BEHAVIORAL RISK CLASSIFIER: MACHINE LEARNING ALGORITHMS TO CLASSIFY USERS BASED ON ONLINE BEHAVIOUR FOR IDENTIFTING POTENTIAL RISKS

A report submitted in partial fulfillment of the requirements for the Degree of

Bachelor of Technology

In

DATA SCIENCE

BY

K. HARSHINI	(2111CS030037)
G. KRANTHI VARMA	(2111CS030048)
G. NIKESH	(2111CS030065)
M. PRASUNA	(2111CS030079)

Under the esteemed guidance of

Dr.N.VIMALA

Associate Professor

Department of Data Science

School of Engineering

MALLA REDDY UNIVERSITY

Maisammaguda, Dulapally, Hyderabad, Telangana 500100

2025

Major Project Report on

BEHAVIORAL RISK CLASSIFIER: MACHINE LEARNING ALGORITHMS TO CLASSIFY USERS BASED ON ONLINE BEHAVIOUR FOR IDENTIFTING POTENTIAL RISKS

A report submitted in partial fulfillment of the requirements for the Degree of

Bachelor of Technology
in

DATA SCIENCE

by

K. HARSHINI	**(2111CS030037)**
G. KRANTHI VARMA	**(2111CS030048)**
G. NIKESH	**(2111CS030065)**
M. PRASUNA	**(2111CS030079)**

Under the esteemed guidance of

DR. N.VIMALA

Associate Professor

Department of Data Science

School of Engineering

MALLA REDDY UNIVERSITY

Maisammaguda, Dulapally, Hyderabad, Telangana 500100

2025

MALLA REDDY UNIVERSITY

(Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)

Department of Data Science

CERTIFICATE

This is to certify that the project report entitled **"BEHAVIOR RISK CLASSIFIER- MACHINE LEARNING ALGORITHMS TO CLASSIFY USERS BASED ON ONLINE BEHAVIOR FOR IDENTIFYING POTENTIAL RISKS",** submitted by **K.HARSHINI (2111CS030037), G.KRANTHIVARMA (2111CS030048), G.NIKESH (2111CS030065), M. PRASUNA (2111CS030079)** towards the partial fulfillment for the award of Bachelor's Degree in COMPUTER SCIENCE ENGINEERING from the Department of DATA SCIENCE, Malla Reddy University, Hyderabad, is a record of bonafide work done by them. The results embodied in the work are not submitted to any other University or Institute for award of any degree or diploma.

Internal Guide:	**Dean / Head of the Department:**
Name: Dr.N.Vimala	**Dr. G.S. Naveen Kumar**
Designation: Associate Professor	

External Examiner

DECLARATION

We hereby declare that the project report entitled "**Behavioral Risk Classifier-Machine Learning Algorithms to Classify Users Based on Online Behavior for Identifying Potential Risks**" has been carried out by us and this work has been submitted to the Department of Data Science, Malla Reddy University, Hyderabad in partial fulfillment of the requirements for the award of degree of Bachelor of Technology. We further declare that this project work has not been submitted in full or part for the award of any other degree in any other educational institutions.

Place: Hyderabad

Date:

K. Harshini	2111CS030037
G. Kranthi Varma	2111CS030048
G. Nikesh	2111CS030065
M. Prasuna	2111CS030079

ACKNOWLEDGEMENT

We extend our sincere gratitude to all those who have contributed to the completion of this project report. Firstly, We would like to extend our gratitude to Dr. V. S. K Reddy, Vice-Chancellor, for his visionary leadership and unwavering commitment to academic excellence.

We would also like to express my deepest appreciation to our project guide **Dr. N.Vimala, Associate Professor** whose invaluable guidance, insightful feedback, and unwavering support have been instrumental throughout the course of this project for successful outcomes.

We are also grateful to **Dr. G. S. Naveen Kumar**, Dean / Head of the Department of **Data Science**, for providing us with the necessary resources and facilities to carry out this project.

We extend our sincere gratitude to **Dr. N. Vimala,** our Project Coordinator, for their valuable suggestions, continuous support, and motivation throughout the course of this project.

We are deeply indebted to all of them for their support, encouragement, and guidance, without which this project would not have been possible.

K. Harshini (2111CS030037)

G. Kranthi Varma (2111CS030048)

G. Nikesh (2111CS030065)

M. Prasuna(2111CS030079)

ABSTRACT

With the exponential growth of online activities, classifying users based on behavioral risks has become crucial for cybersecurity, fraud prevention, and personalized marketing. According to recent statistics, 82% of data breaches involved some form of human error or online behavior misuse, and cybercrimes targeting users rose by 50% in the past year alone. Despite the significance, current manual risk detection systems often struggle with real-time analysis and predictive accuracy. In this work, we propose a Behavioral Risk Classifier that uses Gradient Boosting Algorithm to classify the users into categories of "Safe" or "Risky" based on their online behavior. Our system employs preprocessing techniques such as data cleaning, splitting, normalization, and feature extraction to enhance the quality of input data. A machine learning (ML) model, specifically trained to detect patterns in user interactions, is used to predict potential risks. The model is capable of identifying subtle behavior deviations, allowing proactive risk management and improving user safety.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background and Introduction

With the rapid expansion of digital transactions, e-commerce, social media, and online banking, user behavior analysis has gained significance for security and risk management. In India, the rise of digital adoption has led to an increase in cyber fraud, financial scams, and identity theft. According to reports, India witnessed a 15% rise in cybercrime cases in 2023, with over 13.9 lakh (1.39 million) cases registered. The Reserve Bank of India (RBI) reported that financial frauds through online transactions accounted for losses exceeding ₹1,300 crores in a year. Human errors, phishing attempts, and malicious activities have been primary causes of these incidents. Traditional risk detection methods rely on manual monitoring and rule-based systems, which struggle with scalability and real-time detection. By leveraging machine learning, we can develop a Behavioral Risk Classifier to automatically categorize users as Safe or Risky based on their online behavior patterns. This approach enhances cybersecurity, prevents fraudulent activities, and enables personalized user engagement across various domains such as banking, e-commerce, and social networking.

1.2 Problem Definition

Before the adoption of machine learning in behavioral risk classification, traditional methods heavily relied on rule-based systems and manual reviews to detect potential threats. These methods often suffered from delayed responses, limited adaptability, and high false positives or negatives. Cybersecurity analysts had to manually assess suspicious transactions or user behavior, which was both time- consuming and prone to human error. Additionally, predefined rule-based fraud detection systems lacked the flexibility to identify new and evolving threats, making them ineffective against dynamic cyber-attacks. For instance, fraudulent users could easily bypass conventional detection mechanisms by slightly modifying their behavior patterns. Moreover, organizations struggled with scaling up risk detection processes, leading to severe financial losses and data breaches. The absence of real-time risk classification left companies vulnerable to attacks, requiring a shift towards automated, intelligent, and adaptive risk assessment models.

1.3 Research Motivation

The increasing threat landscape in digital interactions and the limitations of traditional risk assessment methods drive the need for this research. India has experienced an alarming rise in cyber fraud, with 70% of companies reporting security breaches in the past year alone. Despite the rapid growth of cybersecurity technologies, attackers continually evolve their tactics, making it difficult for conventional systems to keep up. The challenge lies in accurately predicting user risks in real time, reducing false alarms, and ensuring seamless user experiences without unnecessary restrictions. Existing risk management models are either too rigid or too lenient, leading to loss of sensitive data or user frustration due to incorrect flagging. The integration of machine learning-based behavioral risk classification can provide a solution that dynamically adapts to new threats while maintaining high accuracy. The potential impact of such a system on fraud prevention, cybersecurity, and personalized user interactions makes it a compelling research area.

1.4 Need

With the surge in digital adoption, individuals and businesses increasingly depend on secure online transactions, data sharing, and authentication mechanisms. However, cyber attackers leverage AI-driven techniques to exploit vulnerabilities in user behavior, leading to financial frauds, identity theft, and phishing attacks. Traditional rule-based detection methods fail to capture complex behavioral deviations that indicate potential threats. A machine learning-based Behavioral Risk Classifier addresses this gap by analyzing vast amounts of user activity data, identifying hidden patterns, and classifying users into risk levels. This enables organizations to implement real-time risk mitigation strategies, such as multi-factor authentication, transaction blocking, or targeted security alerts. Furthermore, regulatory authorities like RBI, SEBI, and CERT-In encourage AI- driven cybersecurity solutions to combat financial fraud. By implementing automated risk classification, businesses can enhance trust, improve user safety, and reduce economic losses caused by fraudulent activities.

1.5 Applications

- ***Cybersecurity*** – Detecting and preventing cyber threats based on user behavior in banking, e-commerce, and online services.
- ***Fraud Detection*** – Identifying and mitigating financial frauds, phishing attacks, and unauthorized transactions.
- ***Social Media Monitoring*** – Identifying fake profiles, misinformation spreaders, and cyberbullying.
- ***Insider Threat Detection*** – Monitoring employee activities to detect policy violations and potential insider risks
- ***Personalized Marketing*** – Enhancing user experience through personalized recommendations based on behavior patterns.
- ***Government and Law Enforcement*** – Assisting in criminal investigations and national security surveillance.
- ***E-commerce Risk Management*** – Detecting fraudulent buyers or sellers and preventing chargebacks.
- ***Online Education Platforms*** – Identifying plagiarism, cheating, and suspicious student activities.

CHAPTER 2

LITERATURE SURVEY

Traditional means of information dissemination, like newspapers, magazines, books, television, and radio, are being replaced by readily available and accessible digital sources of information, including web channels, online social networks, podcasts, etc. .

The insight into how the user interacts with the internet to retrieve information, share it on social media, or trust information can help filter information effectively to suit user intent.

Authors in summarized through survey that an approach towards finding users' online intention follows two steps analyzing the user perspective and approach towards online platforms and using technology to generate a general profile of user online preferences, and developing learning models to identify users based on online information preferences. Many studies and models aim to classify and predict user intention in seeking information online .

One of the trending research topics is to capture user intention in acquiring information and provide ease in bringing that information to the user screen. The research stretches from user navigation, individual preferences, likes and dislikes, search results, and relevance of search queries to semantic analysis of websites for customizing search results to better suit the user intent.

In the study the users' physical attributes are used to analyze user interaction in image searching and content to design a search intent system. Another study analyzes clickstream data in predicting the intention of shopping online using deep learning models. Research on studying and modeling user intention in online searching also focuses on user engagement and behavior on social media with millions of active users.

Online platforms have also become a dynamic source of information, as was noted by the researchers during Covid'19. This research and many others have shaped search engine results, social platform information feeds, and online marketing. Even the traditional means of information dissemination are using web-based technologies to compete with social media platforms.

It is also true that the information available via authenticated print media and news platforms is still considered more reliable than the social platforms. Social platforms, in some way, act like digital word of mouth. Recent surveys show that social platforms are the prime source of information gathering and exchange.

The authors investigated differences in gender behavior and intention in information sharing. According to the authors, the information shared on social platforms varies from status updates and liking a post to advertisement sharing, specially on social media like Facebook or Twitter. Furthermore, the intention to share information differs for men and women based on social ties and commitment.

Furthermore, the intention to share information differs for men and women based on social ties and commitments. The study discusses factors like extraordinary circumstances or times of crisis (e.g., the Covid19 pandemic), social influence, or user attitudes as causes of an increase in the use of social platforms to access and share information being the only way to connect to rest of the world. Another study investigated Taiwan's Instagram users' social behavior and used big data analytics and k-mean to cluster users and generate user profiles for social media and commerce development.

Information supply and circulation have increased drastically with easy access to social platforms like Facebook, Twitter, YouTube, etc. But unfortunately, not every piece of information is reliable. Information verification has become a challenge for news agencies and end users with the spread of fake information. Research shows that users easily fall for false information if it supports their viewpoint.

A recent study showed that certain types of social media users are highly motivated to verify information versus the users whose primary intent on social media is to seek entertainment. Authors in conducted a study exploring the user's purpose in verifying the information and the methods employed.

CHAPTER 3
TRADITIONAL SYSTEM

Before the adoption of machine learning and AI-driven techniques, behavioral risk classification relied heavily on rule-based systems, manual monitoring, and statistical analysis. Traditional approaches used predefined rules set by experts to detect anomalies in user behavior. These systems would flag activities based on factors such as IP address changes, multiple failed login attempts, or unusual transaction amounts. For instance, in online banking, a user making multiple large transactions in a short time might trigger a manual review by a fraud detection team. Similarly, cybersecurity teams used signature-based detection, where threats were identified based on known attack patterns stored in databases. Financial institutions and online service providers depended on static thresholds, meaning any activity crossing a predefined limit was considered suspicious. Analysts manually investigated flagged cases, making decisions based on historical data and personal judgment.

While this approach worked in controlled environments, it struggled to keep up with evolving cyber threats. Attackers quickly learned how to bypass fixed rules by slightly modifying their behavior, making traditional systems ineffective against dynamic fraud patterns. Additionally, these methods required constant manual intervention to update rules and analyze flagged cases, leading to delays in risk detection. As online activities increased exponentially, traditional risk classification systems became overwhelmed with false positives, leading to unnecessary account blocks and poor user experience. Due to these inefficiencies, organizations started shifting toward automated, adaptive machine learning models capable of real-time analysis and pattern recognition.

Limitations of Traditional Systems

- ***Rule-Based Limitations*** – Fixed rules cannot detect new or evolving threats, making them ineffective against adaptive fraud techniques.
- ***High False Positives*** – Many legitimate users get wrongly flagged as risky, leading to frustration and revenue loss for businesses.

- ***Slow Response Time*** – Manual intervention slows down fraud detection and risk mitigation, causing delayed actions.
- ***Scalability Issues*** – As user activity grows, traditional methods fail to handle large datasets efficiently.
- ***Human Error*** – Dependence on manual reviews increases the risk of misclassification due to subjective judgment.
- ***Lack of Real-Time Adaptability*** – Cannot detect sophisticated attacks in real time, allowing threats to go unnoticed.
- ***Resource Intensive*** – Requires large teams of analysts, making it expensive and inefficient.
- ***Poor Personalization*** – Unable to customize risk assessments based on individual user behavior patterns.

CHAPTER 4
PROPOSED SYSTEM

4.1 Uploading the Dataset

Step 1: Upload Online Behavioral Risk Dataset

The process begins with a graphical user interface (GUI) implemented using Tkinter, where the user is prompted to select and upload an online behavioral risk dataset (typically in CSV format). Upon selecting the file through a file dialog, the dataset is read into a Pandas DataFrame, and its basic information such as the file name, dimensions, and a preview of the data is displayed in the application's text widget. This initial step ensures that the dataset is correctly loaded and provides a quick visual confirmation of the data structure for further processing.

Step 2: Data Preprocessing

After uploading the dataset, the next step involves comprehensive data preprocessing to ensure data quality and consistency. The code converts the 'Timestamp' column to a datetime format and extracts useful temporal features like day, month, year, hour, minute, and second, subsequently dropping the original timestamp. In addition, categorical features such as 'Device_Type' and 'Social_Media_Usage' are transformed into numeric values using Label Encoding. Although the sample code does not explicitly handle null values and duplicate removal, these steps are typically integrated into this phase along with the application of a Standard Scaler to normalize the feature set for improved model performance.

Step 3: Label Encoding, PCA, SMOTE, and Resampling

stage refines the data further by balancing the dataset and reducing its dimensionality. First, label encoding is applied to all necessary categorical variables to convert them into numerical format. Next, Synthetic Minority Over-sampling Technique (SMOTE) is used to address class imbalance by generating synthetic samples for the underrepresented class, followed by an additional resampling to standardize the number of samples. The dataset is then normalized using Standard Scaler, and Principal Component Analysis (PCA) is employed to reduce the feature space to a manageable number of components.

Step 4: Deep Neural Network(DNN) Classifier

In this step, machine learning model—the Deep Neural Network Classifier—is employed. The DNN model is trained on the processed training data to learn patterns associated with different risk categories.A sequential neural network is constructed using TensorflowKeras API, incorporating dense layers with ReLU activation, batch normalization, and dropout layers to prevent overfitting. After training, the model makes predictions on the test set, and the performance is evaluated using various metrics. The system calculates accuracy, precision, recall, and F1-score, and generates a confusion matrix and classification report. These evaluations helps to determine how well the traditional machine learning approach is performing in classifying users based on their online behavior.

Step 5: Gradient Boosting Classifier(GBC):

The proposed method introduces a Gradient Boosting classifier to enhance performance and capture more complex patterns in the data. Gradient Boosting is an ensemble learning technique that builds multiple weak learners, typically decision trees, in a sequential manner to improve predictive accuracy. It starts with an initial model that makes predictions, and the errors from these predictions are calculated using a loss function. A new tree is then trained to correct these residuals, learning from the mistakes of the previous model. The final prediction is obtained by adding the weighted outputs of all trees, controlled by a learning rate to prevent overfitting. The process continues for a set number of iterations, gradually minimizing the error. Gradient Boosting optimizes a loss function, such as log loss for classification or mean squared error for regression, making it highly effective for complex datasets. This iterative learning process helps in reducing bias and variance, improves the overall performance. Gradient Boosting Classifier is applied to PCA-transformed and SMOTE-balanced data, ensuring better feature representation and class balance. PCA reduces dimensionality, improving computational efficiency, while SMOTE addresses class imbalances for fairer training. The trained model is evaluated using accuracy, precision, recall, and F1-score, providing insights into its performance. A confusion matrix further helps visualize classification effectiveness. By analyzing online behavioral patterns, the model classifies users into "Safe" or "Risky" cybersecurity behavior categories. Gradient Boosting adaptability and robustness made it a powerful tool for predictive modeling in cybersecurity risk assessment.

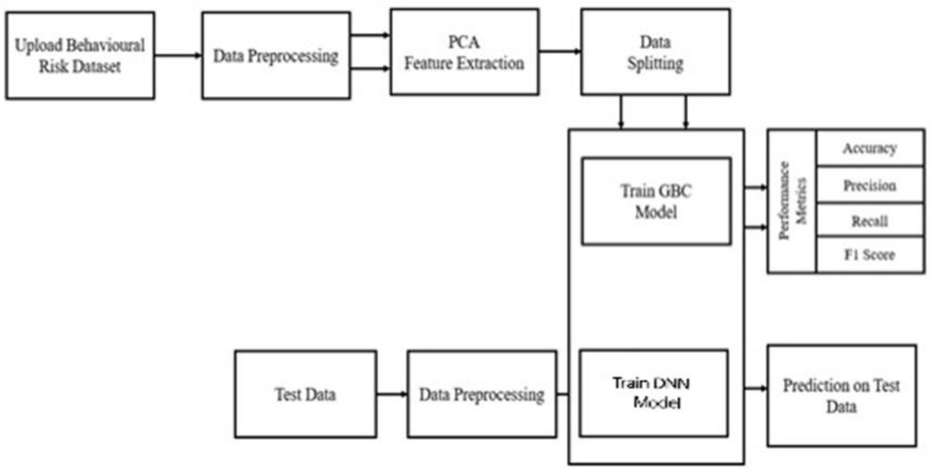
Step 6: Performance Confusion Metrics and Graph

After both models have been trained, their performance is compared using key metrics such as accuracy, precision, recall, and F1-score. The system stores these metrics for both the Gradient Boosting Classifier and the DNN-based classifier, and then visualizes the comparison through a bar graph. This graph provides an intuitive overview of the strengths and weaknesses of each model, highlighting which approach performs better across the different evaluation parameters. Such comparative visualization helps in understanding the improvements brought by the proposed Gradient Boosting classifier.

Step 7: Prediction of Output from Test Data Using the Gradient Boosting Classifier-Trained Model

In the final step, the trained Gradient Boosting Classifier model is applied to a new, unseen test dataset to predict user risk categories. The test data undergoes the same preprocessing, scaling, and PCA transformation as the training data, ensuring consistency in feature representation.

The pre-processed test data is then fed into the previously saved Gradient Boosting Classifier feature extractor to obtain the features, which are subsequently classified using the saved classifier model. The predicted risk labels are mapped back to their corresponding classes (such as ‘Safe’ or ‘Risky’) and appended to the original test dataset. The predictions are then displayed in the application’s text widget, offering a clear outcome of the risk classification process.



4.1: Block Diagram of the Proposed System.

4.2 Data Preprocessing and Data Splitting

Data Preprocessing

Data preprocessing is a crucial step in machine learning to clean and prepare raw

data for training models. In this project, the following preprocessing steps are applied:

1. ***Handling Timestamps:***
   - The dataset contains a Timestamp column, which is converted into separate date and time features such as day, month, year, hour, minute, and second.
   - The original Timestamp column is dropped to avoid redundancy.
2. ***Encoding Categorical Data:***
   - Since machine learning models work with numerical data, categorical features such as Device_Type, Social_Media_Usage, Password_Strength, and other behavioral attributes are encoded using Label Encoding.
3. ***Handling Missing and Duplicate Values:***

- The dataset is checked for missing (NaN) values, and appropriate handling methods (such as removal or imputation) are applied.
- Duplicate rows are identified and removed to avoid data redundancy.

4. ***Feature Scaling***:

- To ensure that all numerical features have a similar scale, StandardScaler is applied, transforming the data to a standardized normal distribution with mean = 0 and variance =1.

Data Splitting

Once preprocessing is complete, the dataset is split into training and testing sets:

1. ***Balancing the Data***:
    - Since real-world data is often imbalanced, Synthetic Minority Over- sampling Technique (SMOTE) is used to generate synthetic samples for the minority class, ensuring that the model does not become biased toward the majority class.Random Resampling further adjusts the dataset to ensure a well-distributed sample set.
2. ***Dimensionality Reduction using PCA***:
    - **Principal Component Analysis (PCA)** is applied to reduce the number of features while retaining essential information. This helps in improving computational efficiency and model performance.
3. ***Train-Test Split***:
    - The processed dataset is divided into 80% training and 20% testing using train_test_split().
    - The training set is used to train models, while the test set evaluates performance on unseen.

4.3 ML Model Building

Deep Neural Network (DNN) Classifier

What is DNN Classifier?

A Deep Neural Network (DNN) classifier is an artificial neural network with multiple hidden layers that can learn complex patterns in data. It is an advanced machine learning model capable of handling large-scale and high-dimensional data.

How DNN Works?

1. ***Input Layer***
   - Takes numerical features from the dataset and passes them to the next layer.
2. ***Hidden Layers***
   - Multiple layers of neurons apply transformations using weights, biases, and activation functions (e.g., ReLU).
   - Each neuron processes inputs from the previous layer and passes the output to the next layer.
3. ***Backpropagation and Optimization***
   - The model is trained using backpropagation, where errors from the output are propagated back to adjust weights.
   - Optimizers such as Adam or SGD (Stochastic Gradient Descent) help update the weights.
4. ***Output Layer***
   - The final layer applies an activation function (e.g., softmax for classification) to produce the probability of each class.

Architecture of DNN

- ***Input Layer***: Receives input features.
- ***Multiple Hidden Layers***: Process data using neurons with activation functions.
- ***Activation Functions***: Typically ReLU (Rectified Linear Unit) for hidden layers, and softmax for classification.
- ***Backpropagation***: Computes gradients and updates weights using optimizers.

- ***Output Layer***: Produces final class probabilities

Gradient Boosting Classifier (GBC)

What is GBC Classifier?

Gradient Boosting Classifier (GBC) is an ensemble learning technique that builds a strong classifier by combining multiple weak learners, typically decision trees. It is based on the principle of boosting, where models are trained sequentially to correct the mistakes of the previous models.

How GBC Works?

1. **Initialize the Model**
    - A weak model (typically a decision tree) is trained on the dataset.
    - The model predicts the target variable, and residual errors (differences between actual and predicted values) are calculated.
2. **Sequential Learning**
    - A new decision tree is trained to predict the residual errors from the previous model.
    - The predictions from the new tree are added to improve the previous model's predictions.
    - This process repeats for multiple iterations, where each new model learns to correct the previous errors.
3. **Gradient Descent Optimization**
    - The model minimizes the loss function (e.g., log loss for classification) by adjusting weights using gradient descent.

4. **Final Prediction**

- After several iterations, the final model aggregates all weak learners to make a strong prediction.

Architecture of GBC

- **Base Learner**: Decision trees (shallow, weak models).
- **Loss Function**: Typically log loss (for classification).
- **Gradient Descent**: Optimizes the model by reducing error at each step
- **Ensemble of Models**: Each tree is added sequentially to correct previous errors.

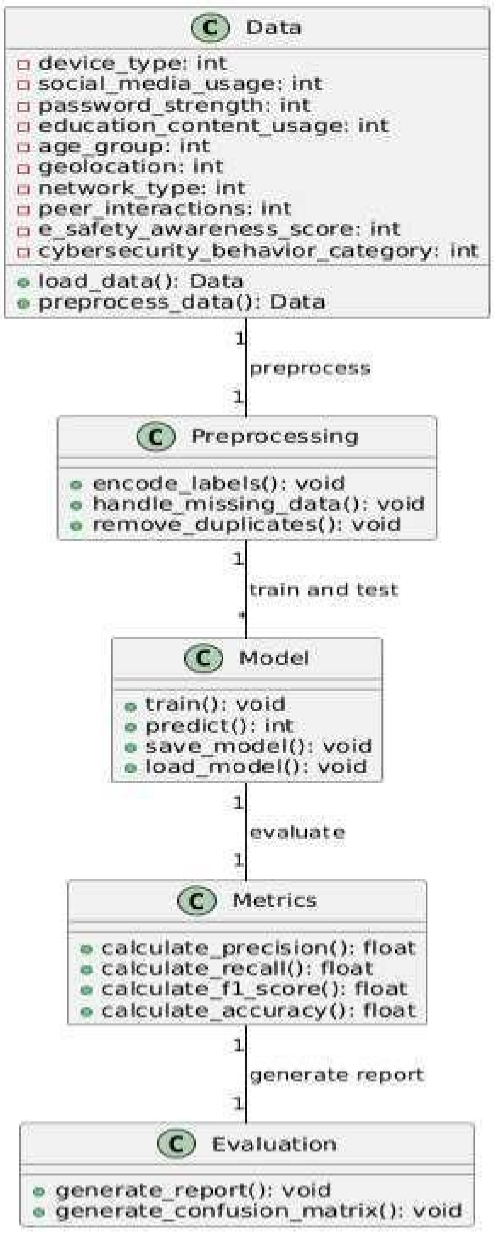
CHAPTER 5
UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general- purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

5.1 Class diagram

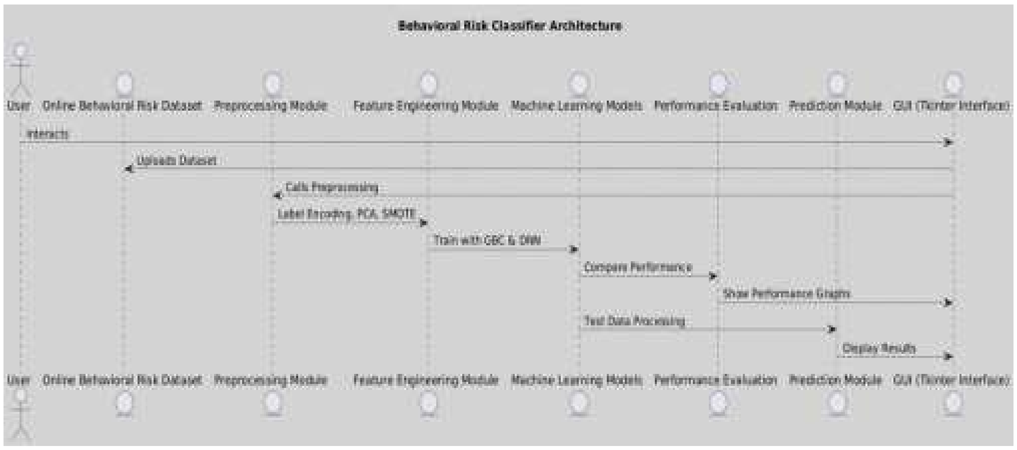
The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram was capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.



5.1 : Class Diagram

5.2 Sequence Diagram

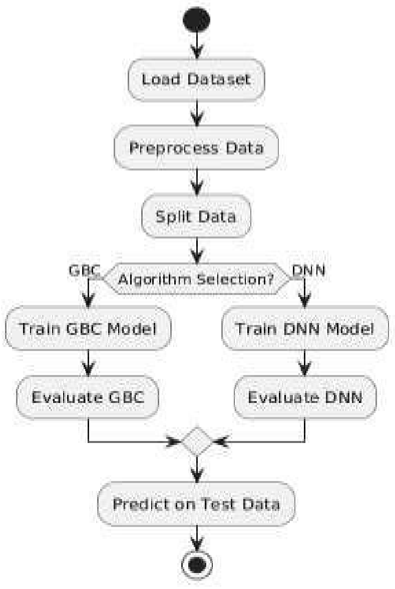
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



5.2 : Sequence Diagram

5.3 Activity diagram

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step- by-step workflows of components in a system. An activity diagram shows the overall flow of control.

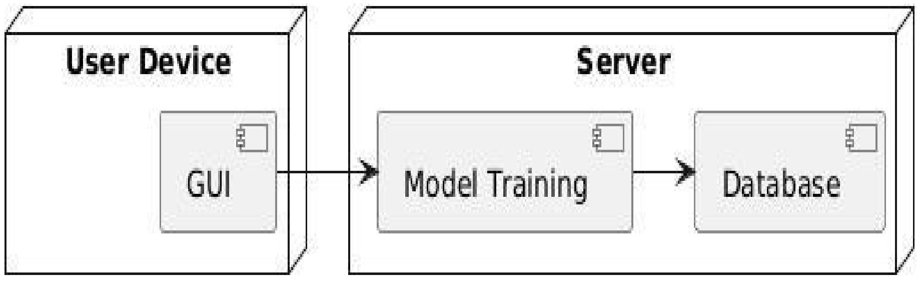


5.3 : Activity diagram

5.4 Deployment Diagram:

A deployment diagram in UML illustrates the physical arrangement of hardware and software components in the system. It visualizes how different software artifacts, such as data processing scripts and model training components, are deployed across hardware

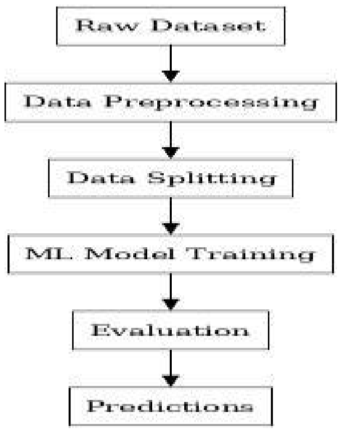
goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



5.4: Deployment Diagram

5.5 Data flow diagram

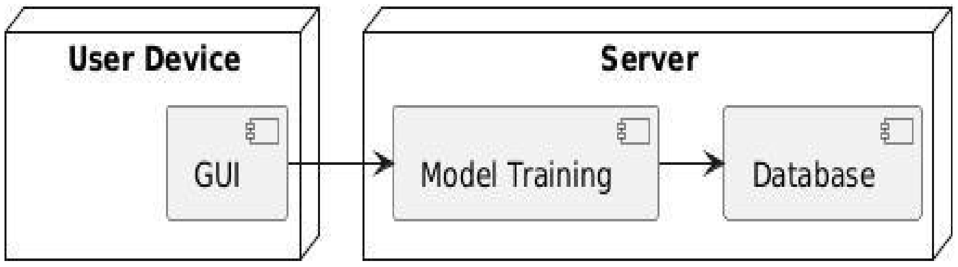
A data flow diagram (DFD) is a graphical representation of how data moves within an information system. It is a modeling technique used in system analysis and design to illustrate the flow of data between various processes, data stores, data sources, and data destinations within a system or between systems. Data flow diagrams are often used to depict the structure and behavior of a system, emphasizing the flow of data and the transformations it undergoes as it moves through the system.



5.5: Data flow diagram

5.6 Use Case diagram:

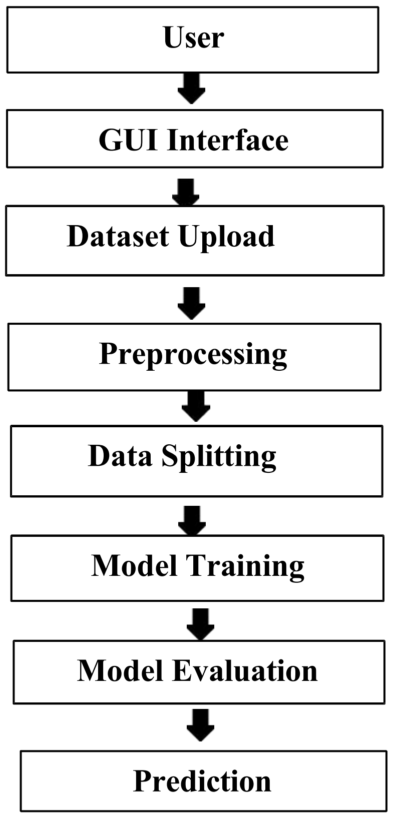
A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



5.6 :Use Case Diagram

5.7 Architectural Block Diagram

An architectural block diagram offers a high-level view of a system's structure, showcasing the main components and their interactions. It represents how major modules, such as data sources, processing units, and evaluation components, are organized and how they communicate with each other to accomplish the system's objectives. This diagram helps in understanding the overall design and flow of the system.



5.7: Architectural Block Diagram

CHAPTER 6
SOFTWARE ENVIRONMENT

6.1 What is Python?

Below are some facts about Python.

- Python is currently the most widely used multi-purpose, high-level programming language.
- Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.
- Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
- Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard libraries which can be used for the following –

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like Opencv, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia

6.2 Advantages of Python

1. *Extensive Libraries*

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

2. *Extensible*

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

3. *Embeddable*

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

4. *Improved Productivity*

The language’s simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

5. *IOT Opportunities*

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet Of Things. This is a way to connect the language with the real world.

6. *Simple and Easy*

When working with Java, you may have to create a class to print ‘Hello World’. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

7. *Readable*

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. This further aids the readability of the code.

8. ***Object-Oriented***

This language supports both the procedural and object-oriented programming paradigms.

9. ***Free and Open-Source***

Like said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

10. ***Portable***

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

11. ***Interpreted***

Lastly, will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

Advantages of Python Over Other Languages

1. ***Less Coding***

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don't have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

2. ***Affordable***

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

3. ***Python is for Everyone***

While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you

can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all- rounder programming language.

6.3 Disadvantages of Python

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

1. ***Speed Limitations***

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

2. ***Weak in Mobile Computing and Browsers***

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

The reason it is not so famous despite the existence of Brython is that it isn't that secure.

3. ***Design Restrictions***

As you know, Python is dynamically-typed. This means that you don't need to declare the type of variable while writing the code. It uses duck-typing. But wait, what's that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

4. ***Underdeveloped Database Access Layers***

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

5. ***Simple***

No, we're not kidding. Python's simplicity can indeed be a problem. Take my example. I don't do Java, I'm more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

6.4 History of Python

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wicklund &Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners1, Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin- end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

6.5 Python Development Steps

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991. This release included already exception handling, functions, and the core data types of list, dict, str and others. It was also object oriented and had a module system.

Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting unicode. Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python

3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only one -- obvious way to do it. "Some changes in Python 7.3: Print is now a function.

- Views and iterators instead of lists
- The rules for ordering comparisons have been simplified. E.g., a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.
- There is only one integer type left, i.e., int. long is int as well.

Purpose

We demonstrated that our approach enables successful segmentation of intra-retinal layers—even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout—with the assistance of the ANIS feature.

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with

which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

6.6 Modules Used in Project

1.*NumPy*

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

- A powerful N-dimensional array object
- Sophisticated (broadcasting)
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

2.*Pandas*

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

3.*Matplotlib*

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

4.***Scikit – learn***

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but

it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

6.7 Python Installation

Install Python Step-by-Step in Windows and Mac

Python a versatile programming language doesn't come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

How to Install Python on Windows and Mac

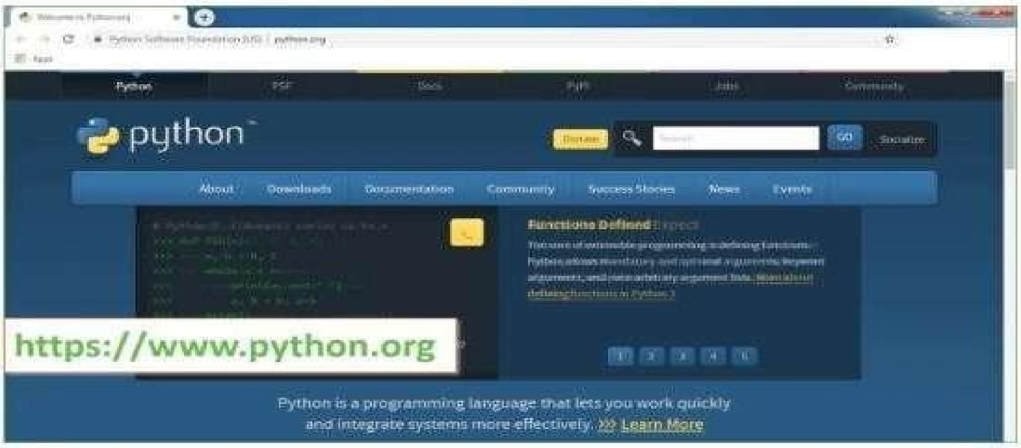
There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices. Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e. operating system and based

processor, you must download the python version. My system type is a Windows 64-bit operating system. So the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here.The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better. Download the Correct version into the system

Step 1: Go to the official site to download and install python using Google Chrome or any other web

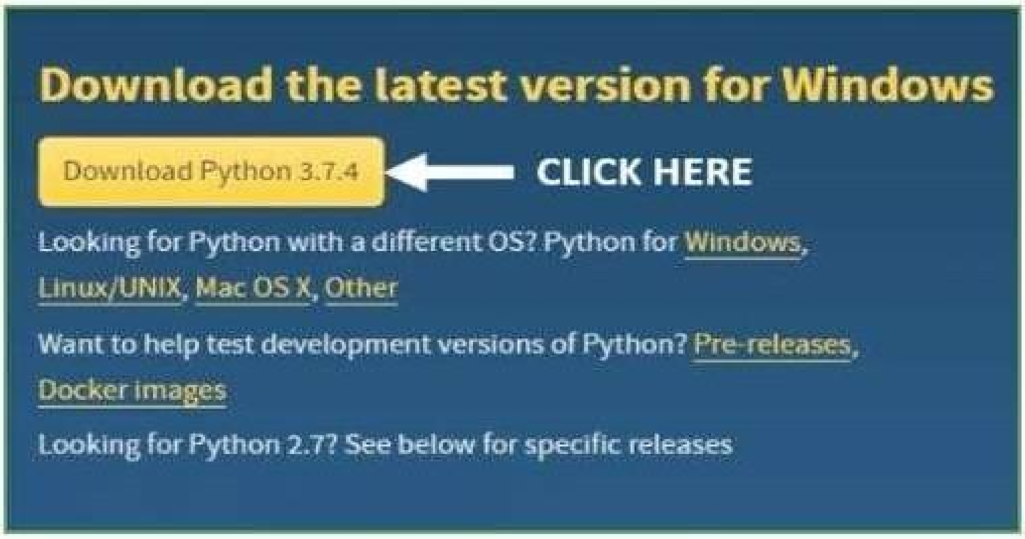
browser. OR Click on the following link: https://www.python.org



Python is a programming language that lets you work quickly
and integrate systems more effectively. >>> Learn More

Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.



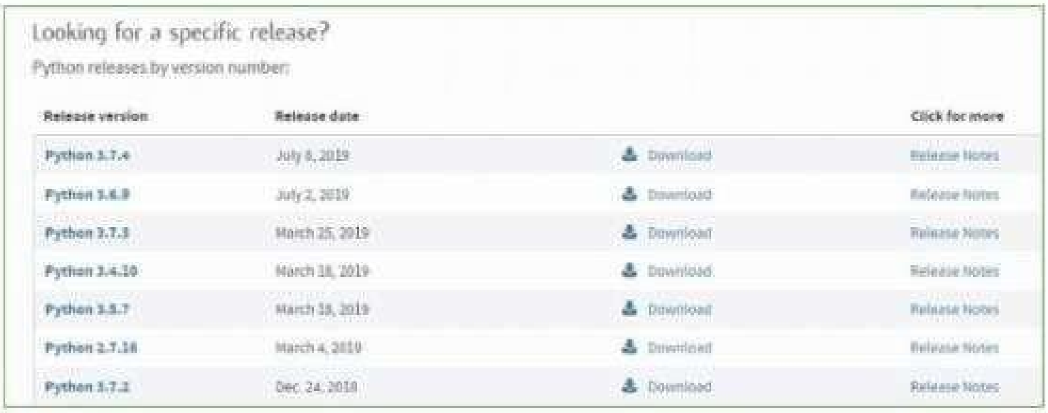
Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases, Docker images

Looking for Python 2.7? See below for specific releases

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4



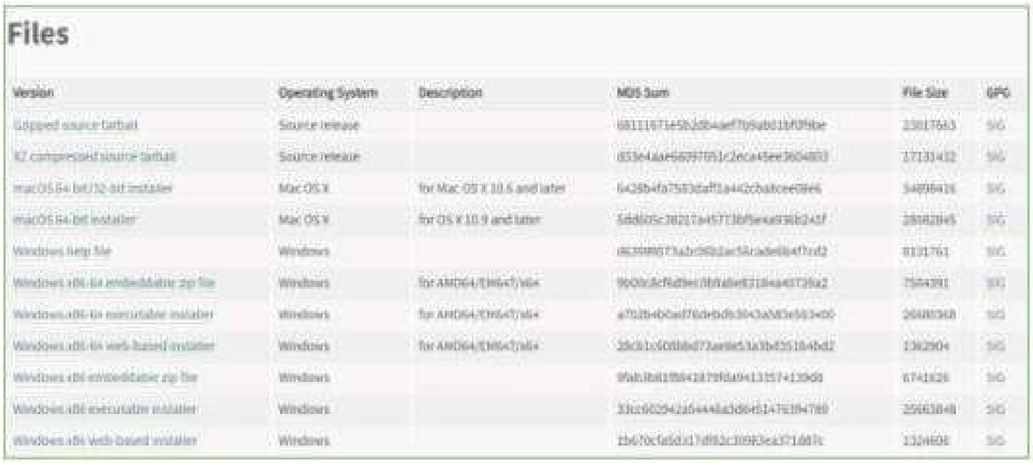
Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.7.4	July 8, 2019	Download	Release Notes
Python 3.6.9	July 2, 2019	Download	Release Notes
Python 3.7.3	March 25, 2019	Download	Release Notes
Python 3.4.10	March 18, 2019	Download	Release Notes
Python 3.5.7	March 18, 2019	Download	Release Notes
Python 2.7.16	March 4, 2019	Download	Release Notes
Python 3.7.2	Dec. 24, 2018	Download	Release Notes

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		[illegible]	23017663	SIG
XZ compressed source tarball	Source release		[illegible]	17131432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	[illegible]	34898416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	[illegible]	28082845	SIG
Windows help file	Windows		[illegible]	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	[illegible]	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	[illegible]	26680368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	[illegible]	1362904	SIG
Windows x86 embeddable zip file	Windows		[illegible]	6741626	SIG
Windows x86 executable installer	Windows		[illegible]	25663848	SIG
Windows x86 web-based installer	Windows		[illegible]	1324608	SIG

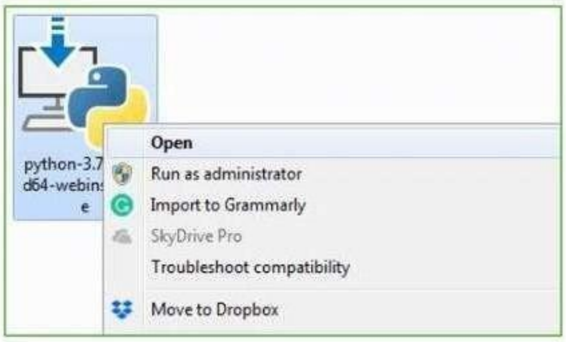
- To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
- To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e. Installation

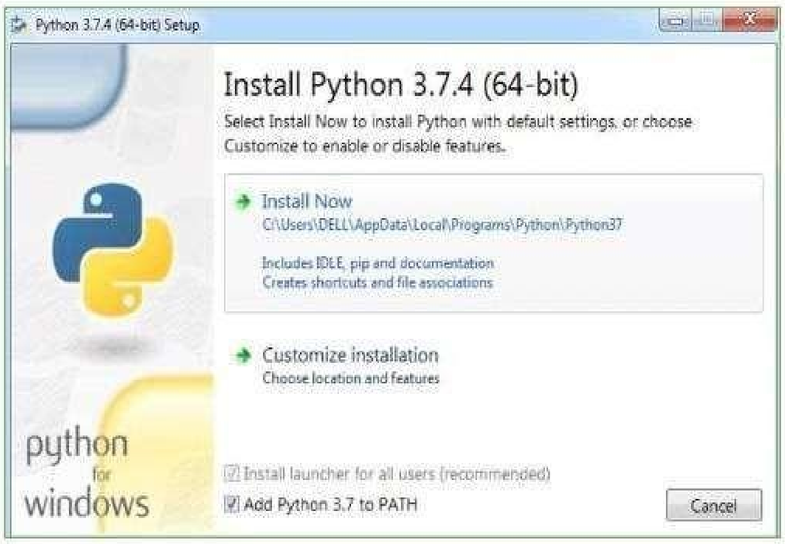
Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.



Step 2: Before you click on Install Now, Make sure to put a tick on Add Python 3.7 to PATH.



Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

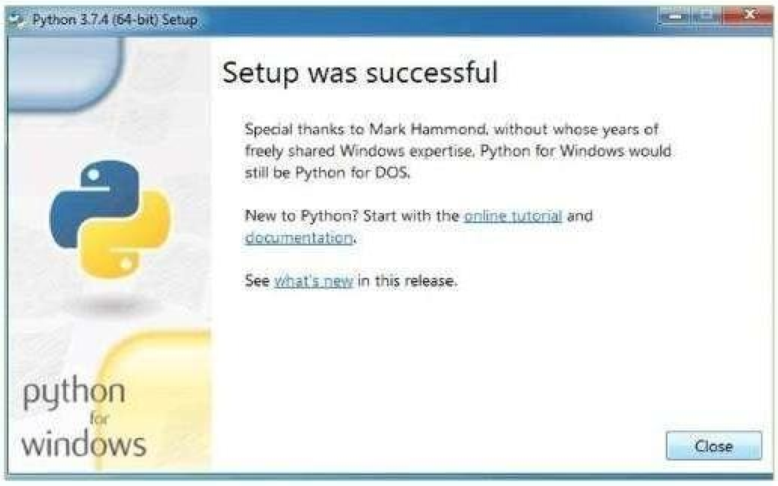
Install Now
C:\Users\DELL\AppData\Local\Programs\Python\Python37

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

Step 3: Click on Install NOW After the installation is successful. Click on Close.



Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

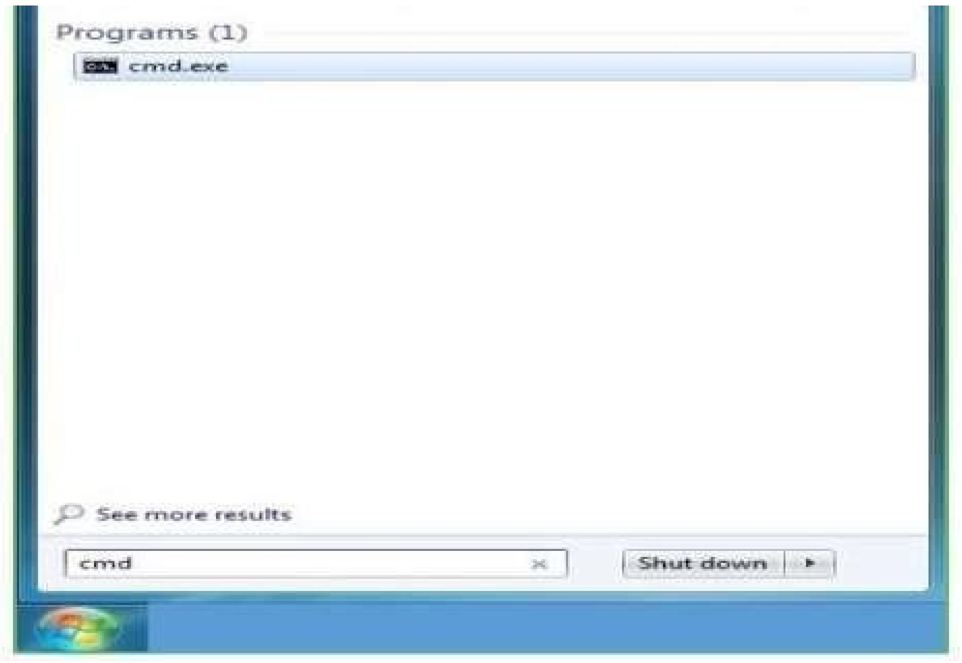
See what's new in this release.

With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.
Note: The installation process might take a couple of minutes.

Verify the Python Installation

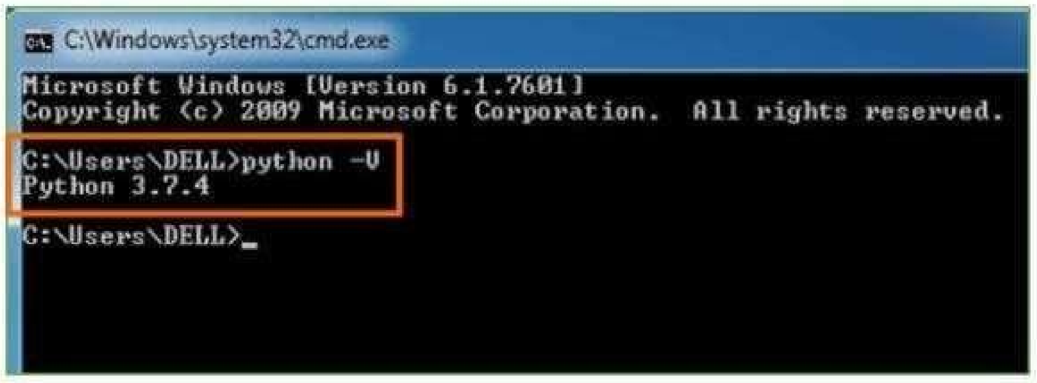
Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.



Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\DELL>python -V
Python 3.7.4

C:\Users\DELL>_
```

Step 5: You will get the answer as 3.7.4

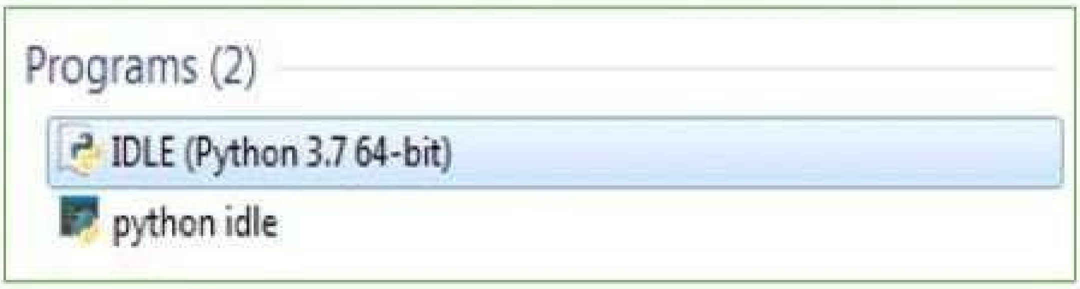
Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python

IDLE works Step 1: Click

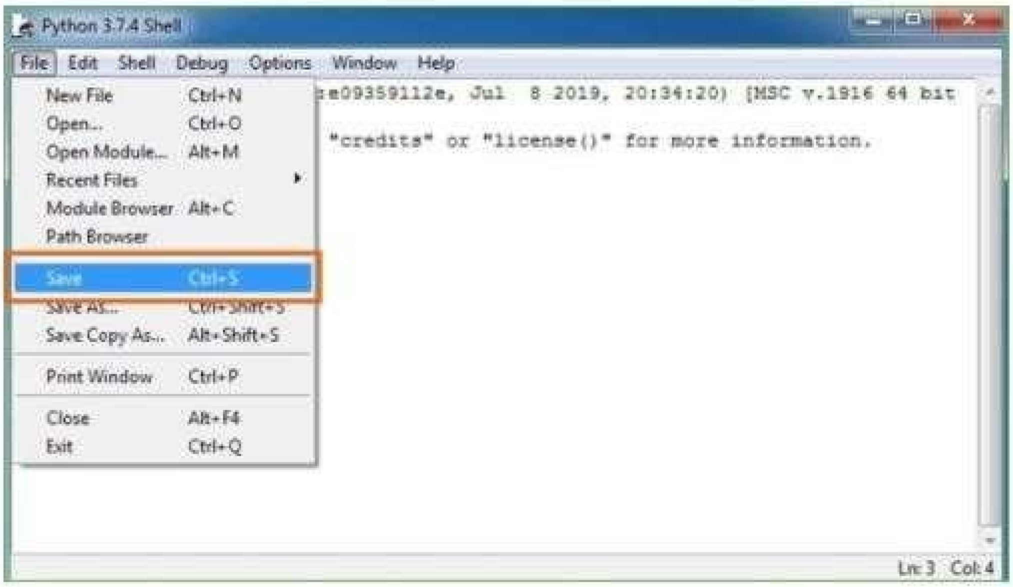
on Start

Step 2: In the Windows Run command, type “python idle”.



Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

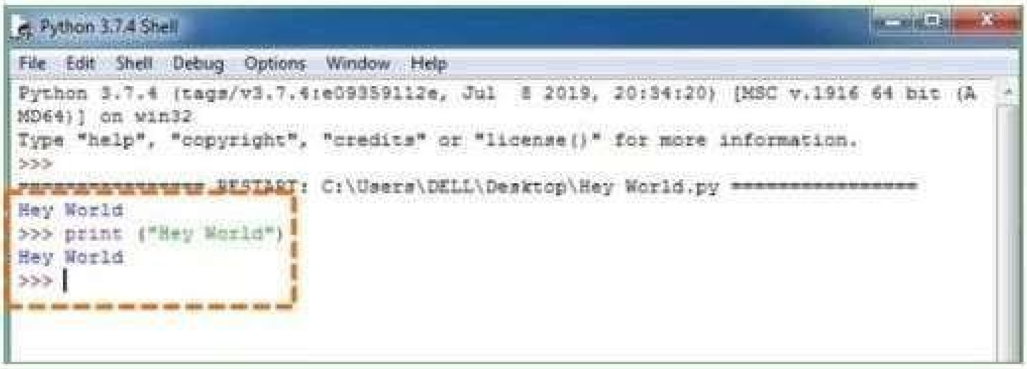
Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save



```
e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit

"credits" or "license()" for more information.
```

Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g. enter print (“Hey World”) and Press Enter



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (A
MD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\DELL\Desktop\Hey World.py ================
Hey World
>>> print ("Hey World")
Hey World
>>>
```

CHAPTER 7

SYSTEM REQUIREMENTS

Software requirements

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

- Python IDLE 3.7 version (or)
- Anaconda 3.7 (or)
- Jupiter (or)
- Google Collab
- Visual Studio

Hardware requirements

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

• Operating system	:	Windows, Linux
• Processor	:	minimum intel i3
• Ram	:	minimum 4 GB
• Hard disk	:	minimum 250GB

CHAPTER 8
FUNCTIONAL REQUIREMENTS

Output Design

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provides a permanent copy of the results for later consultation. The various types of outputs in general are:

- External Outputs, whose destination is outside the organization
- Internal Outputs whose destination is within organization and they are the
- User's main interface with the computer.
- Operational outputs whose use is purely within the computer department.
- Interface outputs, which involve the user in communicating directly.

Output Definition

The outputs should be defined in terms of the following points:

- Type of the output
- Content of the output
- Format of the output
- Location of the output
- Frequency of the output
- Volume of the output
- Sequence of the output

It is not always desirable to print or display data as it is held on a computer. It should be decided as which form of the output is the most suitable.

Input Design

Input design is a part of overall system design. The main objective during the input design is as given below:

- To produce a cost-effective method of input.
- To achieve the highest possible level of accuracy.
- To ensure that the input is acceptable and understood by the user.

Input Stages

The main input stages can be listed as below:

- Data transcription
- Data conversion
- Data verification
- Data Control
- Data transmission
- Data Validation
- Data Collection
- Flexibility of format
- Speed

Input Types

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

- External inputs, which are prime inputs for the system.
- Internal input, which are user communication with the system
- Operational, which are computer department's communication to system.
- Interactive, which are inputs enter during a dialogue.

Input Media

At this stage choice has to be made about the input media. To conclude about the input.

- Type of input.
- Flexibility of format
- Speed

- Accuracy
- Verification methods
- Rejection rates
- Ease of correction

Storage and handling require me Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

Error Avoidance

At this stage care is to be taken to ensure that input data remains accurate form the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

Error Detection

Even though every effort is make to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

Data Validation

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

User interface Design

It is essential to consult the system users and discuss their needs while designing the user interface:

User interface systems can be broadly classified as:

- User initiated interface the user is in charge, controlling the progress of the user/computer dialogue. In the computer-initiated interface, the computer selects the next stage in the interaction.
- Computer initiated interfaces

In the computer-initiated interfaces the computer guides the progress of the user/computer dialogue. Information is displayed and the user response of the computer takes action or displays further information.

User initiated Interfaces

User initiated interfaces fall into two approximate classes:

- Command driven interfaces: In this type of interface the user inputs commands or queries which are interpreted by the computer.
- Forms oriented interface: The user calls up an image of the form to his/her screen and fills in the form. The forms-oriented interface is chosen because it is the best choice.

Computer-initiated Interfaces

The following computer – initiated interfaces were used:

- The menu system for the user is presented with a list of alternatives and the user chooses one; of alternatives.
- Questions – answer type dialog system where the computer asks question and takes action based on the basis of the users reply.

Right from the start the system is going to be menu driven, the opening menu displays the available options. Choosing one option gives another popup menu with more options. In this way every option leads the users to data entry form where the user can key in the data.

Error message Design

The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

This application must be able to produce output at different modules for different inputs.

Performance Requirements

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

- The system should be able to interface with the existing system
- The system should be accurate
- The system should be better than the existing system.

CHAPTER 9
RESULTS AND DISCUSSION

9.1 Implementation Description

The research follows a structured pipeline to preprocess data, train machine learning models, and evaluate performance. The implementation consists of several key steps:

1. Data Collection and Preprocessing

a) ***Data Acquisition***

- The dataset is collected from a reliable source.
- The data is loaded into a suitable format for further processing.

b) ***Data Cleaning***

- Missing values are identified and handled through imputation or removal.
- Duplicate entries are removed to ensure consistency.
- Outliers are detected and addressed if necessary.

c) ***Feature Engineering***

- Relevant features are selected based on domain knowledge.
- New features are created to enhance model performance.
- Categorical variables are encoded using appropriate techniques
- (e.g., one- hot encoding, label encoding).

d) ***Data Normalization and Scaling***

- Numerical features are scaled to ensure uniformity.
- Standardization or Min-Max scaling is applied depending on the model requirements.

2. Data Splitting

- The dataset is divided into training, validation, and test sets.
- The training set is used to train the machine learning model.
- The validation set helps fine-tune hyperparameters.

3. Machine Learning Model Building

3.1 Deep Neural Network (DNN)

- The neural network architecture is defined with input, hidden, and output layers.
- The activation functions are selected for each layer.
- The dataset is fed into the network for training.
- The model undergoes forward propagation, computing weighted sums and activations.
- Backpropagation updates weights using the loss function and an optimizer (e.g., Adam).
- The trained model is validated and evaluated for performance.

3.2 Gradient Boosting Classifier (GBC)

- The model is initialized with default hyperparameters.
- The training data is fed into the GBC model.
- The model sequentially builds weak learners (decision trees), improving performance at each step.
- Gradient descent optimization updates model weights based on residualerrors.
- The trained model is evaluated using accuracy, precision, recall, and
- F1-score.

4. Model Evaluation

- Performance metrics such as accuracy, precision, recall, and F1-score are calculated.
- A confusion matrix is generated to analyze classification results.
- ROC and AUC curves are plotted to assess the model's ability to distinguish between classes.
- The results of the existing and proposed models are compared.

9.2 Dataset Description

The dataset consists of various attributes related to cybersecurity behaviors, device usage, and online activities. Each record represents an individual's interaction with digital platforms, security measures, and risk factors. The dataset is structured into multiple feature categories.

1. Temporal and Device Information

- ***Timestamp***: The recorded date and time of the activity.
- ***Device_Type***: The type of device used, such as a smartphone, tablet, laptop, or desktop.

2. Security Threat Indicators

- ***Malware_Detection***: Indicates the presence of malware on the device.
- ***Phishing_Attempts***: The number of detected phishing attempts.
- ***Firewall_Logs***: Records of network security events filtered by the firewall
- ***Download_Risk***: The risk level associated with downloaded files.
- ***Password_Strength***: A measure of the robustness of user passwords.
- ***Data_Breach_Notifications***: Alerts received regarding potential data breaches.
- ***Insecure_Login_Attempts***: The count of login attempts detected on unsecured platforms.

3. Online Behavior and Usage Patterns

- ***Social_Media_Usage:*** The duration or frequency of social media activity.
- ***VPN_Usage***: Whether a Virtual Private Network (VPN) is used for secure browsing.
- ***Public_Network_Usage***: The frequency of accessing public or unsecured networks.
- ***Network_Type***: The classification of the network being used, such as home, corporate, or public.
- ***Hours_Online***: The total time spent online.

- ***Website_Visits***: The number of websites accessed.
- ***Risky_Website_Visits***: The count of visits to websites categorized as unsafe.
- ***Cloud_Service_Usage***: The extent of reliance on cloud storage and services.
- ***Unencrypted_Traffic***: The volume of data transmitted without encryption.
- ***Ad_Clicks***: The number of advertisements clicked by the user.

4. Cybersecurity Awareness and Parental Controls

- ***Cyberbullying_Reports***: The number of cyberbullying incidents reported.
- ***Parental_Control_Alerts***: Notifications triggered by parental control software.
- ***E_Safety_Awareness_Score***: A score measuring an individual's knowledge and adherence to cybersecurity best practices.

5. User Demographics and Interactions

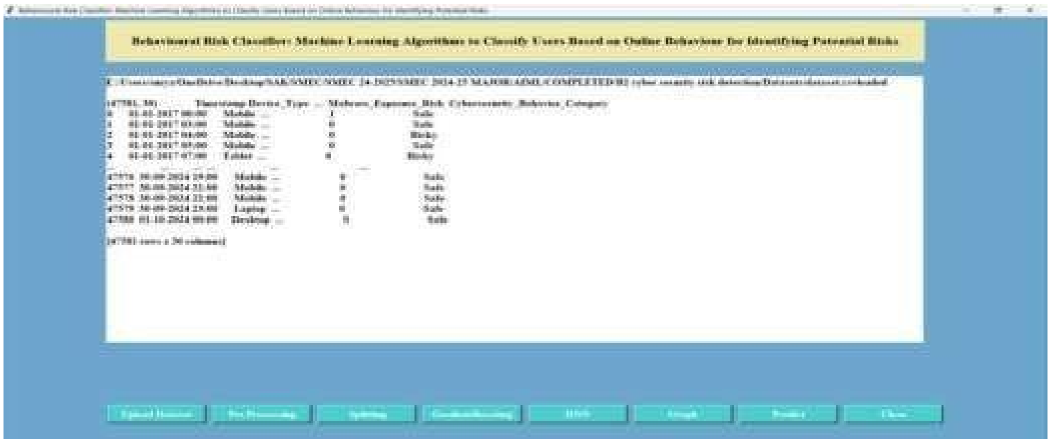
- ***Age_Group***: The age bracket of the user, such as children, teenagers, or adults.
- ***Geolocation***: The geographical location associated with the user's activity.
- ***Peer_Interactions***: The frequency of online social interactions with peers.
- ***Education_Content_Usage***: The level of engagement with educational materials and platforms.

6. Risk Assessment and Cybersecurity Behavior

- ***Online_Purchase_Risk***: The risk level of financial transactions conducted
- ***Malware_Exposure_Risk***: The probability of encountering malware based on usage patterns.
- ***Cybersecurity_Behavior_Category***: A classification label indicating the overall cybersecurity posture, categorized based on safe or risky online behaviors.

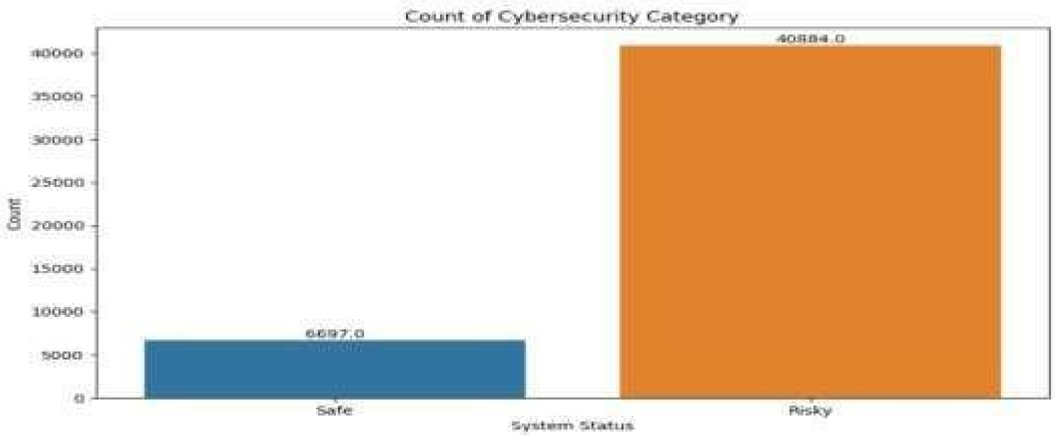
9.3 Results and Description

The below figure displays the graphical user interface (GUI) used for uploading the behavioral risk dataset. The dataset contains various cybersecurity-related attributes, including malware detection, phishing attempts, VPN usage, and online activity patterns. Once the dataset is uploaded, the system processes the data to provide an initial overview, such as summary statistics, missing values, and attribute distributions. The interface ensures a user-friendly experience for data visualization and validation before further processing.

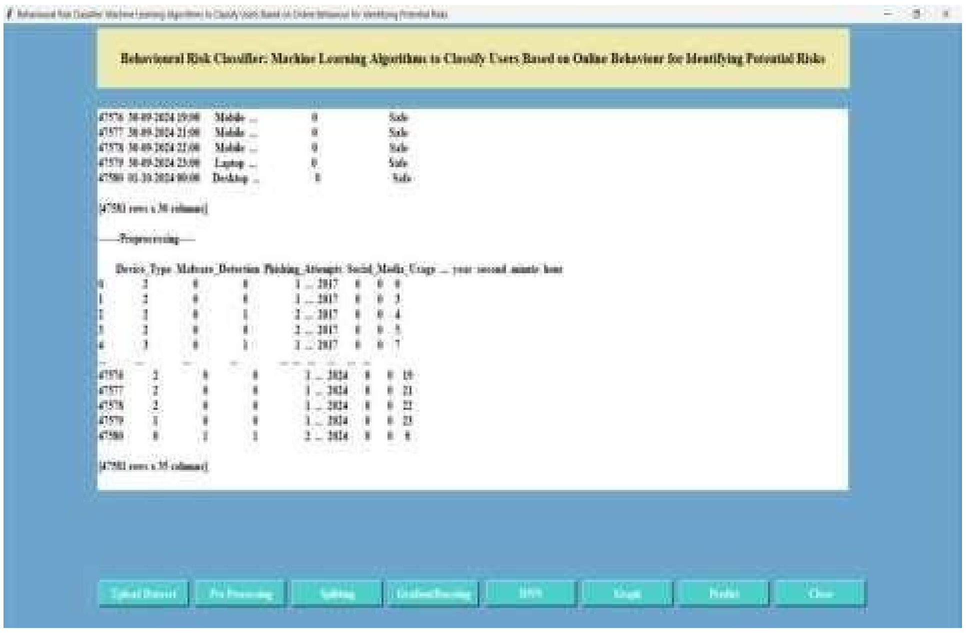

Behavioral Risk Classifier: Machine Learning Algorithms to Classify Users Based on Online Behaviour for Identifying Potential Risks
Upload Dataset
Pre Processing
Splitting
GradientBoosting
DNN
Graph
Predict
Close

```
C:/Users/[illegible]/OneDrive/Desktop/SAK/SMEC/SMEC 24-2025/SMEC 2024-25 MAJOR/AIML/COMPLETED/B] cyber security risk detection/Datasets/dataset.csv loaded

(47581, 30)          Timestamp Device_Type  ...  Malware_Exposure_Risk  Cybersecurity_Behavior_Category
0      01-01-2017 00:00    Mobile  ...                      1                 Safe
1      01-01-2017 03:00    Mobile  ...                      0                 Safe
2      01-01-2017 04:00    Mobile  ...                      0                Risky
3      01-01-2017 05:00    Mobile  ...                      0                 Safe
4      01-01-2017 07:00    Tablet  ...                      0                Risky
...                 ...       ...  ...                    ...                  ...
47576  30-09-2024 19:00    Mobile  ...                      0                 Safe
47577  30-09-2024 21:00    Mobile  ...                      0                 Safe
47578  30-09-2024 22:00    Mobile  ...                      0                 Safe
47579  30-09-2024 23:00    Laptop  ...                      0                 Safe
47580  01-10-2024 00:00   Desktop  ...                      0                 Safe

[47581 rows x 30 columns]
```

9.3.1 : Uploading the Behavioral Risk Dataset and its Analysis in the GUI Interface



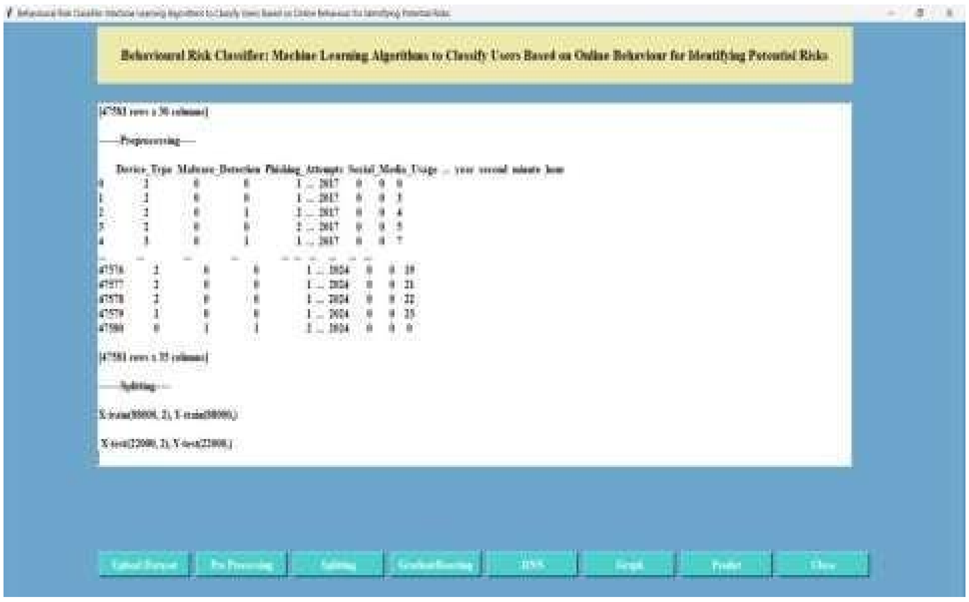
9.3.2 :Count of Cybersecurity Category


Behavioural Risk Classifier: Machine Learning Algorithms to Classify Users Based on Online Behaviour for Identifying Potential Risks
Upload Dataset
Pre Processing
Splitting
GradientBoosting
DNN
Graph
Predict
Close

```
47576  30-09-2024 19:00    Mobile ...        0          Safe
47577  30-09-2024 21:00    Mobile ...        0          Safe
47578  30-09-2024 22:00    Mobile ...        0          Safe
47579  30-09-2024 23:00    Laptop ...        0          Safe
47580  01-10-2024 00:00    Desktop ...       0          Safe

[47581 rows x 30 columns]

-----Preprocessing-----

    Device_Type  Malware_Detection  Phishing_Attempts  Social_Media_Usage  ...  year  second  minute  hour
0         2          0          0          1  ...  2017    0    0    0
1         2          0          0          1  ...  2017    0    0    3
2         2          0          1          2  ...  2017    0    0    4
3         2          0          0          2  ...  2017    0    0    5
4         3          0          1          1  ...  2017    0    0    7
...      ...        ...        ...        ...  ...   ...  ...  ...  ...
47576     2          0          0          1  ...  2024    0    0   19
47577     2          0          0          1  ...  2024    0    0   21
47578     2          0          0          1  ...  2024    0    0   22
47579     1          0          0          1  ...  2024    0    0   23
47580     0          1          1          2  ...  2024    0    0    0

[47581 rows x 35 columns]
```

9.3.3 : Data Preprocessing of the Dataset

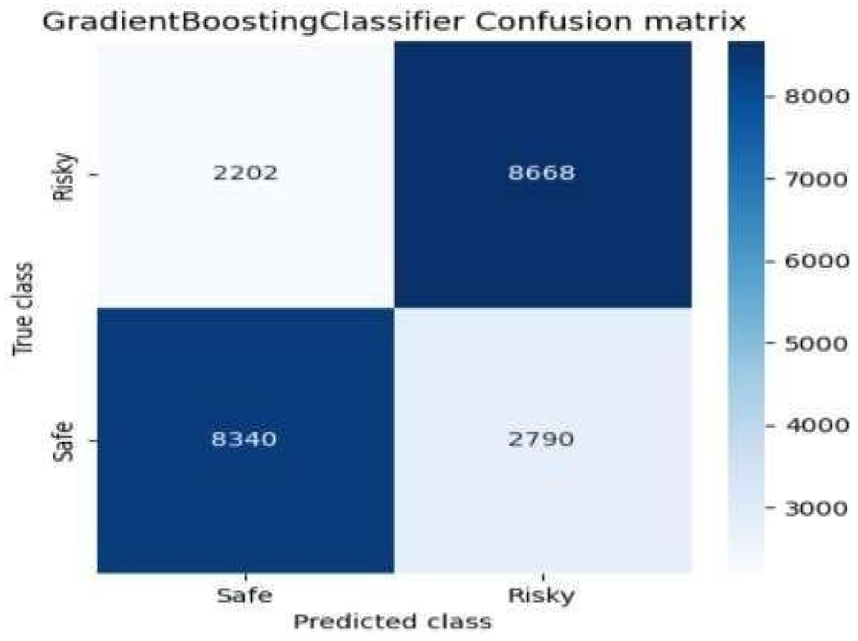
This figure represents the preprocessing steps applied to the dataset to enhance model performance. The preprocessing involves handling missing values, encoding categorical variables, normalizing numerical features, and removing irrelevant attributes. Outliers are detected and addressed to prevent skewed predictions. Feature selection techniques identify the most relevant attributes that contribute to cybersecurity risk classification, ensuring that the dataset is optimized for model training.

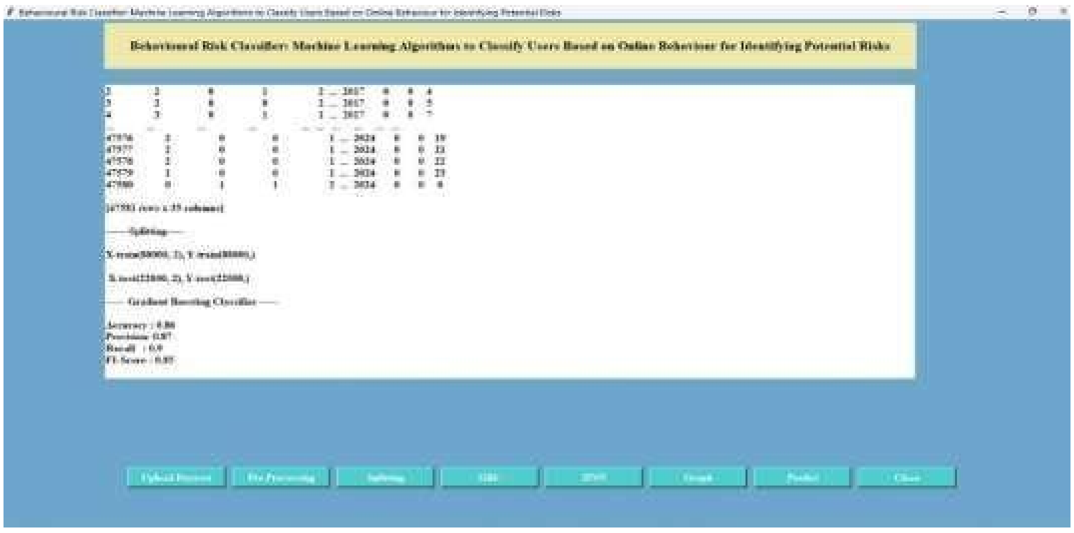

Behavioural Risk Classifier: Machine Learning Algorithms to Classify Users Based on Online Behaviour for Identifying Potential Risks
Upload Dataset
Pre Processing
Splitting
GradientBoosting
DNN
Graph
Predict
Close

```
[47581 rows x 30 columns]

-----Preprocessing-----

    Device_Type  Malware_Detection  Phishing_Attempts  Social_Media_Usage  ...  year  second  minute  hour
0         2          0          0          1  ...  2017    0    0    0
1         2          0          0          1  ...  2017    0    0    3
2         2          0          1          2  ...  2017    0    0    4
3         2          0          0          2  ...  2017    0    0    5
4         3          0          1          1  ...  2017    0    0    7
...      ...        ...        ...        ...  ...   ...  ...  ...  ...
47576     2          0          0          1  ...  2024    0    0   19
47577     2          0          0          1  ...  2024    0    0   21
47578     2          0          0          1  ...  2024    0    0   22
47579     1          0          0          1  ...  2024    0    0   23
47580     0          1          1          2  ...  2024    0    0    0

[47581 rows x 35 columns]

-----Splitting-----

X-train(88000, 2), Y-train(88000,)

X-test(22000, 2), Y-test(22000,)
```

9.3.4 : Data Splitting in the GUI

This figure demonstrates how the dataset is divided into training and testing subsets within the GUI interface. The dataset is split into appropriate proportions, ensuring a balance between training and validation. The training set is used to build the machine learning models, while the test set evaluates their performance. This step is crucial for assessing model generalization and preventing overfitting.

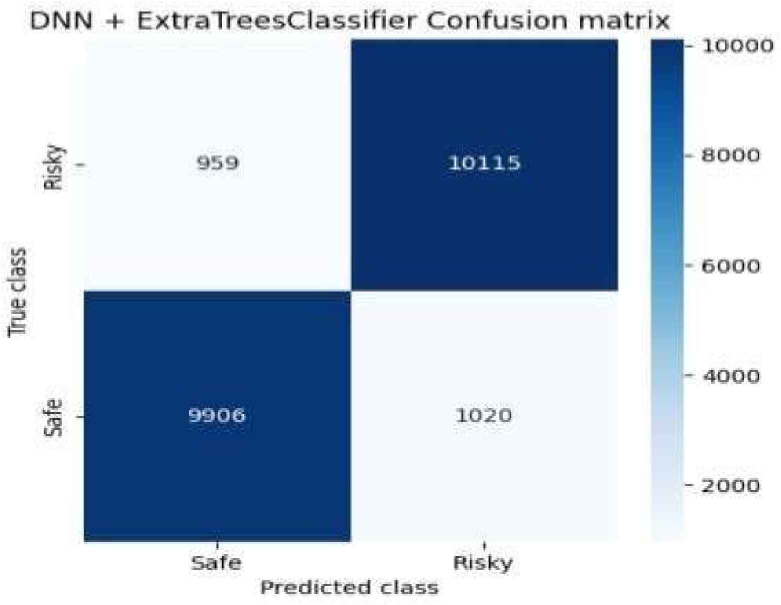


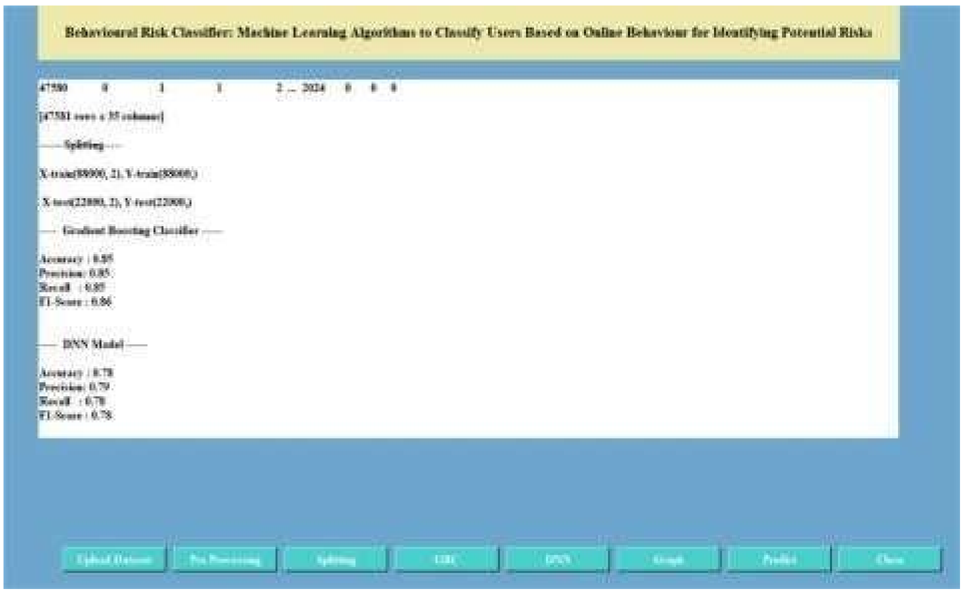

Behavioural Risk Classifier: Machine Learning Algorithms to Classify Users Based on Online Behaviour for Identifying Potential Risks
Upload Dataset
Pre Processing
Splitting
GBC
SVM
Graph
Predict
Close

```
2        2        0        1        1 ... 2017   0   0   4
3        2        0        0        1 ... 2017   0   0   5
4        3        0        1        1 ... 2017   0   0   7
...
47576    2        0        0        1 ... 2024   0   0   19
47577    2        0        0        1 ... 2024   0   0   22
47578    2        0        0        1 ... 2024   0   0   22
47579    1        0        0        1 ... 2024   0   0   23
47580    0        1        1        2 ... 2024   0   0   0

[47581 rows x 35 columns]

------Splitting------

X-train(80000, 2), Y-train(80000,)

 X-test(22000, 2), Y-test(22000,)

------ Gradient Boosting Classifier ------

Accuracy : 0.88
Precision: 0.87
Recall   : 0.9
F1-Score : 0.89
```

9.3.5 : Performance Metrics and Confusion Matrix Plot of GBC

Classifier Model This figure illustrates the performance of the Gradient Boosting Classifier (GBC) model. It includes accuracy, precision, recall, and F1-score values, which indicate the model’s effectiveness in classifying cybersecurity behaviors. The confusion matrix visually represents the model's classification results by showing true positive, true negative, false positive, and false negative counts. The analysis highlights areas where the GBC model performs well and where misclassifications occur.

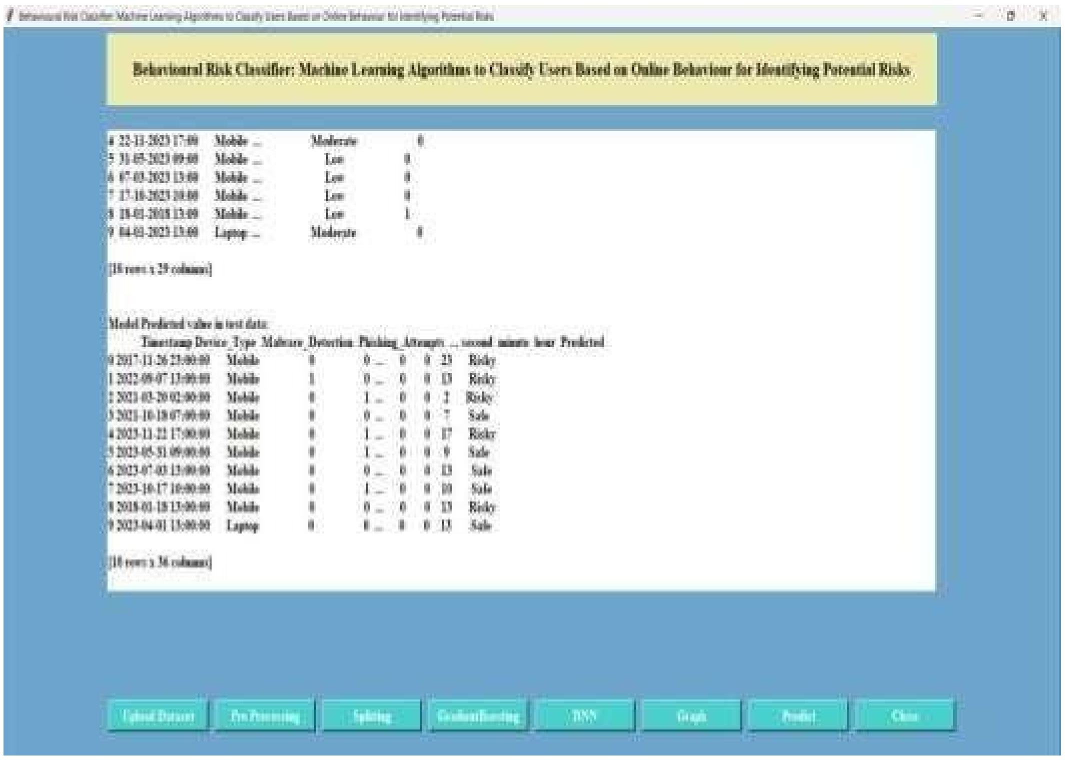



Behavioural Risk Classifier: Machine Learning Algorithms to Classify Users Based on Online Behaviour for Identifying Potential Risks

```
47580     0       1       1       2 ... 2024   0   0   0
[47581 rows x 35 columns]
-----Splitting-----
X-train(88000, 2), Y-train(88000,)
 X-test(22000, 2), Y-test(22000,)
----- Gradient Boosting Classifier -----
Accuracy : 0.85
Precision: 0.85
Recall   : 0.85
F1-Score : 0.86

----- DNN Model -----
Accuracy : 0.78
Precision: 0.79
Recall   : 0.78
F1-Score : 0.78
```

9.3.6 : Performance Metrics Plot of DNN Classifier Model

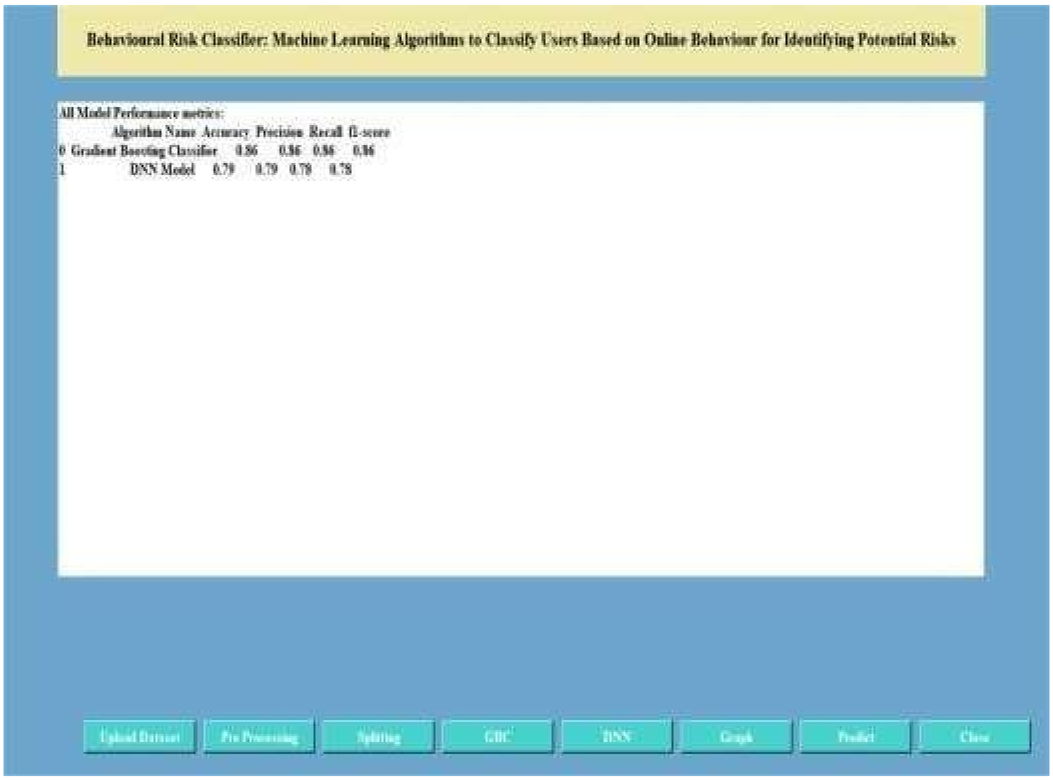
This figure presents the evaluation of the Deep Neural Network (DNN) classifier. The metrics, including accuracy, precision, recall, and F1-score, demonstrate a higher classification performance compared to the GBC model. The confusion matrix provides insight into correct and incorrect predictions, revealing the model's robustness in identifying different cybersecurity behavior categories. The results confirm the effectiveness of deep learning techniques in analyzing complex behavioral patterns.


Behavioural Risk Classifier: Machine Learning Algorithms to Classify Users Based on Online Behaviour for Identifying Potential Risks
Behavioural Risk Classifier: Machine Learning Algorithms to Classify Users Based on Online Behaviour for Identifying Potential Risks
Upload Dataset
Pre Processing
Splitting
GradientBoosting
DNN
Graph
Predict
Close

```
4 22-11-2023 17:00  Mobile ...      Moderate        0
5 31-05-2023 09:00  Mobile ...        Low       0
6 07-03-2023 13:00  Mobile ...        Low       0
7 17-10-2023 10:00  Mobile ...        Low       0
8 18-01-2018 13:00  Mobile ...        Low       1
9 04-03-2023 13:00  Laptop ...      Moderate        0

[10 rows x 29 columns]


Model Predicted value in test data:
        Timestamp Device_Type  Malware_Detection  Phishing_Attempts  ... second  minute  hour  Predicted
0 2017-11-26 23:00:00      Mobile                  0                  0  ...      0       0    23      Risky
1 2022-09-07 13:00:00      Mobile                  1                  0  ...      0       0    13      Risky
2 2023-03-20 02:00:00      Mobile                  0                  1  ...      0       0     2      Risky
3 2021-10-18 07:00:00      Mobile                  0                  0  ...      0       0     7       Safe
4 2023-11-22 17:00:00      Mobile                  0                  1  ...      0       0    17      Risky
5 2023-05-31 09:00:00      Mobile                  0                  1  ...      0       0     9       Safe
6 2023-07-03 13:00:00      Mobile                  0                  0  ...      0       0    13       Safe
7 2023-10-17 10:00:00      Mobile                  0                  1  ...      0       0    10       Safe
8 2018-01-18 13:00:00      Mobile                  0                  0  ...      0       0    13      Risky
9 2023-04-01 13:00:00      Laptop                  0                  0  ...      0       0    13       Safe

[10 rows x 36 columns]
```

9.3.7 : Model Prediction on the Test Data

This figure showcases the final predictions generated by the trained model on the test dataset. The GUI interface displays predicted cybersecurity risk categories based on user behavior attributes. The results help assess how well the model generalizes to unseen data. The visualization provides insights into real-world applicability, demonstrating the model's ability to classify users based on cyber security risks.


Behavioural Risk Classifier: Machine Learning Algorithms to Classify Users Based on Online Behaviour for Identifying Potential Risks
Upload Dataset
Pre Processing
Splitting
GBC
DNN
Graph
Predict
Close

```
All Model Performance metrics:
           Algorithm Name  Accuracy  Precision  Recall  f1-score
0  Gradient Boosting Classifier     0.86       0.86    0.86      0.86
1                   DNN Model     0.79       0.79    0.78      0.78
```

9.3.8 : Performance Comparison of Models

This figure compares the performance of the Gradient Boosting Classifier (GBC) and the Deep Neural Network (DNN) model. The graph visually represents accuracy, precision, recall, and F1-score values for both models. The comparison highlights the superior performance of the DNN model, which achieves higher accuracy and reliability. This analysis aids in selecting the best approach for cybersecurity risk classification and demonstrates the effectiveness of deep learning in this domain.

CHAPTER 10
CONCLUSION AND FUTURE SCOPE

10.1 Conclusion

The research successfully analyzes cybersecurity behaviors and online activity patterns using machine learning techniques. By leveraging a dataset containing various security threats, online habits, and risk indicators, the study effectively identifies factors contributing to cybersecurity vulnerabilities. The implementation of Gradient Boosting Classifier (GBC) and Deep Neural Network (DNN) classifiers enables accurate categorization of users based on their cybersecurity risk levels. The preprocessing steps, including data cleaning, feature selection, and splitting, ensure optimal model performance. The results demonstrate the importance of continuous monitoring, awareness, and proactive security measures in reducing online threats.

10.2 Future Scope

- ***Enhanced Feature Engineering***

  Incorporating additional behavioral metrics, such as real-time browsing patterns and biometric authentication data, to improve model accuracy.

- ***Real-Time Threat Detection***

  Deploying the model in real-time applications for continuous cybersecurity monitoring and automated risk alerts.

- ***Adaptive Learning Models***

  Implementing reinforcement learning or self-adaptive models that dynamically adjust based on evolving cyber threats.

- ***Integration with Cybersecurity Systems***

  Integrating the predictive model with firewalls, antivirus

Z

software, and enterprise security solutions to enhance threat mitigation strategies.

- ***Cross-Platform Analysis***

  Expanding the dataset to include multiple devices and operating systems for broader applicability across different user environment.

- ***User Awareness and Training***

  Developing personalized security awareness programs based on user behavior predictions, reducing susceptibility to cyber threats.

- ***Explainable AI for Cybersecurity***

  Enhancing model interpretability to provide insights into decision- making, helping security analysts understand and trust AI-based.

APPENDIX

```
from tkinter import * import tkinter
from tkinter import filedialog
from tkinter.filedialog import askopenfilename from tkinter import simpledialog
from tkinter import Button, Text, Scrollbar import numpy as np
import pandas as pd import warnings
warnings.filterwarnings('ignore') import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE from sklearn.decomposition import
PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler from sklearn.utils
import resample
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix from
sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score from sklearn.metrics import
recall_score from sklearn.metrics import f1_score  import os, joblib
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout from
tensorflow.keras.callbacks import EarlyStopping
from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier
global filename global df
global x,y,x_train,x_test,y_train,y_test,predict global le,scaler
global test
global model_file main = tkinter.Tk()
main.title("Behavioural Risk Classifier: Machine Learning Algorithms to Classify
Users Based on Online Behaviour for Identifying Potential Risks")
main.geometry('1200x700') main.config(bg='skyblue3') def upload():
global filename global df
```

Z

```
filename = filedialog.askopenfilename() df=pd.read_csv(filename)
text.insert(tkinter.END,filename+'loaded\n\n') text.insert(tkinter.END,str(df.shape))
text.insert(tkinter.END,str(df))
def preprocessing(): global df
df['Timestamp'] = pd.to_datetime(df['Timestamp']) df['day'] =
df['Timestamp'].dt.day
df['month'] = df['Timestamp'].dt.month df['year'] = df['Timestamp'].dt.year
df['second'] = df['Timestamp'].dt.second df['minute'] = df['Timestamp'].dt.minute
df['hour'] = df['Timestamp'].dt.hour
df = df.drop(['Timestamp'], axis = 1) le=LabelEncoder()
df['Device_Type']=le.fit_transform(df['Device_Type'])
df['Social_Media_Usage']=le.fit_transform(df['Social_Media_Usage'])
df['Password_Strength']=le.fit_transform(df['Password_Strength'])

df['Education_Content_Usage']=le.fit_transform(df['Education_Content_Usage'])
df['Age_Group']=le.fit_transform(df['Age_Group'])
df['Geolocation']=le.fit_transform(df['Geolocation'])
df['Network_Type']=le.fit_transform(df['Network_Type'])
df['Peer_Interactions']=le.fit_transform(df['Peer_Interactions'])
df['E_Safety_Awareness_Score']=le.fit_transform(df['E_Safety_Awareness_Score'])
df['Cybersecurity_Behavior_Category']=le.fit_transform(df['Cybersecurity_Behavior
_Ca tegory'])
text.insert(tkinter.END,'\n\n-------Preprocessing          \n\n')
text.insert(tkinter.END,str(df)) category_order = ['Safe','Risky']
plt.figure(figsize=(10, 6))
ax = sns.countplot(x= df['Cybersecurity_Behavior_Category']) plt.xlabel('System
Status')
ax.set_xticklabels(category_order) plt.ylabel('Count')
plt.title('Count of Cybersecurity Category') for p in ax.patches:
ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
ha='center', va='center', fontsize=10, color='black', xytext=(0, 5), textcoords='offset
points')
plt.show() def splitting():
global df,scaler, pca
global x,y,x_train,x_test,y_train,y_test
x=df.drop(['Cybersecurity_Behavior_Category'], axis = 1)
y=df['Cybersecurity_Behavior_Category']
smote = SMOTE(random_state=42) X_smote, y_smote = smote.fit_resample(x, y)
X_sample, y_sample = resample(X_smote, y_smote, n_samples = 110000) scaler =
StandardScaler()
X_scaled = scaler.fit_transform(X_sample)
n_components = 2  # Number of principal components to retain pca =
PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled) x_train,x_test,y_train,y_test=
train_test_split(X_pca,y_sample,test_size=0.2,random_state=40)
text.insert(tkinter.END,'\n\n-------Splitting        \n\n')
text.insert(tkinter.END,"X-train"+str(x_train.shape)+ ", Y-train"+str(y_train.shape))
text.insert(tkinter.END,"\n\n X-test"+str(x_test.shape)+ ", Y-test"+str(y_test.shape))
labels=['Safe','Risky'] precision = []
recall = [] fscore = [] accuracy = []
def calculateMetrics(algorithm, predict, testY): global labels
testY = testY.astype('int') predict = predict.astype('int')
p = precision_score(testY, predict,average='macro') * 100 r = recall_score(testY,
predict,average='macro') * 100
f = f1_score(testY, predict,average='macro') * 100 a =
accuracy_score(testY,predict)*100 accuracy.append(a)
precision.append(p) recall.append(r) fscore.append(f)
print(algorithm+' Accuracy              : '+str(a)) print(algorithm+' Precision  :
'+str(p)) print(algorithm+' Recall         : '+str(r)) print(algorithm+' FSCORE
  : '+str(f))
report=classification_report(predict, testY,target_names=labels)
print('\n',algorithm+" classification report\n",report) text.insert(tkinter.END,'\n\n-----
'+algorithm+'     \n\n')
text.insert(tkinter.END,algorithm + ' Accuracy:   ' + str(a)+'\n\n')
text.insert(tkinter.END,algorithm + ' Precision:  ' + str(p)+'\n\n')
text.insert(tkinter.END,algorithm + ' Recall:    ' + str(r)+'\n\n')
text.insert(tkinter.END,algorithm+ '  FSCORE:          ' + str(f)+'\n\n') conf_matrix =
confusion_matrix(testY, predict)
```

Z

```
plt.figure(figsize =(5, 5))
ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot =
True, cmap="Blues" ,fmt ="g");
ax.set_ylim([0,len(labels)]) plt.title(algorithm+" Confusion matrix") plt.ylabel('True
class') plt.xlabel('Predicted class')
plt.show()
def GBCModel():
global x_train, x_test, y_train, y_test clf = GradientBoostingClassifier()
clf.fit(x_train, y_train) predict=clf.predict(x_test) #joblib.dump(clf, 'model/gbc.pkl')
print("GradientBoostingClassifier model trained and model weights saved.")
calculateMetrics("GradientBoostingClassifier", predict, y_test)
def DNNModel():
global extractor, rfc, x_train, x_test, y_train, y_test global predict, sc
#scaler = StandardScaler()
#x_train = scaler.fit_transform(x_train) #x_test = scaler.transform(x_test)
models_folder = "model"
dnn_features_path = os.path.join(models_folder, 'dnn_features.pkl')
etc_model_path = os.path.join(models_folder, 'rfc_model.pkl') dnn_extractor_path
= os.path.join(models_folder, 'dnn_extractor.h5') print('x_train.shape:',
x_train.shape)
print('y_train.shape:', y_train.shape)
if os.path.exists(dnn_features_path) and os.path.exists(etc_model_path):
x_train_features = joblib.load(dnn_features_path)
rfc = joblib.load(etc_model_path)


if os.path.exists(dnn_extractor_path): extractor = load_model(dnn_extractor_path)
x_test_features = extractor.predict(x_test)
else:
raise FileNotFoundError("DNN extractor model not found in the models folder.") #
Predict with ETC using extracted test features
y_pred = rfc.predict(x_test_features) print('x_test_features.shape:',
x_test_features.shape) print('y_pred:', y_pred)
calculateMetrics("DNN + ExtraTreesClassifier", y_pred, y_test) else:
```

```
# Define and train the DNN model model =
Sequential([ Dense(128, activation='relu',
input_shape=(x_test.shape[1],)), BatchNormalization(),
Dropout(0.3),
Dense(64, activation='relu'), BatchNormalization(), Dropout(0.3),
Dense(32, activation='relu')  # Feature extraction layer
])
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
early_stopping  =  EarlyStopping(monitor='val_loss',  patience=5,
restore_best_weights=True)
model.fit(x_train,  y_train,  epochs=20,  validation_split=0.2,
callbacks=[early_stopping], verbose=0)
extractor = Sequential(model.layers[:-1]) extractor.save(dnn_extractor_path)
x_train_features = extractor.predict(x_train) x_test_features =
extractor.predict(x_test)
# Save extracted features joblib.dump(x_train_features, dnn_features_path)

# Train Extra Trees Classifier
rfc  =  RandomForestClassifier()#(n_estimators=200,
max_features='sqrt', min_samples_split=4, random_state=42)
rfc.fit(x_train_features, y_train) # Save ETC model
joblib.dump(rfc, etc_model_path) # Predict and evaluate
y_pred = rfc.predict(x_test_features)
print("DNN + ExtraTreesClassifier model trained and saved.")
calculateMetrics("DNN + ExtraTreesClassifier", y_pred, y_test)
def predict():
global transform, rfc, extractor, labels, scaler, pca
file = filedialog.askopenfilename(initialdir="Datasets") test1 = pd.read_csv(file)
test = test1 text.delete('1.0', END)
text.insert(END, f'{file} Loaded\n')
text.insert(END, "\n\nLoaded test data: \n" + str(test) + "\n") test['Timestamp'] =
pd.to_datetime(test['Timestamp']) test['day'] = test['Timestamp'].dt.day
test['month'] = test['Timestamp'].dt.month test['year'] = test['Timestamp'].dt.year
```

Z

```
test['second'] = test['Timestamp'].dt.second test['minute'] =
test['Timestamp'].dt.minute test['hour'] = test['Timestamp'].dt.hour
test = test.drop(['Timestamp'], axis = 1) le=LabelEncoder()
test['Device_Type']=le.fit_transform(test['Device_Type'])
test['Social_Media_Usage']=le.fit_transform(test['Social_Media_Usage'])
test['Password_Strength']=le.fit_transform(test['Password_Strength'])
test['Education_Content_Usage']=le.fit_transform(test['Education_Content_Usage'])
test['Age_Group']=le.fit_transform(test['Age_Group'])
test['Geolocation']=le.fit_transform(test['Geolocation'])
test['Network_Type']=le.fit_transform(test['Network_Type'])

test['Peer_Interactions']=le.fit_transform(test['Peer_Interactions'])
test['E_Safety_Awareness_Score']=le.fit_transform(test['E_Safety_Awareness_Score'
])
test_scaled = scaler.transform(test) test_pca = pca.transform(test_scaled)
test_features = extractor.predict(test_pca) y_pred = rfc.predict(test_features)
predicted_labels = [labels[p] for p in y_pred] test1['Predicted'] = predicted_labels
text.insert(END, "\n\nModel Predicted value in test data: \n" + str(test1) + "\n") def
graph():
columns = ["Algorithm Name", "Accuracy", "Precision", "Recall", "f1-score"]
algorithm_names = ["Gradient Boosting Classifier", "DNN Model"]
# Combine metrics into a DataFrame values = []
for i in range(len(algorithm_names)):
values.append([algorithm_names[i], accuracy[i], precision[i], recall[i], fscore[i]])
temp = pd.DataFrame(values, columns=columns)
text.delete('1.0', END)
# Insert the DataFrame in the text console text.insert(END, "All Model Performance
metrics:\n") text.insert(END, str(temp) + "\n")
# Plotting the performance metrics
metrics = ["Accuracy", "Precision", "Recall", "f1-score"]
index = np.arange(len(algorithm_names))  # Positions of the bars # Set up the figure
and axes
fig, ax = plt.subplots(figsize=(10, 6)) bar_width = 0.2  # Width of the bars opacity =
0.8
```

```
# Plotting each metric with an offset
plt.bar(index, accuracy, bar_width, alpha=opacity, color='b', label='Accuracy')
plt.bar(index + bar_width, precision, bar_width, alpha=opacity,
color='g',
label='Precision')
plt.bar(index + 2 * bar_width, recall, bar_width,
  alpha=opacity, color='r', label='Recall')

plt.bar(index + 3 * bar_width, fscore, bar_width, alpha=opacity, color='y',
label='f1- score')
# Labeling the chart plt.xlabel('Algorithm') plt.ylabel('Scores')
plt.title('Performance Comparison of All Models')
plt.xticks(index + bar_width, algorithm_names)    # Setting the labels for x-
axis (algorithms)
plt.legend()
# Display the plot plt.tight_layout() plt.show()
def close():
main.destroy() # Title
title = tkinter.Label(main, text="Behavioural Risk Classifier: Machine Learning
Algorithms to Classify Users Based on Online Behaviour for Identifying Potential
Risks", justify='center')
font = ('times', 15, 'bold')
title.config(bg='pale goldenrod', fg='black', font=font, height=3, width=100)
title.place(x=150, y=6)
# Text Box
font1 = ('times', 12, 'bold')
text = Text(main, height=25, width=150) scroll = Scrollbar(text)
text.configure(yscrollcommand=scroll.set) text.place(x=150, y=106)
text.config(font=font1)
# Make the `text` widget global main.text_widget = text
# Button Font
button_font = ('times', 12, 'bold') # Buttons at the Bottom
button_frame = tkinter.Frame(main, bg='skyblue3')
```

Z

```
button_frame.place(relx=0.5, rely=0.9, anchor='center') button_specs =
[ ("Upload Dataset", upload),
("Pre Processing", preprocessing), ("Splitting", splitting), ("GradientBoosting",
GBCModel), ("DNN", DNNModel),
("Graph", graph), ("Predict", predict), ("Close", close)
for btn_text, btn_command in button_specs:
button = Button(button_frame, text=btn_text, command=btn_command,
bg='medium turquoise', fg='white', width=15)
button.config(font=button_font) button.pack(side='left', padx=5)
main.mainloop()
```

REFERENCES

[1] X. Wang, F. Chao, G. Yu and K. Zhang, "Factors influencing fake news rebuttal acceptance during the COVID-19 pandemic and the moderating effect of cognitive ability", *Compute. Hum. Behav.*, vol. 130, May 2022.

[2] Halevy, C. Canton-Ferrer, H. Ma, U. Ozertem, P. Pantel, M. Saeidi, et al., "Preserving integrity in online social networks", *Commun. ACM*, vol. 65, no. 2,
pp. 92-98, Feb. 2022.

[3] S.-H. Liao, R. Widowati and C.-J. Cheng, "Investigating Taiwan Instagram users' behaviors for social media and social commerce development", *Entertainment Comput.*, vol. 40, Jan. 2022.

[4] P. Bedi, S. B. Goyal, A. S. Rajawat, R. N. Shaw and A. Ghosh, "A framework for personalizing atypical web search sessions with concept-based user profiles using selective machine learning techniques", *Advanced Computing and Intelligent Technologies*, vol. 218, 2022.

[5] A. Mohammed and A. Ferraris, "Factors influencing user participation in social media: Evidence from Twitter usage during COVID-19 pandemic in Saudi Arabia", *Technol. Soc.*, vol. 66, Aug. 2021.

[6] P. Ren, Z. Liu, X. Song, H. Tian, Z. Chen, Z. Ren, et al., "Wizard of search engine: Access to information through conversations with search engines", *Proc. 44th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jul. 2021.

[7] S.-F. Tsao, H. Chen, T. Tisseverasinghe, Y. Yang, L. Li and Z. A. Butt, "What social media told us in the time of COVID-19: A scoping review", *Lancet Digit. Health*, vol. 3, no. 3, pp. e175-e194, Mar. 2021.

[8] S. K. Shivakumar, "A survey and taxonomy of intent-based code search", *Int.J. Softw. Innov.*, vol. 9, no. 1, pp. 69-110, Jan. 2021. D. Fraszczak, "Information propagation in online social networks—A simulation case study", *Proc. 38th Int. Bus. Inf. Manag. Assoc.*, pp. 23-24, 2021.

Z

[9] D. Koehn, S. Lessmann and M. Schaal, "Predicting online shopping behaviour from clickstream data using deep learning", *Expert Syst. Appl.*, vol. 150, Jul. 2020.

[10] H. Yoganarasimhan, "Search personalization using machine learning", *Manage. Sci.*, vol. 66, no. 3, pp. 1045-1070, Mar. 2020.

[11] S. Edgerly, R. R. Mourão, E. Thorson and S. M. Tham, "When do audiences verify? How perceptions about message and source influence audience verification of news headlines", *Journalism Mass Commun. Quart.*, vol. 97, no. 1, pp. 52-71, Mar. 2020.

[12] X. Lin and X. Wang, "Examining gender differences in people's information- sharing decisions on social networking sites", *Int. J. Inf. Manage.*, vol. 50, pp. 45-56, Feb. 2020.

[13] J. Liu, M. Mitsui, N. J. Belkin and C. Shah, "Task information seeking intentions and user behavior: Toward a multi-level understanding of web search", *Proc. Conf. Human Inf. Interact. Retr.*, pp. 123-132, Mar. 2019.

[15] C. I. Eke, A. A. Norman, L. Shuib and H. F. Nweke, "A survey of user profiling: State-of-the-art challenges and solutions", *IEEE Access*, vol. 7, pp. 144907-144924, 2