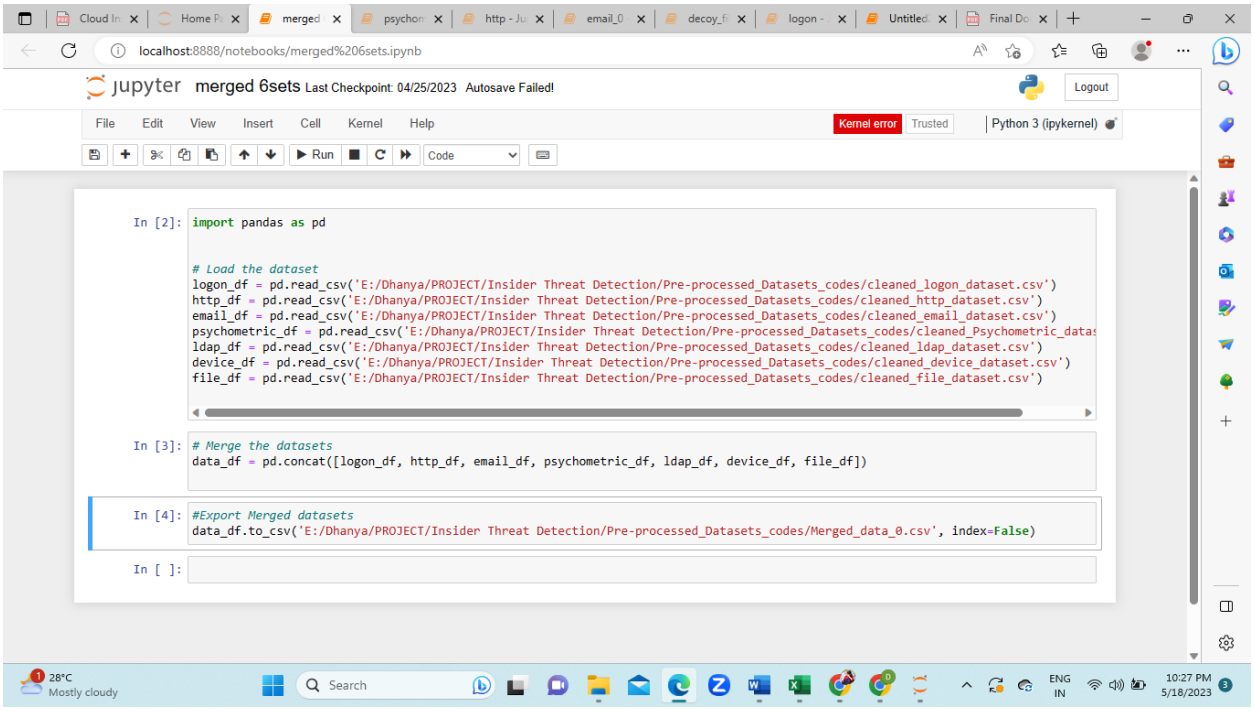


Figure 1. Data integration

ii. Data Transformation

The process of transforming raw data into from one form to another form data analysis to support the model building In order to transform the data into the right form, data transformation techniques also include data cleansing and data reduction.

Label Encoding

- Label encoding is the process of transforming labels into a numeric form so that they may be read by machines.

- Deep learning methods can then be used to determine how well those labels are functioning.

- It is a crucial stage in the supervised learning pre-processing of the structured datasets

In this dataset label encoding data transforming to convert the dataset features from float data type to integer type The below figure 2 shows the encoded dataset feature

Out[8]:

	Unnamed: 0	InsiderThreat	id	date	user_id	pc	activity
0	0	0	{E4E0-E6ME15OR-2988BSJL}	01-02-2010 06:21	NDD0597	pc-7588	Logon
1	1	0	{E3N5-D9NF71OY-8015TELH}	01-02-2010 06:22	JMB0141	pc-0100	Logon
2	2	0	{V5X9-F9OJ87RB-2481RLEY}	01-02-2010 06:37	EEC3140	pc-8416	Logon
3	3	0	{P6E0-B1AN30BI-3688NCXJ}	01-02-2010 06:39	HFB0003	pc-0124	Logon
4	4	0	{P7Z1-O0EZ42UI-9603XJVG}	01-02-2010 06:41	NNW0216	pc-0052	Logon
...	...	...	...	...	...	...	...
200007	7	1	{X5Z8-I0NU24TP-7924CGYS}	05/19/2011 21:36:42	HJB0462	PC-0761	Logoff
200008	8	1	{J7M7-L1AF74XQ-3406EXZP}	05/20/2011 19:27:59	HJB0462	PC-0761	Logon
200009	9	1	{G9V7-I2HP78OM-1408UDVG}	05/20/2011 19:42:34	HJB0462	PC-0761	Logoff
200010	10	1	{K1V5-D5FE66ER-2473NGWG}	05/20/2011 19:51:49	JRM0035	PC-0761	Logon
200011	11	1	{K1I0-V2XV30KI-3221ZOJT}	05/20/2011 20:00:51	JRM0035	PC-0761	Logoff

200012 rows × 7 columns

Figure.2 Encoded data

iii. Data Cleaning

Data cleaning is a method of locating the incomplete, inaccurate, irrelevant, or missing portions of t he data and then replacing, deleting, or modifying them as required In data cleaning the data cleaned by removing any irrelevant or duplicate information. This includes removing empty or null fields, removing duplicate records, and ensuring that the data is consistent in terms of formatting and data type.

In this project the dataset checked for null values and it can be replaced with suitable values the following are the values

i. Handling Null Values

There are almost never any null values in a real-world dataset. No model can handle these NULL values on its own, thus we must step in regardless of whether the issue is one of regression, classification, or any other kind. Python represents NULL with NaN. So, don't mix the two as they can be used alternately.

We can approach this issue in a number of different ways. Dropping the null-valued rows or columns is the simplest solution to this issue.

If the missing values are not handled correctly, you can wind up creating a machine learning model that is biased and produces inaccurate results. Missing data can make the statistical analysis less precise.

ii. Removing Duplicate Values

A dataset contains many instances of a duplicate value. It is frequently discovered when using Excel to work with huge databases.

Data processing will be unsuccessful if duplicate records are not eliminated. The goal of this control is to eliminate multiple records from the dataset in order to make it ready for further processing.

iii. Dropping NAN Values

The numeric data type NaN (Not a Number) refers to undefined values or values that cannot be represented, particularly the outcomes of floating-point calculations. Your Deep learning model will function better if you clean the data.

iv. Standard Scalar

- The Standard Scalar, a machine learning technique, scales the value distribution so that the mean and standard deviation of the observed data are 0 and 1, respectively.
- In the absence of scaling, the technique might benefit the feature with higher magnitude values. Scaling the features in a machine learning model could accelerate the gradient descent process and improve the flow of the cost function reduction
- The standard deviation formula is given as:

$$\sigma=\sqrt{1N\Sigma Ni=1(Xi-\mu)/2}$$

Were,

σ = Standard deviation symbol

μ = Mean

N = Total number of observations.

v. Genuineization

When preparing data for machine learning, Genuineization is a scaling technique that modifies the values of numerical columns to use a standard scale.

Only when the feature ranges in machine learning models differ is it necessary.

Genuineization aims to scale down features to a similar scale. This enhances the model's functionality and training stability.

Genuineization and Scaling: The next step is to Genuineize and scale the data to ensure that all features are on a similar scale. This can be done using techniques such as min-max scaling or standardization.

Encoding: Categorical features such as user role, department, and job title need to be encoded into numerical values for the deep learning models to work with. This can be done using techniques such as one-hot encoding or label encoding.

Sequence Generation: The CMU CERT dataset R6 is a time-series dataset, which means that the sequence of events is important. Therefore, the data needs to be transformed into sequences of events, with each sequence representing the behavior of a single user over a period of time.

Genuineization Techniques in Deep Learning

Even though there are numerous feature Genuineization methods in machine learning, only a few are actually utilized most commonly. These are as follows:

Min-Max Scaling: Scaling is another name for the Min-Max scaling method. The Min-Max scaling strategy assists the dataset in shifting and rescaling the values of its attributes so that they end up falling between 0 and 1.

A Min-Max scaling is typically done via the following equation:

m = (x -xmin) / (xmax -xmin)

Where,

m is our new value

x is the original cell value

xmin is the minimum value of the column

xmax is the maximum value of the column

vi. **Standardization scaling**: Standardization scaling, often referred to as Z-score Genuineization, involves distributing data so that they are centered on the mean and have a unit standard deviation. As a result, the attribute becomes zero and the distribution that has a unit standard deviation.

The formula for calculating Z score Genuineization is

$$\textbf{Z score} = (x - \bar{x})/\sigma$$

Where,

x = Standardized random variable

$\bar{x}$ = Mean

σ = Standard deviation.

vii. **Sampling**

Sampling in data analysis is the process of looking at a small subset of all the data to find the important information in the broader data set.

Only a small portion of an intrusion detection record actually reflects attacks. As a result, building effective models with high attack detection rates is difficult. Because the prediction is incredibly poor with the bulk class samples, a highly biased prediction model is not practically effective.

Over Sampling

The oversampling method includes adding duplicate records at random to the dataset for minority classes.

Oversampling techniques include the following:

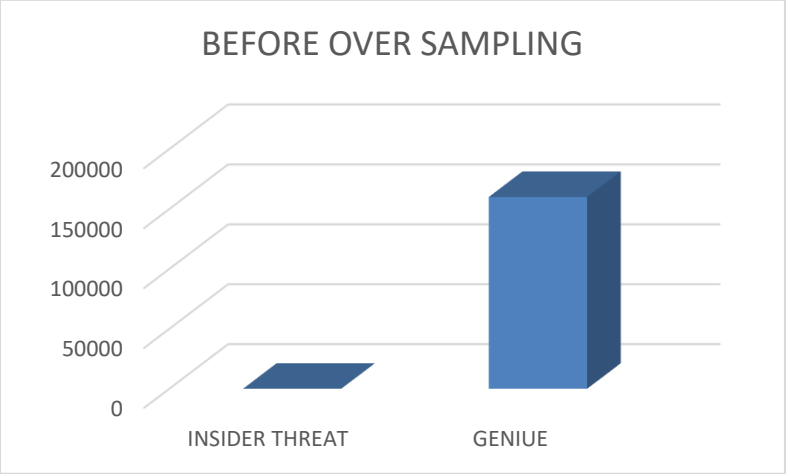
- Borderline-SMOTE

- Borderline Oversampling
- Adaptive Synthetic Sampling (ADASYN)
- Random Oversampling
- Synthetic Minority Oversampling Technique (SMOTE) The Synthetic Minority Oversampling Technique (SMOTE) from Oversampling is utilized in this project to boost the number of majority records in web attacks.

a) Synthetic Minority Oversampling Technique (SMOTE)

The Synthetic Minority Oversampling Technique, or SMOTE, is the most used method for synthesizing new records. In their 2002 study titled "SMOTE: Synthetic Minority Over-sampling Technique," Nitesh Chawla, et al. developed this technique. One of the most popular oversampling techniques for resolving an unbalanced dataset is SMOTE (synthetic minority oversampling technique). By recreating minority classes at random, it seeks to balance the distribution of classes. Here, the SMOTE technique is suggested as a solution to the problem of category imbalance.
SMOTE approaches have been investigated as a means of overcoming the skewness in the dataset. That some attack types, such as web attacks, are very skewed. SMOTE is used to oversample web attack samples. The number of web attack samples is raised using SMOTE to 24125. As there are enormous amounts of samples in typical traffic, there is also the potential for enormous amounts of assaults to be over-sampled. The imbalance ratio is calculated after SMOTE have been applied.

BEFORE OVERSAMPLING



AFTER OVERSAMPLING

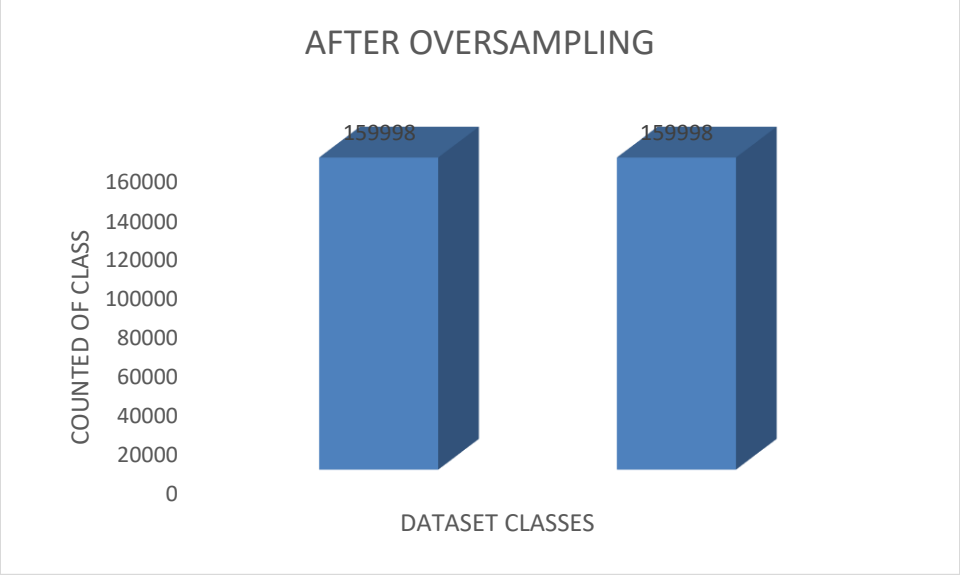


Figure 3 oversampling

The dataset is unbalanced data. It is not suitable for developing deep learning models so hence it is converted to balanced smote

Phase IV – Deep learning Model development

4.5 Model Classification

Deep learning is the branch of machine learning which is based on artificial neural network architecture. An artificial neural network or ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data.

In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other. Each neuron receives input from the previous layer neurons or the input layer. The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the final layer produces the output of the network. The layers of the neural network transform the input data through a series of nonlinear transformations, allowing the network to learn complex representations of the input data.

The science and practice of machine learning may allow programmed computers to learn from the data provided to them. Computers are frequently trained on the data (training set) provided to them throughout the machine learning process and they can demonstrate their performance on a specific data set. (Test set). In this approach, the problem is resolved with the least amount of human involvement. Where traditional methods are ineffective, machine learning is widely used.

Following is a list of possible uses:

- It has the ability to tackle complex problems, which traditional approaches are unable to do.
- It is capable of interpreting enormous amounts of complex data
- It is capable of resolving complex issues for which standard approaches are ineffective.
- Deep learning techniques can offer better answers when the state-of-the-art ones call for excessively frequent external updates or external intervention, respectively.

Different surroundings can support it. Data analysis is frequently used to apply Deep learning techniques to a new circumstance.

Supervised Learning

The training data had been appropriately identified and labeled using this way. For instance, the dataset contains information (tags/labels) regarding the type of each flow (such as Genuine or harmful). These tags are compared to the findings discovered by the algorithm in the following stage, the test/prediction phase, and the algorithm's success is also determined. The strategy has a good track record of success. Although expensive, supervised learning relies on outside services (like manual tagging) for labeling, and this process is repeated until the algorithm performs at a high degree of accuracy.

Model Training: The next step is to train a deep-learning model on the preprocessed data. This typically involves selecting appropriate model architecture, such as an LSTM-Autoencoder or a CNN, and optimizing the hyperparameters of the model using techniques such as grid search or random search. The model is trained on the Genuine behavior data from the CMU CERT dataset R6, so it can learn the patterns of Genuine user behavior.

Here, the algorithms that were employed are listed:

→ Convolution Neural Network, (CNN)

→ Long Short-Term Memory (LSTM)

Convolution Neural Network

Neural networks are a subset of Deep learning and are at the core of deep learning algorithms, as was stated in the Neural Networks Learn Hub page. They are made up of node levels, each of which includes an input layer, one or more hidden layers, and an output layer. Each node has a threshold and weight that are connected to one another. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer. Otherwise, no data is transmitted to the network's next level

There are other kinds of neural nets, which are utilized for diverse use cases and data types, while we mainly concentrated on feed forward networks in that article. Recurrent neural networks, for instance, are frequently used for speech and natural language processing, but convolution neural networks also known as CNNs are more frequently employed for classification and computer vision applications. Before CNNs, identifying objects in images required the use of laborious, manual feature extraction

techniques. Convolution neural networks, on the other hand, now offer a more scalable method for classifying images and recognizing objects by using matrix multiplication and other concepts from linear algebra to find patterns in images. However, they can be computationally challenging, requiring the use of graphics processing units (GPUs) when modeling them.

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

- Convolutional layer
- Pooling layer
- Fully-connected (FC) layer

The Convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

Convolutional Layer

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let’s assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

Pooling Layer

Pooling layers, also known as down sampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

- **Max pooling:** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- **Average pooling:** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.
- While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of over fitting

Fully-Connected Layer

The name of the full-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.

This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a soft-max activation function to classify inputs appropriately, producing a probability from 0 to 1.

Long Short-Term Memory

LSTM is a type of neural network architecture that combines the long short-term memory (LSTM) network with an auto encoder. The purpose of this architecture is to perform unsupervised learning on sequential data, such as time series or natural language data.

LSTM is a type of recurrent neural network (RNN) that is commonly used for processing sequential data. It is capable of learning long-term dependencies by maintaining a memory state and selectively forgetting or adding information to this state at each time step.

An auto encoder is a type of neural network that is used for feature extraction and dimensionality reduction. It is trained to reconstruct the input data from a compressed representation, called the encoding, which captures the most important features of the input.

In the LSTM-Auto encoder architecture, the LSTM network is used to encode the sequential data into a compressed representation, and then the auto encoder is used to decode the compressed representation back into the original sequential data. The is trained to minimize the reconstruction error between the original data and the decoded data, while the LSTM network is trained to minimize the error in the compressed representation.

RNN is a type of Neural Network where the **output from the previous step is fed as input to the current step**. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is **Hidden state**, which remembers some information about a sequence.

Pseudo code of Recurrent Neural Network (RNN) Algorithm

Input: *x (t)*is taken as the input to the network at time step *t.* For example, *x1,*could be a one-hot vector corresponding to a word of a sentence.

Hidden state*: h (t)*represents a hidden state at time t and acts as "memory" of the network. *h(t)* is calculated based on the current input and the previous time step's hidden state: **h(t)** = *f*(*U* **x(t)** + *W* **h(t−1)**). The function *f* is taken to be a non-linear transformation

Weights: The RNN has input to hidden connections parameterized by a weight matrix U, hidden-to-hidden recurrent connections parameterized by a weight matrix W, and hidden-to-output connections parameterized by a weight matrix V and all these weights (*U,V,W)* are shared across time.

Output: *o (t)*illustrates the output of the network.

In the formula just put an arrow after *o (t)* which is also often subjected to non-linearity, especially when the network contains further layers downstream.

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i=y^{(t)}}$$

The LSTM architecture is useful for anomaly detection, where the model is trained on Genuine data and then used to identify anomalies in new data. The model can also be used for data compression and denoising, where the compressed representation can be used to store and transmit data more efficiently, or to remove noise from the input data.

Overall, the LSTM-Autoencoder architecture is a powerful tool for processing sequential data, and it has been successfully applied in various domains

Weighted Average

Weighted average or weighted sum ensemble is an ensemble machine learning technique that aggregates predictions from numerous models, with the weighting of each model's contribution being proportional to its capability or competency. The weighted average ensemble and the voting ensemble are related. In this manner, the averaging strategy is broadened. Each model is given a different weight, indicating the importance of that model for making predictions.

$$\textbf{Weighted Average} = \sum \textbf{(P)i\%} \times \textbf{(x)}$$

Were,

Weighted average for quantities (x)

I having weights in percentage (P)i%

PHASE V – PERFORMANCE EVALUATION

4.6 EVALUATING MODEL PERFORMANCE

A deep learning model's evaluation is essential for evaluation or validation. A machine learning model is evaluated using a variety of measures. To optimize a model based on performance, choosing the best metrics is crucial. The parameters listed below are used for evaluation,

- The Confusion Matrix
- Accuracy
- Precision
- Recall
- F1-score

Confusion Matrix

The counts of test records that the classification model correctly and incorrectly predicted are used to assess the performance of the model. The confusion matrix gives a more insightful picture of a predictive model's performance, showing which classes are forecasted correctly and incorrectly as well as the kind of errors that are being made

True Positive (TP): The cases in which one predicted yes, the actual output was also yes.

True Negative (TN): The cases in which one predicted no and the actual output was no.

False Positive (FP): The cases in which one predicted yes and the actual output was no.

False Negative (FN): The cases in which one predicted no and the actual output was yes

Actual

	NEGATIVE	POSITIVE
NEGATIVE	True Negative	False Negative
POSITIVE	False Positive	True Positive

PREDICTED

Table 3 shows the positive and negative of confusion matrices

Accuracy

Accuracy is sometimes referred to as the proportion of cases that were correctly classified to all of the cases being evaluated. Accuracy has a greatest value of 1, and a worst value of 0.

Accuracy Score = (TP + TN) / (TP + TN + FP + FN)

Precision

Both of the classes can be used to determine precision. The classifier's capability to refuse to classify a negative sample as positive is known as the precision of the negative class. The classifier's capacity to avoid classifying a positive sample as negative is known as the precision of the positive class. **Precision has a greatest value of 1, and a worst value of 0 Precision = TP / (TP + FP)**

Model Evaluation: The final step is to evaluate the performance of the model on the dataset R6. This involves calculating metrics such as precision, recall, and F1 score, as well as visualizing the results using techniques such as ROC curves or confusion matrices. The model may be refined based on the evaluation results to improve its performance.

Overall, the methodology for behavioral-based insider threat detection using deep learning on the CMU CERT synthetic insider threat dataset R6 involves preprocessing the data, training a deep learning model on the Genuine behavior data, detecting anomalous behavior in new data, and evaluating the performance of the model.

There is coding involved in the data preprocessing step for behavioral-based insider threat detection using deep learning on the CMU CERT synthetic insider threat dataset R6. The specific coding may vary depending on the programming language and deep learning framework being used.

This code loads the CMU CERT dataset R6, cleans the data by removing irrelevant or duplicate information, selects relevant features, performs feature engineering, Genuine and scales the data, encodes categorical features, and generates sequences of events for each user in the dataset. This preprocessed data can then be used to train a deep learning model for behavioral-based insider threat detection.

The input is reconstructed to the output, during the model training phase. The model is trained over Genuine or negatively labeled data i-e. Insider = 1. During the testing phase if the reconstruction error is high, it is considered as anomalous behavior. The model is tested using both positive and negative samples. The reconstruction error for insider user is very high as compared to Genuine users. A threshold value is defined to segregate Genuine and malicious behavior. If the value is higher than threshold it is considered as **‘‘Insider’’** and if the value is lower, it is considered

‘‘Genuine’’ The reconstruction error for Genuine user is low because the model is trained with Genuine data. Once the model is trained, it is then tested on mix data samples including both Genuine and malicious instances. The reconstruction error for insider user is very high as compared to Genuine

S.no	Algorithm	Accuracy	precision	F1 Score	Dataset
1	CNN	98.60%	97	98	R6.1
2	LSTM	97	94	96	R6.1

Table 4. Performance of algorithms

Out of the various performance metrics mentioned above the following are suitable for validation metrics to assess the deep learning model for insider threat detection

CHAPTER V

RESULT AND DISCUSSION

This chapter discusses detailed about the results obtained from each phase of the proposed methodology they are lobately given below

Data Collection

The dataset used is the CMU CERT synthetic insider threat dataset r6. The dataset consists of synthetic data from both Genuine and malicious insiders

The figure 4,5,6,7,8,9 refers the features of the datasets

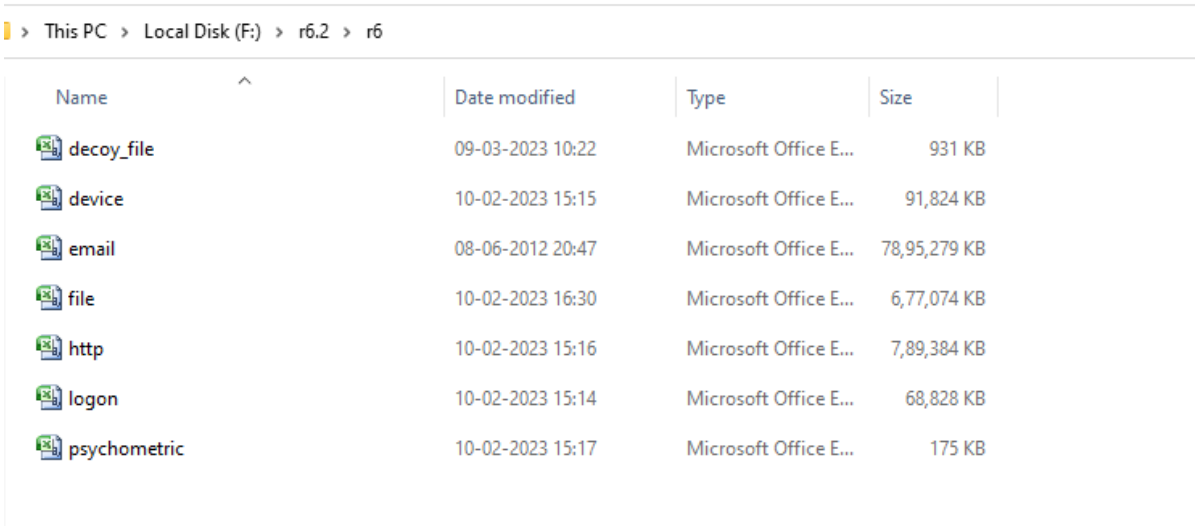


Figure: 4 CMU CERT synthetic insider threat Dataset r6.

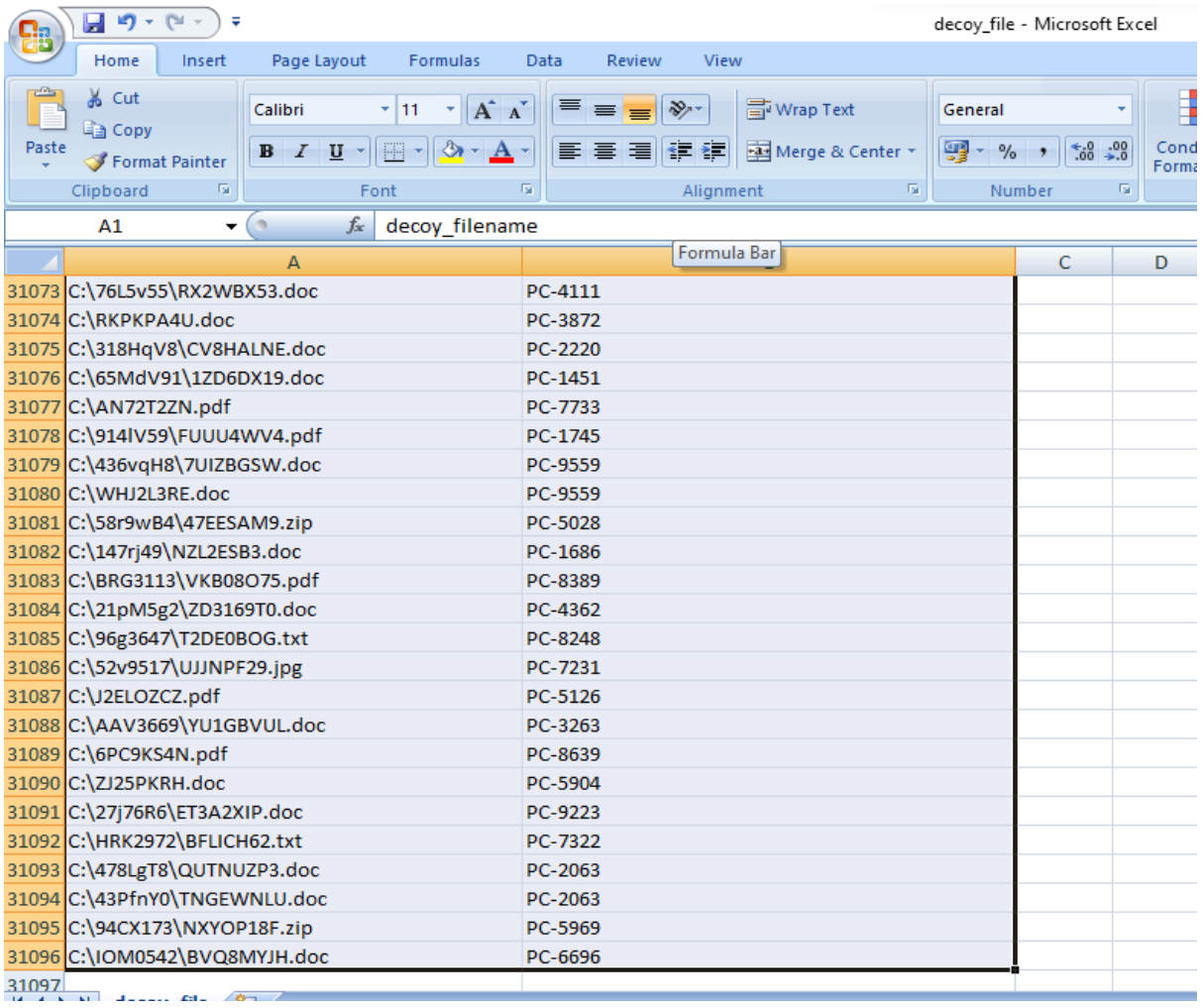
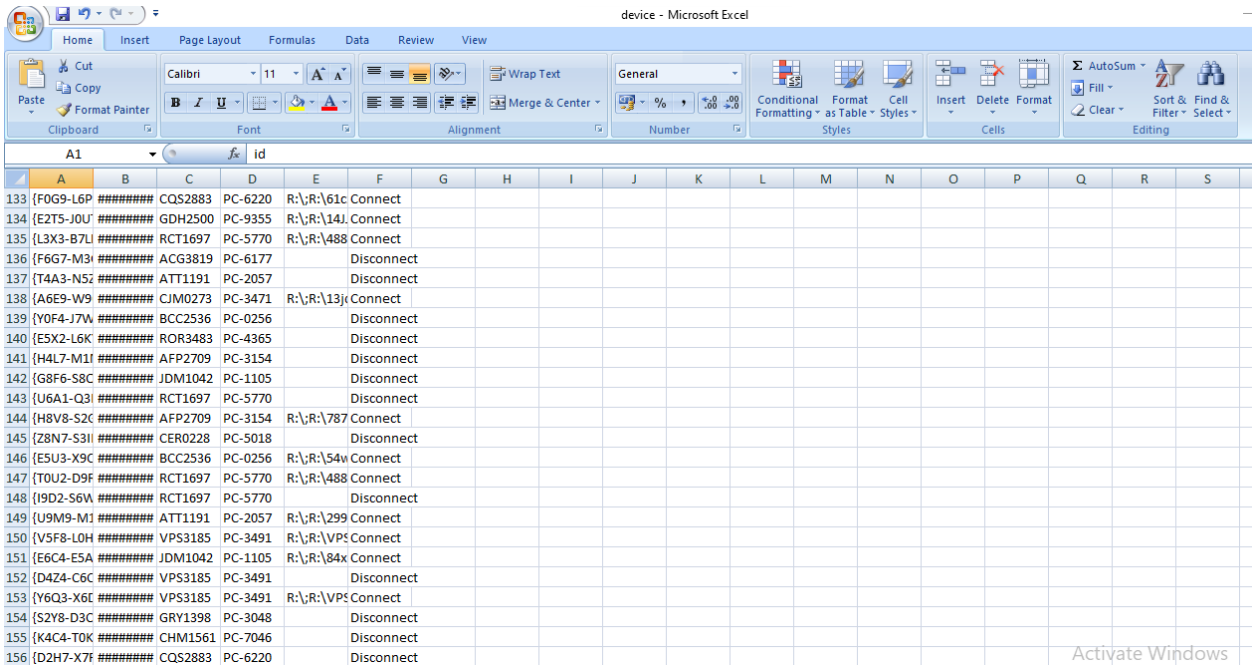
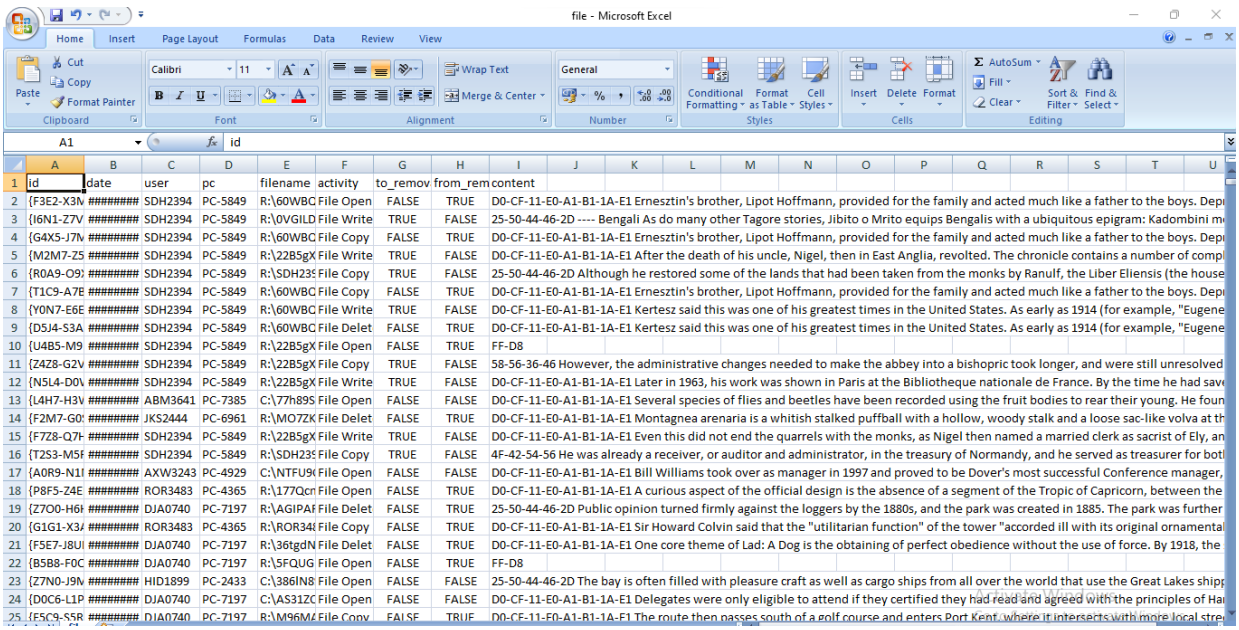


Figure.5 **Decoy file.csv**



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
133	{F0G9-L6P	########	CQS2883	PC-6220	R:\;R:\61c	Connect													
134	{E2T5-J0U	########	GDH2500	PC-9355	R:\;R:\14J	Connect													
135	{L3X3-B7L	########	RCT1697	PC-5770	R:\;R:\488	Connect													
136	{F6G7-M3(	########	ACG3819	PC-6177		Disconnect													
137	{T4A3-N5Z	########	ATT1191	PC-2057		Disconnect													
138	{A6E9-W9	########	CJM0273	PC-3471	R:\;R:\13j(	Connect													
139	{Y0F4-J7W	########	BCC2536	PC-0256		Disconnect													
140	{E5X2-L6K	########	ROR3483	PC-4365		Disconnect													
141	{H4L7-M1I	########	AFP2709	PC-3154		Disconnect													
142	{G8F6-S8C	########	JDM1042	PC-1105		Disconnect													
143	{U6A1-Q3I	########	RCT1697	PC-5770		Disconnect													
144	{H8V8-S2(	########	AFP2709	PC-3154	R:\;R:\787	Connect													
145	{Z8N7-S3I	########	CER0228	PC-5018		Disconnect													
146	{E5U3-X9C	########	BCC2536	PC-0256	R:\;R:\54w	Connect													
147	{T0U2-D9F	########	RCT1697	PC-5770	R:\;R:\488	Connect													
148	{I9D2-S6W	########	RCT1697	PC-5770		Disconnect													
149	{U9M9-M1	########	ATT1191	PC-2057	R:\;R:\299	Connect													
150	{V5F8-L0H	########	VPS3185	PC-3491	R:\;R:\VPS	Connect													
151	{E6C4-E5A	########	JDM1042	PC-1105	R:\;R:\84x	Connect													
152	{D4Z4-C6C	########	VPS3185	PC-3491		Disconnect													
153	{Y6Q3-X6[	########	VPS3185	PC-3491	R:\;R:\VPS	Connect													
154	{S2Y8-D3C	########	GRY1398	PC-3048		Disconnect													
155	{K4C4-T0K	########	CHM1561	PC-7046		Disconnect													
156	{D2H7-X7F	########	CQS2883	PC-6220		Disconnect													

Figure 6 **Device.csv**



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	id	date	user	pc	filename	activity	to_remov	from_rem	content												
2	{F3E2-X3M	########	SDH2394	PC-5849	R:\60WBC	File Open	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 Ernesztin's brother, Lipot Hoffmann, provided for the family and acted much like a father to the boys. Depi												
3	{I6N1-Z7V	########	SDH2394	PC-5849	R:\0VGILD	File Write	TRUE	FALSE	25-50-44-46-2D ---- Bengali As do many other Tagore stories, Jibito o Mrito equips Bengalis with a ubiquitous epigram: Kadombini m												
4	{G4X5-J7M	########	SDH2394	PC-5849	R:\60WBC	File Copy	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 Ernesztin's brother, Lipot Hoffmann, provided for the family and acted much like a father to the boys. Depi												
5	{M2M7-Z5	########	SDH2394	PC-5849	R:\22B5gX	File Write	TRUE	FALSE	D0-CF-11-E0-A1-B1-1A-E1 After the death of his uncle, Nigel, then in East Anglia, revolted. The chronicle contains a number of compl												
6	{R0A9-O9	########	SDH2394	PC-5849	R:\SDH23	File Copy	TRUE	FALSE	25-50-44-46-2D Although he restored some of the lands that had been taken from the monks by Ranulf, the Liber Eliensis (the house												
7	{T1C9-A7E	########	SDH2394	PC-5849	R:\60WBC	File Copy	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 Ernesztin's brother, Lipot Hoffmann, provided for the family and acted much like a father to the boys. Depi												
8	{Y0N7-E6E	########	SDH2394	PC-5849	R:\60WBC	File Write	TRUE	FALSE	D0-CF-11-E0-A1-B1-1A-E1 Kertesz said this was one of his greatest times in the United States. As early as 1914 (for example, "Eugene												
9	{D5J4-S3A	########	SDH2394	PC-5849	R:\60WBC	File Delet	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 Kertesz said this was one of his greatest times in the United States. As early as 1914 (for example, "Eugene												
10	{U4B5-M9	########	SDH2394	PC-5849	R:\22B5gX	File Open	FALSE	TRUE	FF-D8												
11	{Z4Z8-G2V	########	SDH2394	PC-5849	R:\22B5gX	File Copy	TRUE	FALSE	58-56-36-46 However, the administrative changes needed to make the abbey into a bishopric took longer, and were still unresolved												
12	{N5L4-D0V	########	SDH2394	PC-5849	R:\22B5gX	File Write	TRUE	FALSE	D0-CF-11-E0-A1-B1-1A-E1 Later in 1963, his work was shown in Paris at the Bibliotheque nationale de France. By the time he had sav												
13	{L4H7-H3V	########	ABM3641	PC-7385	C:\77h89S	File Open	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 Several species of flies and beetles have been recorded using the fruit bodies to rear their young. He foun												
14	{F2M7-G0	########	JKS2444	PC-6961	R:\MO7ZK	File Delet	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 Montagnea arenaria is a whitish stalked puffball with a hollow, woody stalk and a loose sac-like volva at th												
15	{F7Z8-Q7I	########	SDH2394	PC-5849	R:\22B5gX	File Write	TRUE	FALSE	D0-CF-11-E0-A1-B1-1A-E1 Even this did not end the quarrels with the monks, as Nigel then named a married clerk as sacrist of Ely, an												
16	{T2S3-M5F	########	SDH2394	PC-5849	R:\SDH23	File Copy	TRUE	FALSE	4F-42-54-56 He was already a receiver, or auditor and administrator, in the treasury of Normandy, and he served as treasurer for bot												
17	{A0R9-N1I	########	AXW3243	PC-4929	C:\NTFU9	File Open	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 Bill Williams took over as manager in 1997 and proved to be Dover's most successful Conference manager,												
18	{P8F5-Z4E	########	ROR3483	PC-4365	R:\177Qcn	File Open	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 A curious aspect of the official design is the absence of a segment of the Tropic of Capricorn, between the												
19	{Z7O0-H6I	########	DJA0740	PC-7197	R:\AGIPAI	File Delet	FALSE	TRUE	25-50-44-46-2D Public opinion turned firmly against the loggers by the 1880s, and the park was created in 1885. The park was further												
20	{G1G1-X3A	########	ROR3483	PC-4365	R:\ROR34	File Copy	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 Sir Howard Colvin said that the "utilitarian function" of the tower "accorded ill with its original ornamental												
21	{F5E7-J8U	########	DJA0740	PC-7197	R:\36tgdN	File Delet	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 One core theme of Lad: A Dog is the obtaining of perfect obedience without the use of force. By 1918, the												
22	{B5B8-F0C	########	DJA0740	PC-7197	R:\5FQUG	File Open	FALSE	TRUE	FF-D8												
23	{Z7N0-J9N	########	HID1899	PC-2433	C:\386IN8	File Open	FALSE	FALSE	25-50-44-46-2D The bay is often filled with pleasure craft as well as cargo ships from all over the world that use the Great Lakes shipp												
24	{D0C6-L1P	########	DJA0740	PC-7197	C:\AS312C	File Open	FALSE	FALSE	D0-CF-11-E0-A1-B1-1A-E1 Delegates were only eligible to attend if they certified they had read and agreed with the principles of Ha												
25	{E5C9-S5R	########	DJA0740	PC-7197	R:\M96M4	File Copy	FALSE	TRUE	D0-CF-11-E0-A1-B1-1A-E1 The route then passes south of a golf course and enters Port Kent, where it intersects with more local stre												

Figure 7 **FILE.CSV**

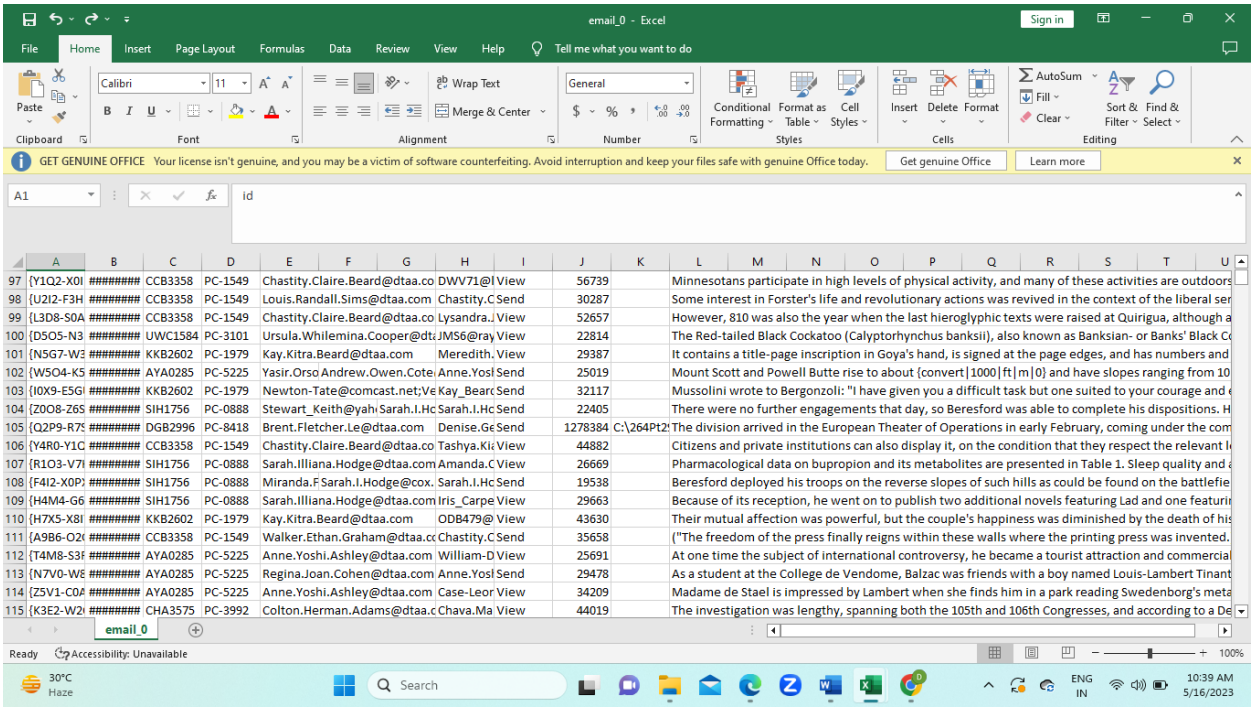


Figure 8 **EMAIL.CSV**

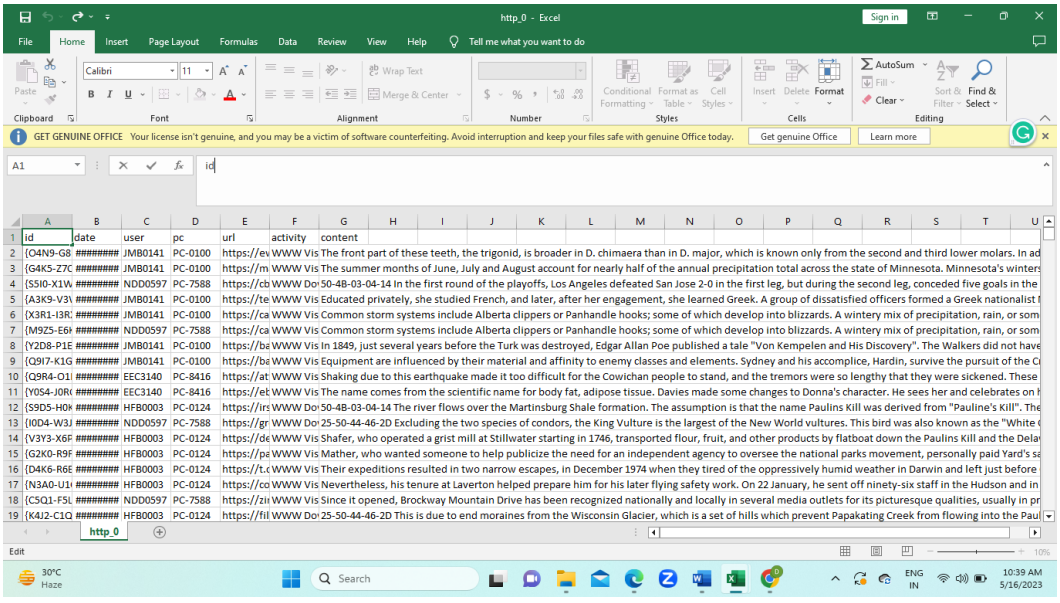


Figure .9 **HTTP.CSV**

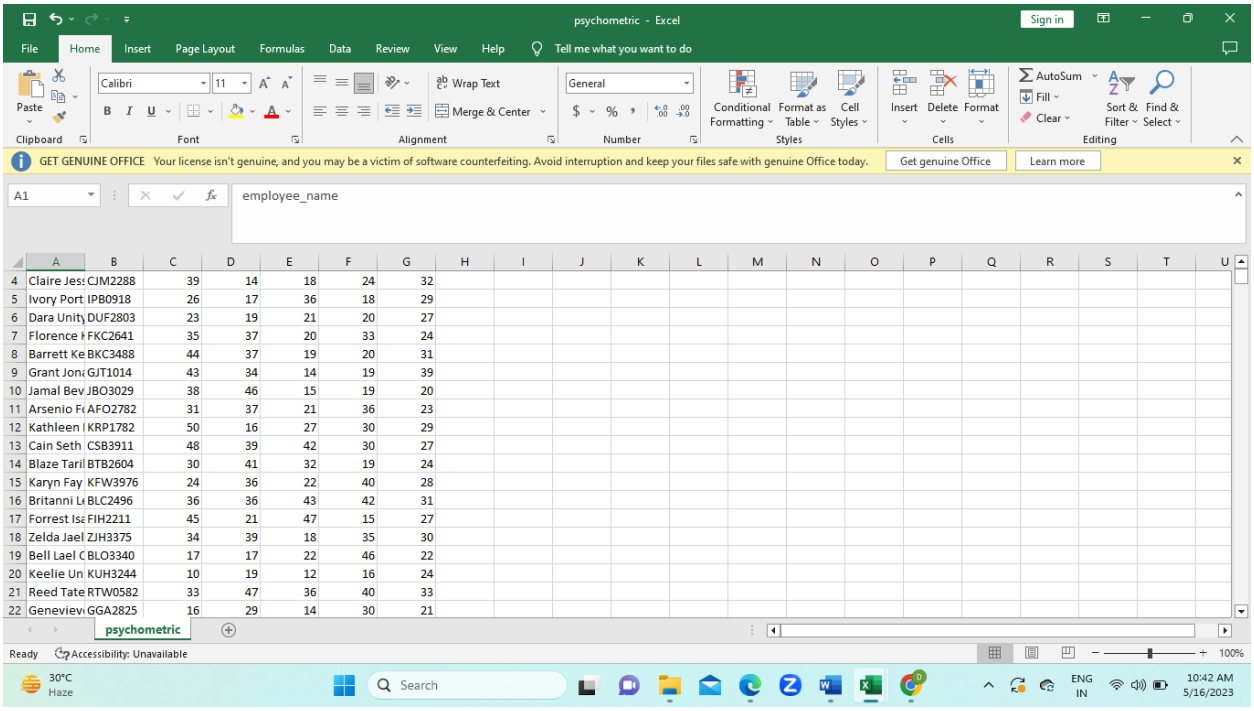


Figure 10 **PSYCHOMETRIC.CSV**

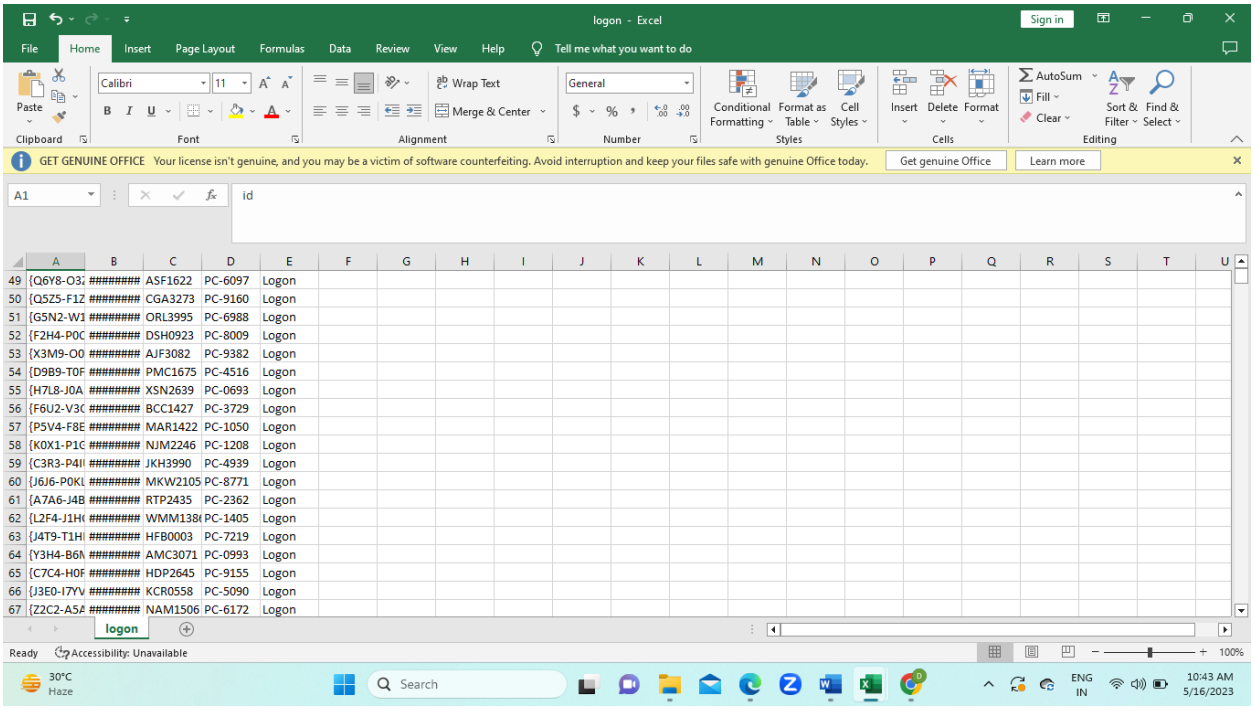


Figure 11 **LOGON.CSV**

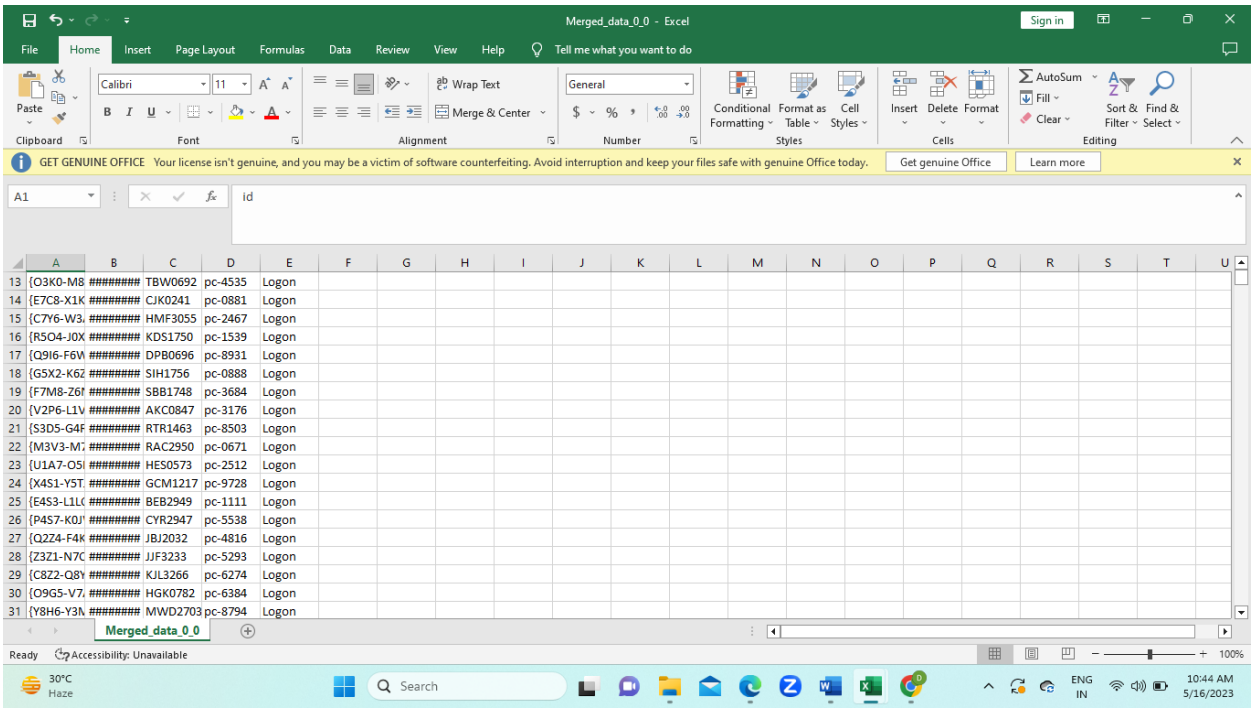


Figure 12 **MERGED 7 DATASETS INTO ONE CSV FILE**

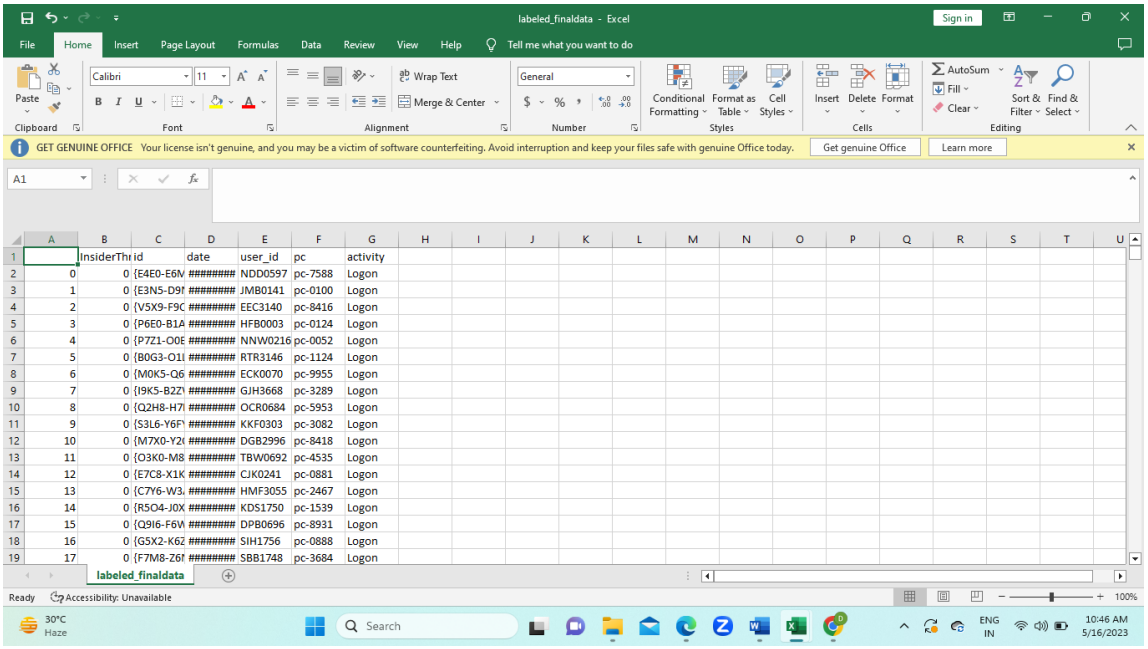


Figure 13 **ADDED THREAT IN RAW DATASET**

5.2 Data Pre-Processing in the CMU CERT synthetic insider threat dataset R6.

A. **Removing Duplicate values in the Dataset**

A dataset contains many instances of duplicate values. It is frequently discovered when using Excel to work with huge databases. Data processing will be unsuccessful if duplicate records are not eliminated. The goal of this control is to eliminate multiple records from the dataset in order to make it ready for further processing

B. Converting the Float Datatype to Int Datatype

```
memory usage: 10.7+ MB

In [5]: data.shape
Out[5]: (200012, 7)

In [6]: data.columns
Out[6]: Index(['Unnamed: 0', 'InsiderThreat', 'id', 'date', 'user_id', 'pc',
               'activity'],
              dtype='object')

In [7]: data.InsiderThreat.value_counts()
Out[7]: 0    200000
        1        12
        Name: InsiderThreat, dtype: int64

In [8]: df=pd.DataFrame(data)
        df
```

Out[8]:

	Unnamed: 0	InsiderThreat	id	date	user_id	pc	activity
0	0	0	{E4E0-E6ME15OR-2988BSJL}	01-02-2010 06:21	NDD0597	pc-7588	Logon
1	1	0	{E3N5-D9NF71OY-8015TELH}	01-02-2010 06:22	JMB0141	pc-0100	Logon
2	2	0	{V5X9-F9OJ87RB-2481RLEY}	01-02-2010 06:37	EEC3140	pc-8416	Logon
3	3	0	{P6E0-B1AN30BI-3688NCXJ}	01-02-2010 06:39	HFB0003	pc-0124	Logon
4	4	0	{P7Z1-O0EZ42UI-9603XJVG}	01-02-2010 06:41	NNW0216	pc-0052	Logon

Out[8]:

	Unnamed: 0	InsiderThreat	id	date	user_id	pc	activity
0	0	0	{E4E0-E6ME15OR-2988BSJL}	01-02-2010 06:21	NDD0597	pc-7588	Logon
1	1	0	{E3N5-D9NF71OY-8015TELH}	01-02-2010 06:22	JMB0141	pc-0100	Logon
2	2	0	{V5X9-F9OJ87RB-2481RLEY}	01-02-2010 06:37	EEC3140	pc-8416	Logon
3	3	0	{P6E0-B1AN30BI-3688NCXJ}	01-02-2010 06:39	HFB0003	pc-0124	Logon
4	4	0	{P7Z1-O0EZ42UI-9603XJVG}	01-02-2010 06:41	NNW0216	pc-0052	Logon
...	...	...	...	...	...	...	...
200007	7	1	{X5Z8-I0NU24TP-7924CGYS}	05/19/2011 21:36:42	HJB0462	PC-0761	Logoff
200008	8	1	{J7M7-L1AF74XQ-3406EXZP}	05/20/2011 19:27:59	HJB0462	PC-0761	Logon
200009	9	1	{G9V7-I2HP78OM-1408UDVG}	05/20/2011 19:42:34	HJB0462	PC-0761	Logoff
200010	10	1	{K1V5-D5FE66ER-2473NGWG}	05/20/2011 19:51:49	JRM0035	PC-0761	Logon
200011	11	1	{K1I0-V2XV30KI-3221ZOJT}	05/20/2011 20:00:51	JRM0035	PC-0761	Logoff

200012 rows × 7 columns

Figure 14 **Converting the Float Datatype to Int Datatype**

C. Label Encoding

Label encoding is the process of transforming labels into a numeric form so that they may be read by machines. Machine learning methods can then be used to determine how well those labels are functioning. It is a crucial stage in the supervised learning pre-processing of the structured dataset.

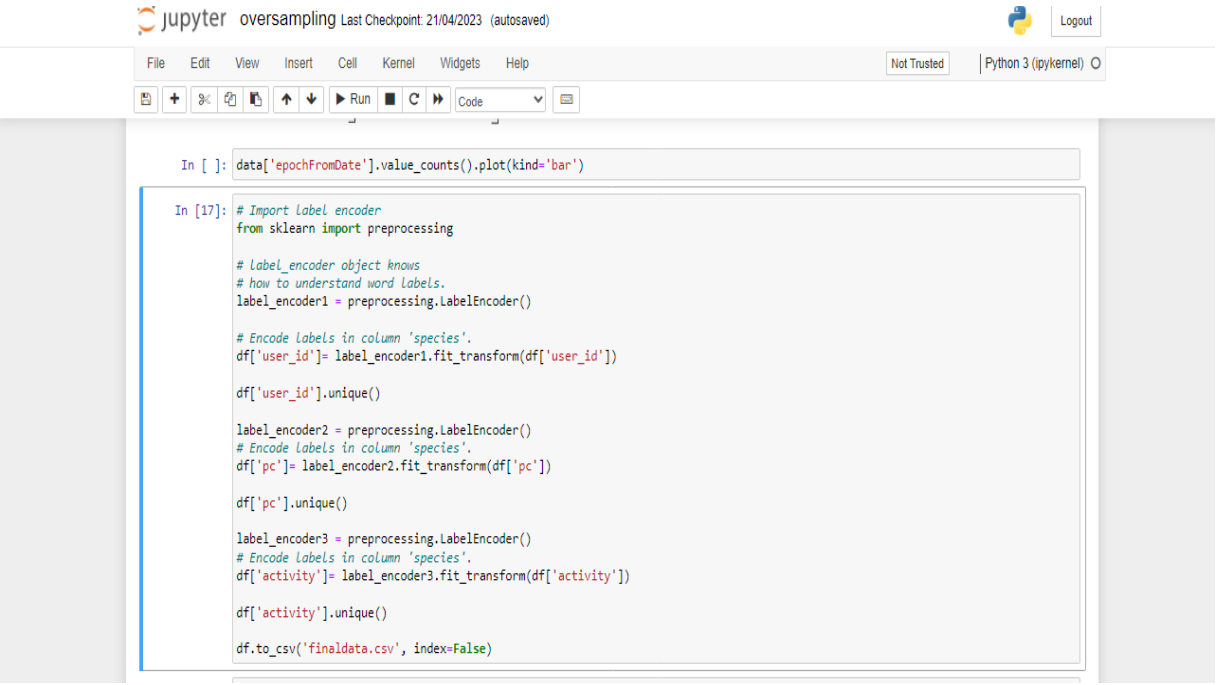


Figure 15 LABEL ENCODING

D. Oversampling

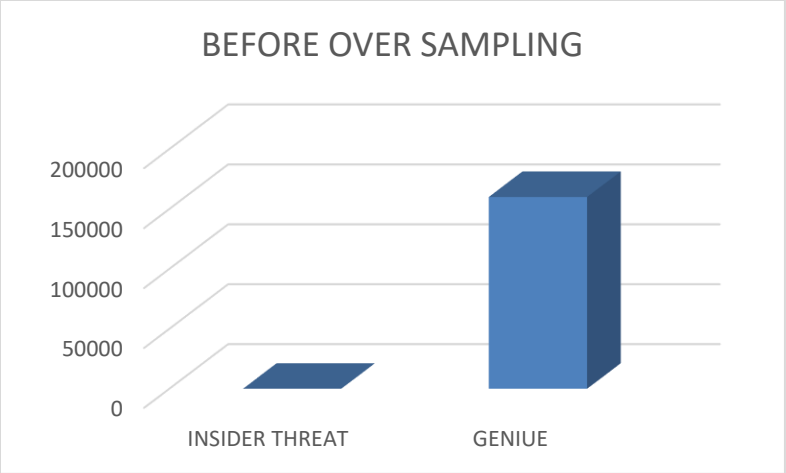
Random oversampling includes adding replacement samples drawn at random from the minority class to the training dataset

BEFORE OVERSAMPLING

```
In [20]: # # SMOTE to solve class-imbalance
         # # Before Sampling Label Value Count
         pd.Series(y_train).value_counts()

Out[20]: 0    159998
         1        11
         Name: InsiderThreat, dtype: int64
```

AFTER OVERSAMPLING

```
In [22]: from imblearn.over_sampling import SMOTE
         smote=SMOTE()

         X_train, y_train = smote.fit_resample(X_train, y_train)

         # # After Sampling insiderthreat Value Count

         pd.Series(y_train).value_counts()

Out[22]: 0    159998
         1    159998
         Name: InsiderThreat, dtype: int64
```



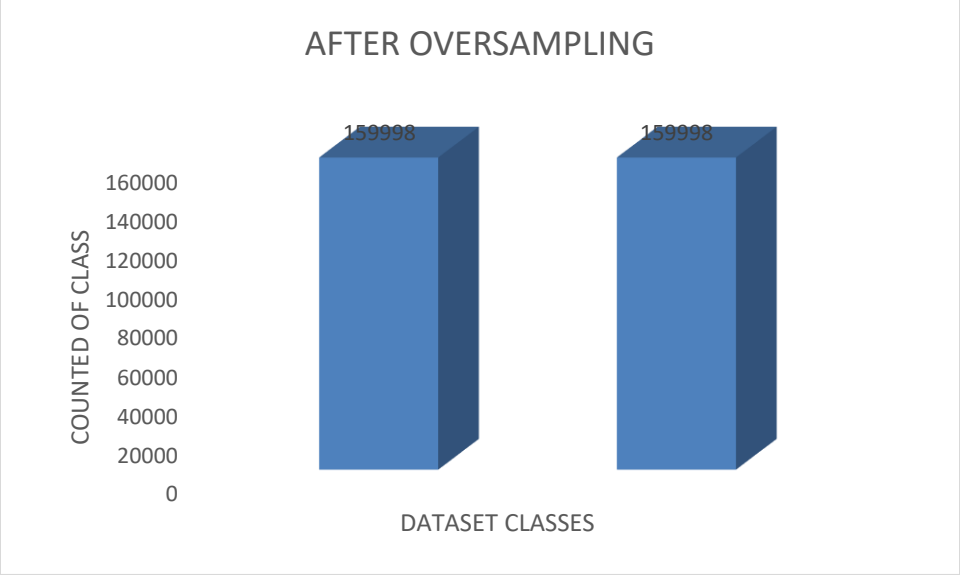


Figure 16 **OVERSAMPLING**

E. Splitting of Dataset into Train and Test CMU CERT synthetic insider threat Dataset r6.

To assess how well our machine learning model works, we must divide a dataset into train and test sets. The statistics of the train set are known and are utilized to fit the model. The test data set, which is the second set, is utilized just for predictions

```
In [25]: X_train, y_train

Out[25]: (        user_id    pc  activity  epochFromDate
          0          3125  2541         0   1.278003e+09
          1          2367  2564         1   1.263829e+09
          2          1248  1862         0   1.278010e+09
          3          3051   547         0   1.270141e+09
          4           203    65         1   1.264162e+09
          ...         ...   ...       ...            ...
          319991      926     0         0   1.321564e+09
          319992     2003     0         0   1.305921e+09
          319993     1850     0         0   1.305892e+09
          319994     1593     0         0   1.305920e+09
          319995     1381     0         0   1.307694e+09

          [319996 rows x 4 columns],
          0         0
          1         0
          2         0
          3         0
          4         0
```

Figure 17 X_TAIN Y_TRAIN

```
In [26]: combined = pd.concat([X_train, y_train], axis=1, join='inner')
         combined
```

Out[26]:

	user_id	pc	activity	epochFromDate	InsiderThreat
0	3125	2541	0	1.278003e+09	0
1	2367	2564	1	1.263829e+09	0
2	1248	1862	0	1.278010e+09	0
3	3051	547	0	1.270141e+09	0
4	203	65	1	1.264162e+09	0
...	...	...	...	...	...
319991	926	0	0	1.321564e+09	1
319992	2003	0	0	1.305921e+09	1
319993	1850	0	0	1.305892e+09	1
319994	1593	0	0	1.305920e+09	1
319995	1381	0	0	1.307694e+09	1

319996 rows × 5 columns

Figure 18 COMBINED X_TAIN AND Y_TRAIN USING INNNER JOIN

F. BALANCING DATA

Balance Dataset

```
In [27]: non_fraud = combined[combined['InsiderThreat']==0]
         fraud = combined[combined['InsiderThreat']==1]

In [28]: non_fraud.shape, fraud.shape
Out[28]: ((159998, 5), (159998, 5))

In [29]: non_fraud = non_fraud.sample(fraud.shape[0])
         non_fraud.shape
Out[29]: (159998, 5)

In [30]: data = fraud.append(non_fraud, ignore_index=True)
         data
```

In [30]:
```
data = fraud.append(non_fraud, ignore_index=True)
data
```

Out[30]:

	user_id	pc	activity	epochFromDate	InsiderThreat
0	1593	0	0	1.305841e+09	1
1	2003	0	0	1.305922e+09	1
2	848	1	0	1.306906e+09	1
3	848	1	1	1.320188e+09	1
4	848	1	1	1.306904e+09	1
...	...	...	...	...	...
319991	2618	3411	1	1.291207e+09	0
319992	2989	2175	1	1.263544e+09	0
319993	2702	517	0	1.264273e+09	0
319994	1855	4036	1	1.263442e+09	0
319995	1028	4226	1	1.264665e+09	0

319996 rows × 5 columns

In [31]:
```
data['InsiderThreat'].value_counts()
```

Out[31]:
```
1    159998
0    159998
Name: InsiderThreat, dtype: int64
```

Figure 19 BALANCING DATA

G. Standard Scaler

```
In [32]: X = data.drop('InsiderThreat', axis = 1)
         y = data['InsiderThreat']

In [33]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0, stratify = y)

In [34]: X_train.shape, X_test.shape
Out[34]: ((255996, 4), (64000, 4))

In [35]: from sklearn.preprocessing import StandardScaler

In [36]: scaler = StandardScaler()
         X_train = scaler.fit_transform(X_train)
         X_test = scaler.transform(X_test)

In [37]: y_train = y_train.to_numpy()
         y_test = y_test.to_numpy()

In [38]: X_train.shape
Out[38]: (255996, 4)

In [39]: X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
         X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

In [40]: X_train.shape, X_test.shape
Out[40]: ((255996, 4, 1), (64000, 4, 1))
```

Figure 20 Standard Scaler

H. MODEL CLASSIFICATION IN DATASET R6

Build CNN

```
In [41]: import tensorflow as tf
         from tensorflow import keras
         from tensorflow.keras import Sequential
         from tensorflow.keras.layers import Flatten, Dense, Dropout, BatchNormalization
         from tensorflow.keras.layers import Conv1D, MaxPool1D
         from tensorflow.keras.optimizers import Adam
         print(tf.__version__)

         2.8.0

In [42]: epochs = 10
         model = Sequential()
         model.add(Conv1D(16, 2, activation='relu', input_shape = X_train[0].shape))
         model.add(BatchNormalization())
         model.add(Dropout(0.2))

         model.add(Conv1D(32, 2, activation='relu'))
         model.add(BatchNormalization())
         model.add(Dropout(0.5))

         model.add(Flatten())
         model.add(Dense(64, activation='relu'))
         model.add(Dropout(0.5))

         model.add(Dense(1, activation='sigmoid'))
```

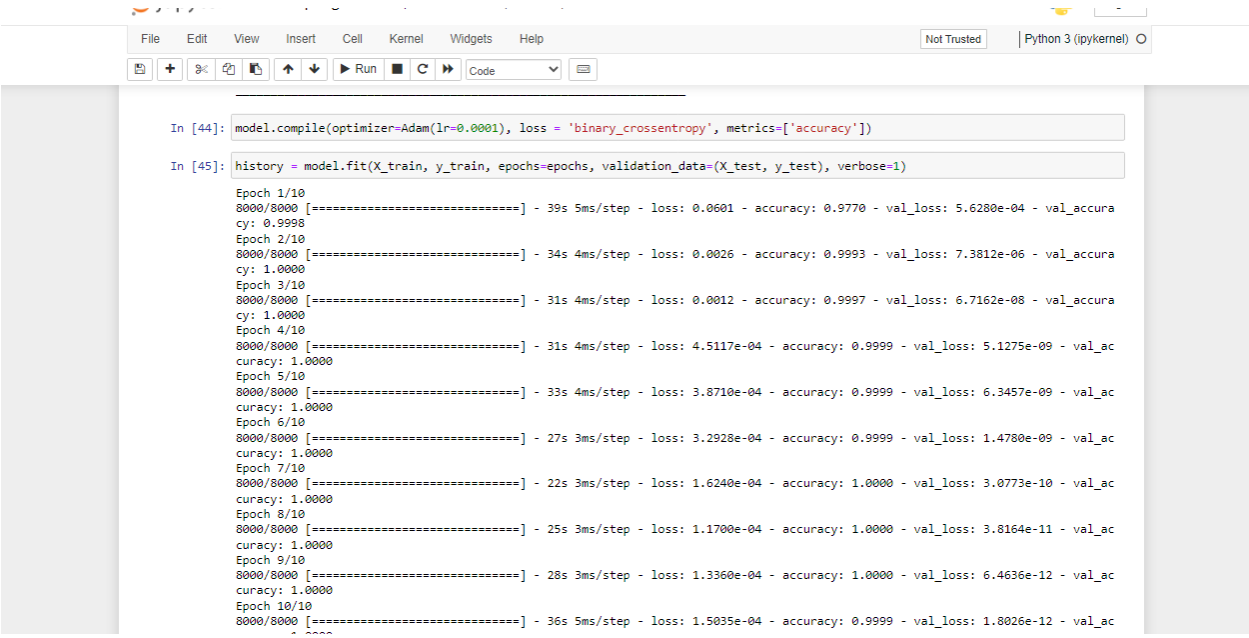
Figure 21 BUILDING CNN

I. MODEL SUMMARY

```
In [43]: model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 3, 16)             48

 batch_normalization (BatchN  (None, 3, 16)            64
 ormalization)

 dropout (Dropout)           (None, 3, 16)             0

 conv1d_1 (Conv1D)           (None, 2, 32)             1056

 batch_normalization_1 (Batc  (None, 2, 32)            128
 hNormalization)

 dropout_1 (Dropout)         (None, 2, 32)             0

 flatten (Flatten)           (None, 64)                0

 dense (Dense)               (None, 64)                4160

 dropout_2 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 5,521
Trainable params: 5,425
Non-trainable params: 96
_________________________________________________________________
```

Figure 22 MODEL SUMMARY

J. MODEL ACCURACY



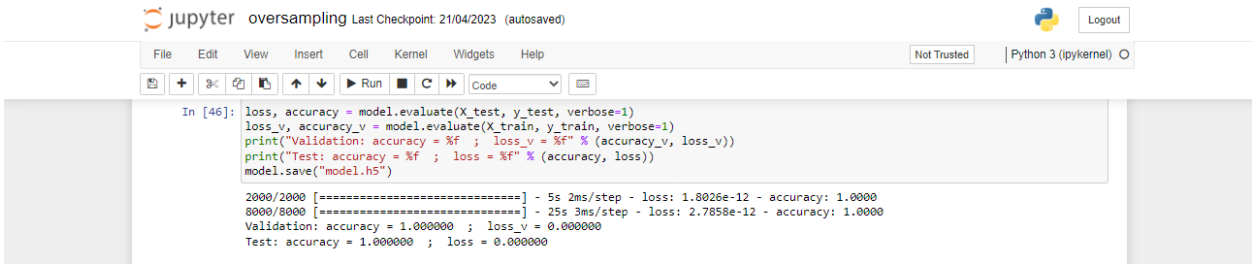


Figure 24 MODEL ACCURACY

K. PLOT VISUALIZING

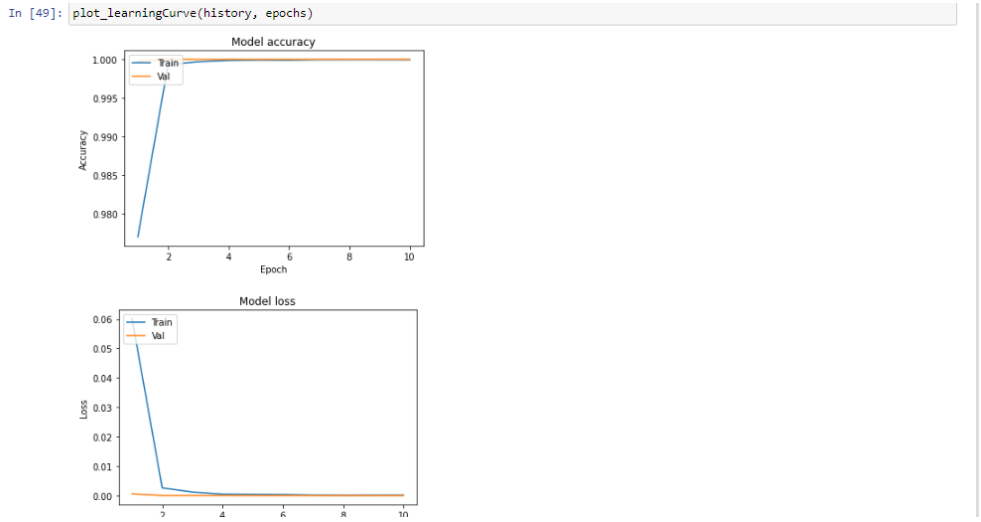


Figure 25 PLOT VISUALIZING

BUILDING LSTM

a) Min Max Scaler

Transform features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g., between zero and one

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: dataset_train = pd.read_csv('D:/Pre-processed_Datasets_codes-20230321T052125Z-001/Pre-processed_Datasets_codes/labeled_finaldata
        dataset_train.head()
```

Out[2]:

	Unnamed: 0	InsiderThreat	id	date	user_id	pc	activity
0	0	0	{E4E0-E6ME15OR-2988BSJL}	01-02-2010 06:21	NDD0597	pc-7588	Logon
1	1	0	{E3N5-D9NF71OY-8015TELH}	01-02-2010 06:22	JMB0141	pc-0100	Logon
2	2	0	{V5X9-F9OJ87RB-2481RLEY}	01-02-2010 06:37	EEC3140	pc-8416	Logon
3	3	0	{P6E0-B1AN30BI-3688NCXJ}	01-02-2010 06:39	HFB0003	pc-0124	Logon
4	4	0	{P7Z1-O0EZ42UI-9603XJVG}	01-02-2010 06:41	NNW0216	pc-0052	Logon

```
In [3]: dataset_train.shape
Out[3]: (200012, 7)

In [4]: training_set = dataset_train.iloc[:,1:2].values

In [5]: from sklearn.preprocessing import MinMaxScaler
        sc = MinMaxScaler(feature_range=(0,1))
        training_set_scaled = sc.fit_transform(training_set)
```

Figure 26 MINMAXSCALER

b) Splitting of Dataset into Train and Test

To assess how well our machine learning model works, we must divide a dataset into train and test sets. The statistics of the train set are known and are utilized to fit the model. The test data set, which is the second set, is utilized just for predictions

```
In [6]: X_train = []
        y_train = []
        for i in range(60,1258):
            X_train.append(training_set_scaled[i-60:i, 0])
            y_train.append(training_set_scaled[i, 0])

        X_train, y_train = np.array(X_train), np.array(y_train)

In [7]: X_train.shape
Out[7]: (1198, 60)

In [8]: y_train.shape
Out[8]: (1198,)

In [9]: y_train.shape# reshape to 3 dimesnsions - No of stock prices, No of timestamps, No of indicatores
        X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

In [10]: X_train.shape
Out[10]: (1198, 60, 1)
```

Figure 27 Splitting of Dataset into Train and Test

c) Budling RNN and Importing

Innitializing the RNN

```
In [12]: regressor = Sequential()

In [13]: regressor.add(LSTM(50, return_sequences=True, input_shape = (X_train.shape[1], 1)))
         regressor.add(Dropout(0.2))

In [14]: regressor.add(LSTM(50, return_sequences=True))
         regressor.add(Dropout(0.2))

In [15]: regressor.add(LSTM(50, return_sequences=True))
         regressor.add(Dropout(0.2))

In [16]: regressor.add(LSTM(50, return_sequences=False))
         regressor.add(Dropout(0.2))

In [17]: regressor.add(Dense(1))

In [18]: regressor.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 50)            10400

 dropout (Dropout)           (None, 60, 50)            0

 lstm_1 (LSTM)               (None, 60, 50)            20200

 dropout_1 (Dropout)         (None, 60, 50)            0

 lstm_2 (LSTM)               (None, 60, 50)            20200

 dropout_2 (Dropout)         (None, 60, 50)            0

 lstm_3 (LSTM)               (None, 50)                20200

 dropout_3 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
_________________________________________________________________

In [67]: from tensorflow.keras.optimizers import Adam
```

Figure 27 MODEL IMPORTING

d) MODEL SUMMARY

Fitting the RNN to the training set

```
In [76]: regressor.fit(X_train, y_train, epochs = 10, batch_size = 18)

Epoch 1/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 2/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 3/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 4/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 5/10
67/67 [==============================] - 2s 33ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 6/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 7/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 8/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 9/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 10/10
67/67 [==============================] - 2s 32ms/step - loss: 0.0000e+00 - accuracy: 1.0000

Out[76]: <keras.callbacks.History at 0x20cac9dbfd0>
```

Figure 28 MODEL ACCURACY

e) MAKING THE PREDICTIONS

Making the predictions and visualizing the results

```
In [22]: dataset_test = pd.read_csv('D:/Pre-processed_Datasets_codes-20230321T052125Z-001/Pre-processed_Datasets_codes/labeled_finaldata.c
         dataset_test.head()
```

Out[22]:

	Unnamed: 0	InsiderThreat	id	date	user_id	pc	activity
0	0	0	{E4E0-E6ME15OR-2988BSJL}	01-02-2010 06:21	NDD0597	pc-7588	Logon
1	1	0	{E3N5-D9NF71OY-8015TELH}	01-02-2010 06:22	JMB0141	pc-0100	Logon
2	2	0	{V5X9-F9OJ87RB-2481RLEY}	01-02-2010 06:37	EEC3140	pc-8416	Logon
3	3	0	{P6E0-B1AN30BI-3688NCXJ}	01-02-2010 06:39	HFB0003	pc-0124	Logon
4	4	0	{P7Z1-O0EZ42UI-9603XJVG}	01-02-2010 06:41	NNW0216	pc-0052	Logon

```
In [23]: dataset_test.shape
Out[23]: (200012, 7)

In [24]: real_stock_price = dataset_test.iloc[:,1:2].values

In [26]: dataset_total = pd.concat((dataset_train['InsiderThreat'], dataset_test['InsiderThreat']), axis = 0)

In [27]: dataset_total.shape
Out[27]: (400024,)

In [28]: inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values

In [29]: inputs.shape
```

Figure 29 **THE PREDICTIONS**

f) RESHAPING THE DATA

```
In [30]: inputs = inputs.reshape(-1,1)

In [31]: inputs.shape
Out[31]: (200072, 1)

In [32]: inputs = sc.transform(inputs)

In [33]: X_test = []
         for i in range(60, 80):
             X_test.append(inputs[i-60:i, 0])

         X_test = np.array(X_test)

In [34]: X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

In [35]: predicted_stock_price = regressor.predict(X_test)

In [36]: predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Figure 30 **RESHAPING THE DATA**

g) VISUALIZING THE RESULTS

Visualizing the results

```
In [40]: plt.plot(real_stock_price, color="red", label = "activity")
         plt.plot(predicted_stock_price, color="blue", label = "Predicted insiderthreat")
         plt.title("Predicted insiderthreat")
         plt.xlabel("activity")
         plt.ylabel("insiderthreat")
         plt.legend()
Out[40]: <matplotlib.legend.Legend at 0x20c8f68ba30>
```

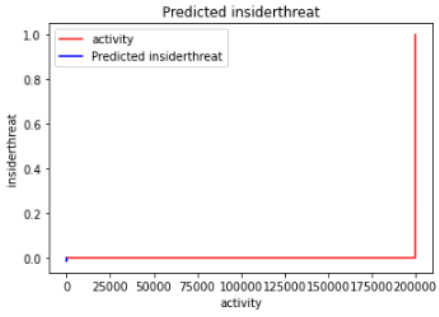


Figure 31 VISUALIZING THE RESULTS

AVERAGE ACCURACY

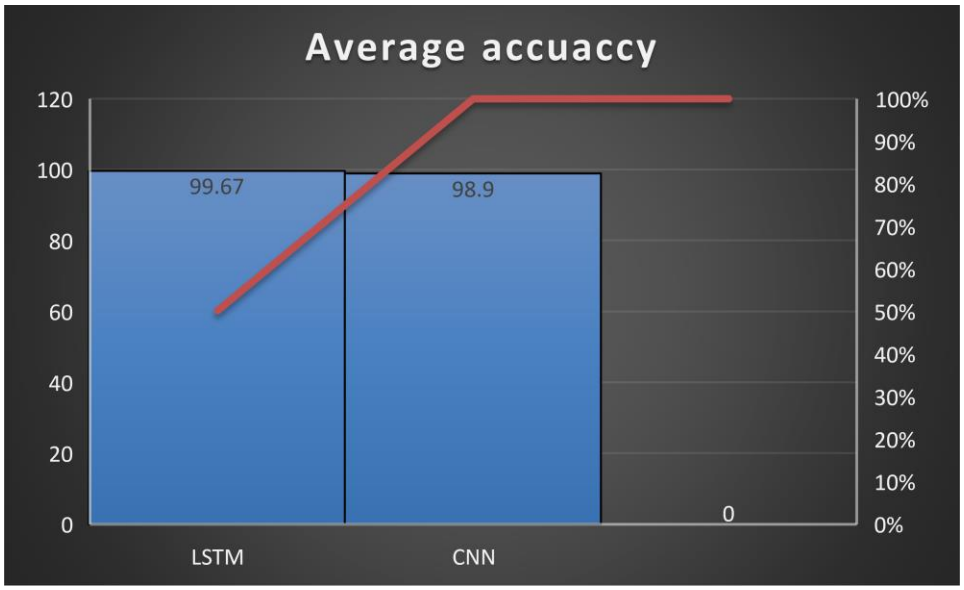


Figure 32 AVERAGE ACCURACY

CHAPTER VI

Conclusion and Future Scope

We study the insider threat problem and identified that mitigating this problem is a challenging task. Nowadays, mitigation against insider threats is achieved by implementing user access controls, user behavior monitoring, and physical security controls.

In this work, a Deep Learning based insider attack detection scheme is presented. The main aim behind the development of this scheme is its application on user technical data within an organization with low processing and memory requirements. Moreover, the developed system is simple and adaptable with minimum domain knowledge requirements.

In this project the deep learning models are derived to detect the insider threat using C NN and LSTM as developed models The model is trained and tested on CMU CERT R6. The platform used for the evaluation of the scheme is Anaconda 2021.12, Build: py37_0 using Jupyter Notebook 5.7.4. Moreover, the performance of the proposed algorithm has been compared to other well-known techniques i.e., CNN, LSTM-RNN, the comparison showed that our novel approach produces relatively good accuracy (98.60%), precision (98%)

In order to create a robust Insider detection system, we need to create more diverse insider threat scenarios, as there is a lack of publicly available threat scenarios. This will help us in solving insider problems with more creativity, high quality, and accuracy

Based on the CNN and LSTM, in future the cloud insider threat detection model will be enhanced with applying other deep neural network algorithm to provide the real time problems

Outcome: Based on the comparative analysis of CNN and LSTM Deep learning models ,CNN model out performs with highest all 98.60% in detecting malicious insider In an outcome, based on significant comparative analysis CNN and LSTM it is recommended that CNN models performs better with accuracy for malicious insider threat detection

Reference

1. Insider Report 2018, CA Technol., New York, NY, USA, 2018.

2. Insider Threat Report 2019, CA Technol., San Jose, CA, USA, 2019.

3. S. R. Band, D. M. Cappelli, L. F. Fischer, A. P. Moore, E. D. Shaw, and R. F. Trzeciak, ''Comparing insider IT sabotage and espionage: A model-based analysis,'' Carnegie Mellon Univ., Softw. Eng. Inst., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2006TR-026, 2006.

4. P. Chattopadhyay, L. Wang, and Y.-P. Tan, ''Scenario-based insider threat detection from cyber activities,'' IEEE Trans. Comput. Social Syst., vol. 5, no. 3, pp. 660–675, Sep. 2018.

5. F. Yuan, Y. Cao, Y. Shang, Y. Liu, J. Tan, and B. Fang, ''Insider threat detection with deep neural network,'' in Proc. Int. Conf. Comput. Sci. Cham, Switzerland: Springer, 2018.

6. W. Jiang, Y. Tian, W. Liu, and W. Liu, ''An insider threat detection method based on user behavior analysis,'' in Proc. Int. Conf. Intell. Inf. Process. Amsterdam, The Netherlands: International Federation for Information Processing, 2018, pp. 421–429.

7. C. Liu, Y. Zhong, and Y. Wang, ''Improved detection of user malicious behavior through log mining based on IHMM,'' in Proc. 5th Int. Conf. Syst. Informant. (ICSAI), Nov. 2018, pp. 1193–1198.

8. Z. Zamanian, A. Feizollah, N. B. Anuar, L. B. M. Kiah, K. Srikanth, and S. Kumar, ''User profiling in anomaly detection of authorization logs,'' in Computational Science and Technology. Singapore: Springer. 2019.

9.] J. Jiang, J. Chen, K.-K.-R. Choo, K. Liu, C. Liu, M. Yu, and P. Mohapatra, ''Prediction and detection of malicious insiders' motivation based on sentiment profile on webpages and emails,'' in Proc. MILCOM, Oct. 2018, pp. 1–6.

10. D. Zhang, Y. Zheng, Y. Wen, Y. Xu, J. Wang, Y. Yu, and D. Meng, ''Rolebased log analysis applying deep learning for insider threat detection,'' in Proc. Search, Toronto, ON, Canada, Jan. 2018, pp. 18–20.

11. K. A. Tabash and J. Happa, ''Insider-threat detection using Gaussian mixture models and sensitivity profiles,'' Compute Secure., vol. 77, pp. 838–859, Aug. 2018.

12. O. Lo, W. J. Buchanan, P. Griffiths, and R. Macfarlane, ''Distance measurement methods for improved insider threat detection,'' Secure. Common. Netw., vol. 2018, pp. 1–18, Jan. 2018

13. A. Gamachchi, L. Sun, and S. Boztas, ''Graph-based framework for malicious insider threat detection,'' in Proc. 50th Hawaii Int. Conf. Syst. Sci. (HICSS), 2017, p. 10.

14. F. Meng, F. Lou, Y. Fu, and Z. Tian, ‘‘Deep learning-based attribute classification insider threat detection for data security,’’ in Proc. IEEE 3rd Int. Conf. Data Sci. Cyberspace, Jun. 2018, pp. 576–581. [15] A. Shaghaghi, S. S. Kanhere, M. A. Kaafar, E. Bertino, and S. Jha, ‘‘Gargoyle: A network-based insider attack resilient framework for organizations,’’ in Proc. IEEE 43rd Conf. Local Compute.Netw. (LCN), Oct. 2018, pp. 553–561.

15. D. Patil and B. Meshram, ‘‘Network packet analysis for detecting malicious insider,’’ in Proc. 3rd Int. Conf. Converg. Technol., 2018, pp. 1–8.

16. L. Liu, O. De Vel, C. Chen, J. Zhang, and Y. Xiang, ‘‘Anomaly-based insider threat detection using deep autoencoders,’’ in Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW), Nov. 2018, pp. 39–48. [18]

17. L. Lin, S. Zhong, C. Jia, and K. Chen, ‘‘Insider threat detection based on deep belief network feature representation,’’ in Proc. Int. Conf. Green Information. (ICGI), Aug. 2017, pp. 54–59

18. J. Lu and R. K. Wong, ‘‘Insider threat detection with long shortterm memory,’’ in Proc. ACSW, Sydney, NSW, Australia, Jan. 2019, pp. 1–10.

19. J. Wang et al., ‘‘Learning correlation graph and anomalous employee behavior for insider threat detection,’’ in Proc. 21st Int. Conf. Inf. Fusion (FUSION), Jul. 2018, pp. 1–7.

20. X. Wang, Q. Tan, J. Shi, S. Su, and M. Wang, ‘‘Insider threat detection using characterizing user behavior,’’ in Proc. IEEE 3rd Int. Conf. Data Sci. Cyberspace, Jun. 2018, pp. 476–482. VOLUME 9, 2021

Appendix

8.1 Sample Coding of the r6 version of the CMU CERT synthetic insider threat dataset

MERGING DATA

```
# import pandas as pd
import pandas as pd

# list of strings
columns = ['id', 'date','user_id','pc','activity']

# Calling DataFrame constructor on list
data = pd.read_csv("Merged_data_0_0.csv")
print(data)

data.columns
data['InsiderThreat']=0
# show the dataframe
data = data[list(('InsiderThreat','id', 'date','user_id','pc','activity'))]
data
# Calling DataFrame constructor on list
iData = pd.read_csv("InsiderData.csv")
print(iData)
iData['InsiderThreat']=1
# show the dataframe
iData = iData[list(('InsiderThreat','id', 'date','user_id','pc','activity'))]

iData.columns
iData.columns

iData
```

```
labeled_insiderThreat_data=pd.concat([data, iData], axis=0)

labeled_insiderThreat_data.to_csv('labeled_finaldata.csv')
```

Importing Libraries:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score, precision_score, recall_score, f1_score
get_ipython().system('pip install joblib ')
```

Reading the Dataset

```
data = pd.read_csv('D:/Pre-processed_Datasets_codes-20230321T052125Z-001/Pre-processed_Datasets_codes/labeled_finaldata.csv')

data.info()
data.shape
data.columns
data.InsiderThreat.value_counts()
df=pd.DataFrame(data)
df

df.info()
''' DATA CLEANSING'''
''' TREATING DATE COLUMN to get EPOCH '''
# convert the date field into Unix Epoch Time
import datetime as dt
```

```
import datetime
# convert it to milliseconds from epoch
#combined['date'] = pd.to_datetime(combined['date'])
map(lambda s: s[:-3], df['date'])
print(df.dtypes)
df['date'] = pd.to_datetime(df['date'], dayfirst=True, errors='coerce')
df['date'] = df['date'].astype('datetime64[ns]')

#combined['date'] = combined['date'].astype('datetime64[ns]')
df['epochFromDate']=(df['date'] - dt.datetime(1970,1,1)).dt.total_seconds()
del df['date']
#combined['epochFromDate']=combined['date'].astype('int64') // 1e9
print(df.info())
df=df.drop(['id','Unnamed: 0'],axis=1)
df.columns
fig, ax1 = plt.subplots(1, 1, figsize= (10, 5))
data['InsiderThreat'].value_counts().plot(kind='bar', ax=ax1)

data['pc'].value_counts().plot(kind='bar')
data['activity'].value_counts().plot(kind='bar')

data['epochFromDate'].value_counts().plot(kind='bar')
```

Import label encoder

```
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder1 = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
```

```
df['user_id']= label_encoder1.fit_transform(df['user_id'])

df['user_id'].unique()

label_encoder2 = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
df['pc']= label_encoder2.fit_transform(df['pc'])

df['pc'].unique()

label_encoder3 = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
df['activity']= label_encoder3.fit_transform(df['activity'])

df['activity'].unique()

df.to_csv('finaldata.csv', index=False)
```

Splitting of Dataset into Train and Test

```
X = df.drop(['InsiderThreat'],axis=1)
y = df['InsiderThreat']
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size = 0.8, test_size = 0.2, random_state
= 0) #shuffle=False

df.columns
```

SMOTE to solve class-imbalance

Before Sampling Label Value Count

```
pd.Series(y_train).value_counts(
df
from imblearn.over_sampling import SMOTE
```

```
smote=SMOTE()
X_train, y_train = smote.fit_resample(X_train, y_train)
```

After Sampling insider threat Value Count

```
pd.Series(y_train).value_counts()
X_train
y_train
X_train, y_train
combined = pd.concat([X_train, y_train], axis=1, join='inner')
combined
```

Balance Dataset

```
non_fraud = combined[combined['InsiderThreat']==0]
fraud = combined[combined['InsiderThreat']==1]
non_fraud.shape, fraud.shape
non_fraud = non_fraud.sample(fraud.shape[0])
non_fraud.shape


data = fraud.append(non_fraud, ignore_index=True)
data
data['InsiderThreat'].value_counts()
X = data.drop('InsiderThreat', axis = 1)
y = data['InsiderThreat']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0, stratify = y)
X_train.shape, X_test.shape
from sklearn.preprocessing
import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
X_train.shape
```

```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
X_train.shape, X_test.shape
```

Build CNN

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout, BatchGenuineization
from tensorflow.keras.layers import Conv1D, MaxPool1D
from tensorflow.keras.optimizers import Adam
print(tf.__version__)

epochs = 10
model = Sequential()
model.add(Conv1D(16, 2, activation='relu', input_shape = X_train[0].shape))
model.add(BatchGenuineization())
model.add(Dropout(0.2))

model.add(Conv1D(32, 2, activation='relu'))
model.add(BatchGenuineization())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
model.summary()
model.compile(optimizer=Adam(lr=0.0001), loss = 'binary_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
loss, accuracy = model.evaluate(X_test, y_test, verbose=1)
loss_v, accuracy_v = model.evaluate(X_train, y_train, verbose=1)
print("Validation: accuracy = %f  ;  loss_v = %f" % (accuracy_v, loss_v))
print("Test: accuracy = %f  ;  loss = %f" % (accuracy, loss))
model.save("model.h5")

import matplotlib.pyplot as plt
def plot_learningCurve(history, epoch):
  # Plot training & validation accuracy values
epoch_range = range(1, epoch+1)
plt.plot(epoch_range, history.history['accuracy'])
plt.plot(epoch_range, history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```

Plot training & validation loss values

```
plt.plot(epoch_range, history.history['loss'])
plt.plot(epoch_range, history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
plot_learningCurve(history, epochs)
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
dataset_train = pd.read_csv('D:/Pre-processed_Datasets_codes-20230321T052125Z-001/Pre-processed_Datasets_codes/labeled_finaldata.csv')
dataset_train.head()

dataset_train.shape
training_set = dataset_train.iloc[:,1:2].values

from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)

X_train = []
y_train = []
for i in range(60,1258):
X_train.append(training_set_scaled[i-60:i, 0])
y_train.append(training_set_scaled[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)

X_train.shape
y_train.shape
y_train.shape
```

Reshape to 3 dimensions –

No of stock prices, No of timestamps, No of indicatores

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_train.shape
```

Building the LSTM

```
from keras.models import Sequential
from keras.layers import Dense
```

```
from keras.layers import LSTM
from keras.layers import Dropout
```

Initializing the LSTN

```
regressor = Sequential()
regressor.add(LSTM(50, return_sequences=True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))
regressor.add(LSTM(50, return_sequences=True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(50, return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(50, return_sequences=False))
regressor.add(Dropout(0.2))
regressor.add(Dense(1))
regressor.summary()
from tensorflow.keras.optimizers import Adam
```

Compiling the LSTM

```
regressor.compile(optimizer=Adam(lr=0.0001), loss = 'binary_crossentropy', metrics=['accuracy'])
```

Fitting the LSTM to the training set

```
regressor.fit(X_train, y_train, epochs = 10, batch_size = 18)
```

Making the predictions and visualizing the results

```
dataset_test = pd.read_csv('D:/Pre-processed_Datasets_codes-20230321T052125Z-001/Pre-processed_Datasets_codes/labeled_finaldata.csv')
dataset_test.head()
dataset_test.shape
real_stock_price = dataset_test.iloc[:,1:2].values
dataset_total = pd.concat((dataset_train['InsiderThreat'], dataset_test['InsiderThreat']), axis = 0)
dataset_total.shape
```

```
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs.shape
inputs = inputs.reshape(-1,1)
inputs.shape
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 80):
X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Visualizing the results

```
plt.plot(real_stock_price, color="red", label = "activity")
plt.plot(predicted_stock_price, color="blue", label = "Predicted insiderthreat")
plt.title("Predicted insiderthreat")
plt.xlabel("activity")
plt.ylabel("insiderthreat")
plt.legend()
```