**PERSONALIZED AD TARGETING SYSTEM USING MACHINE LEARNING FOR OPTIMIZING AD DELIVERY**

**ABSTRACT**

The increasing reliance on digital platforms for marketing and advertising has made personalized ad delivery a critical component of modern business strategies. Traditional advertising systems often fail to adapt to user preferences dynamically, leading to suboptimal engagement and conversion rates. This aims to address these shortcomings by leveraging advanced machine learning techniques to enhance ad targeting and delivery. The system utilizes datasets comprising user demographics, browsing behavior, device information, and temporal patterns to predict the likelihood of ad clicks. Data preprocessing techniques, including handling missing values, encoding categorical data, and balancing imbalanced datasets using SMOTE, ensure data quality and enhance predictive accuracy. Various machine learning models, such as Gradient Boosting Classifier (GBC), Decision Tree Classifier (DTC), and a hybrid Feed-Forward Neural Network (FFNN) with Random Forest (RF), are employed to evaluate their effectiveness in predicting ad click-through rates (CTR). Performance metrics such as accuracy, precision, recall, and F1-score are analyzed to determine the best-performing algorithm. The proposed system integrates explainable AI and visualization tools to assist advertisers in understanding user behavior and improving targeting strategies. Unlike traditional systems that rely on static rule-based methods, this approach provides dynamic and data-driven insights, optimizing ad delivery in real time. The system's modularity ensures adaptability across various industries and datasets, making it scalable for widespread adoption. By bridging the gap between user preferences and ad targeting, this project highlights the significance of machine learning in driving personalized marketing solutions. The outcomes have the potential to significantly increase ROI for advertisers, improve user engagement, and set a benchmark for intelligent ad targeting systems in the digital marketing domain.

**CHAPTER 1**

**INTRODUCTION**

**1.1 Background and Overview**

Digital advertising has evolved significantly from traditional print and television ads to highly personalized, data-driven strategies. Historically, advertisers relied on broad demographic segmentation and static keyword-based targeting, which often resulted in inefficiencies and lower engagement rates. In India, with over 900 million internet users as of 2023, digital marketing presents an enormous opportunity for businesses. Reports suggest that digital ad spending in India will reach $21 billion by 2025, emphasizing the growing importance of personalized advertising. Traditional ad delivery methods often failed to account for dynamic user behavior, leading to wasted ad impressions and reduced return on investment (ROI). With advancements in artificial intelligence and machine learning, businesses can now leverage data-driven insights to optimize ad delivery. Personalized Ad Targeting Systems using machine learning analyze user preferences, browsing history, demographics, and real-time interactions to deliver tailored ads, increasing engagement and conversion rates. Unlike traditional approaches, AI-driven systems continuously adapt, ensuring the right ads are shown to the right users at the right time. This enhances user experience while maximizing advertiser revenue. By incorporating machine learning, businesses can transition from static rule-based advertising to dynamic, self-improving models that drive efficiency and reduce costs. The adoption of AI-powered ad targeting represents a fundamental shift in digital marketing, shaping the future of personalized advertising and optimizing ad spend.

**1.2 Problem Definition**

Before the integration of machine learning, digital advertising faced several inefficiencies that limited its effectiveness. Traditional ad targeting methods relied on predefined demographic segmentation, which often resulted in generalized ad delivery rather than personalized experiences. Keyword-based targeting methods struggled with context and relevance, leading to ads being displayed to users who had no real interest in the products or services being promoted. Manual optimization techniques required constant human intervention, making it difficult to scale personalized ad campaigns effectively. Additionally, static rules for ad placement failed to account for dynamic user behavior, device usage, and real-time trends, reducing the effectiveness of ad delivery. Advertisers experienced high ad fatigue, where users became unresponsive to repeated advertisements, leading to decreased engagement and lower conversion rates. Furthermore, the lack of adaptive learning mechanisms meant that businesses had limited insights into changing consumer preferences. These challenges resulted in wasted resources, lower ROI, and reduced efficiency in digital marketing campaigns. Without AI-driven optimization, traditional ad systems struggled to balance ad relevance with user experience, making it difficult for businesses to achieve meaningful engagement.

**1.3 Research Motivation**

The growing complexity of user behavior in digital spaces has highlighted the need for intelligent ad targeting mechanisms that go beyond traditional approaches. With millions of daily online interactions, businesses require systems that can automatically analyze user preferences and deliver ads in a context-aware manner. Studies show that personalized advertising can improve engagement rates by over 40% and boost conversion rates significantly. In India, where digital advertising is rapidly expanding, businesses must adopt AI-driven approaches to remain competitive. The emergence of big data, combined with advancements in deep learning and predictive analytics, presents a unique opportunity to revolutionize digital marketing. Machine learning models can process vast amounts of user data, learn from past interactions, and dynamically optimize ad placements for maximum efficiency. The motivation behind this research is to develop an adaptive, real-time ad targeting system that enhances both user experience and advertiser revenue. By reducing irrelevant ad impressions and improving targeting accuracy, businesses can significantly improve their ROI. Moreover, addressing issues like ad fatigue and banner blindness can lead to higher engagement rates, making personalized advertising a necessity rather than an option. The integration of AI in ad targeting represents a crucial step toward the future of marketing, where efficiency, personalization, and user satisfaction are prioritized.

**1.4 Need**

The rise of digital platforms has made advertising a primary source of revenue for businesses, but traditional methods fail to capitalize on the full potential of personalization. A machine learning-driven Personalized Ad Targeting System is essential for optimizing ad delivery by making intelligent, data-driven decisions. Businesses need this system to enhance engagement rates, increase conversions, and improve customer satisfaction. With millions of ads being displayed daily, ensuring relevance is critical to maintaining user interest and avoiding ad fatigue. Additionally, real-time learning allows businesses to continuously refine their targeting strategies, adapting to changes in user behavior and market trends. The ability to analyze browsing history, device preferences, demographics, and contextual data ensures that ads are highly relevant to individual users. This project is also crucial for improving cost efficiency, as advertisers can maximize their budget by targeting only the most relevant users. Moreover, AI-driven ad targeting can enhance brand perception by delivering meaningful and non-intrusive ad experiences. As digital marketing becomes increasingly competitive, businesses that fail to leverage machine learning risk falling behind in efficiency and customer engagement. The need for this project is driven by the fundamental shift in consumer expectations, where personalized experiences are now the standard rather than the exception.

**1.5 Applications**

* **E-commerce Advertising** – Optimizes product recommendations based on user preferences and browsing history.
* **Social Media Marketing** – Enhances ad targeting on platforms like Facebook, Instagram, and Twitter.
* **Video Ad Targeting** – Delivers personalized video ads on platforms like YouTube based on user engagement patterns.
* **Search Engine Advertising** – Improves relevance in Google Ads by predicting click-through probabilities.
* **Mobile App Advertising** – Optimizes in-app ad placements based on user behavior and app usage trends.
* **Retail and Local Business Promotions** – Enhances hyper-local targeting for small businesses.
* **Subscription-Based Services** – Improves ad delivery for OTT platforms like Netflix and Hotstar.
* **News and Content Platforms** – Personalizes sponsored content recommendations for better audience engagement.

**CHAPTER 2**

**LITERATURE SURVEY**

Ungureanu and Popescu [1] investigated the evolution of online advertising and explored its historical challenges. They analyzed the transition from traditional to digital advertising strategies in detail. Their work provided insights into the transformation of adtech methodologies. The study underscored the importance of adapting to technological changes in the advertising landscape. Watkins [2] presented a comprehensive guide to advertising technology. He reviewed key innovations and underlying mechanisms driving digital marketing advancements. His work examined the impact of technological progress on advertising practices. The study clarified fundamental principles that have shaped the adtech sector. Ezzat [3] explored the historical development of advertising through the lens of media technology. The research documented significant technological milestones that influenced marketing communications. Ezzat analyzed how evolving media platforms transformed advertising methods. The study offered a historical perspective on the interplay between technology and marketing.

Dahlén and Edenius [4] compared consumer responses to traditional versus non-traditional advertising media. They conducted experiments to measure engagement and perception across different formats. Their findings revealed variations in audience reactions based on ad presentation. The study laid the groundwork for understanding the effectiveness of diverse advertising strategies. Studer et al. [5] proposed a machine learning process model integrated with quality assurance methodology. They developed the CRISP-ML (Q) framework to ensure transparency and reliability in ML systems. Their research focused on incorporating QA techniques throughout the machine learning lifecycle. The study highlighted the significance of systematic validation in automated decision-making processes. Wall and Fontenot [6] conducted a comparative analysis of machine learning models to predict quality assurance outcomes in radiation therapy planning. They evaluated various algorithms to determine the most effective approach in a healthcare context. Their findings established performance benchmarks for ML applications. The study supported the integration of machine learning in treatment optimization processes.

Khankhoje [7] examined quality assurance practices within machine learning applications. He outlined techniques to verify and validate ML outputs in automated systems. The research addressed challenges in maintaining high-quality and accountable results. The study contributed frameworks to enhance the reliability of ML-driven processes. Lee and Cho [8] analyzed current trends and future prospects in digital advertising. They reviewed the influence of emerging technologies on ad targeting and campaign optimization. Their work provided a critical evaluation of innovations shaping the digital marketing landscape. The study forecasted potential growth areas within the adtech sector. Choi and Lim [9] identified various machine learning techniques for the classification of target advertising. They compared algorithm performance to determine the most precise methods for ad targeting. Their research examined both the strengths and limitations of the evaluated techniques. The study advanced understanding of how ML methods refine targeted advertising strategies.

Ullal et al. [10] investigated the role of machine learning in digital marketing strategies. They analyzed how ML improved targeting accuracy and personalization in advertising campaigns. Their research demonstrated measurable enhancements in campaign performance through algorithmic innovation. The study underscored ML’s transformative influence on digital marketing practices. Miralles-Pechuán et al. [11] presented a novel methodology that employed genetic algorithms to optimize display advertising campaigns. They developed an approach aimed at increasing click-through rates and conversion metrics. Their work integrated evolutionary techniques to enhance campaign performance. The study offered valuable insights into algorithmic strategies for digital ad optimization. Nuara et al. [12] conducted a study on the online joint optimization of bid strategies and daily budgets in internet advertising campaigns. They developed an AI-based framework that balanced bid prices with budget constraints to maximize ad performance. Their research provided a model for real-time campaign optimization. The study advanced the field by proposing methods to enhance financial efficiency in digital advertising.

Nuara [13] explored the application of machine learning algorithms for optimizing internet advertising campaigns. He reviewed various algorithmic approaches to improve performance metrics such as click-through rates and conversions. His research critically evaluated ML strategies within the adtech industry. The study contributed to a deeper understanding of campaign optimization through machine learning. Daghistani and Alshammari [14] compared logistic regression with random forest techniques in predicting diabetes. They conducted empirical analyses to evaluate performance differences between statistical and machine learning approaches. Their findings demonstrated the advantages of ensemble methods in complex prediction tasks. The study highlighted the potential of ML models in enhancing healthcare prediction accuracy. Greene et al. [15] discussed unsupervised learning and clustering techniques for multimedia organization and retrieval. They conducted case studies to illustrate the extraction of meaningful patterns from unlabelled data. Their research provided practical insights into the application of clustering methods in multimedia contexts. The study contributed to the development of effective data organization strategies.

Cremonesi et al. [16] performed analytic performance modeling and analysis of detailed neuron simulations. They evaluated computational efficiency in simulating complex neural networks. Their research established benchmarks for high-performance computing in neural simulations. The study advanced understanding of simulation performance in computational environments. Phillips [17] analyzed the return-on-investment (ROI) process, highlighting prevalent issues and emerging trends. He examined methodologies for evaluating ROI in various technological and educational contexts. His work discussed the evolution of ROI measurement techniques. The study offered insights into improving the accuracy of investment effectiveness assessments.

Ichsani and Suhardi [18] investigated the effect of return on equity (ROE) and ROI on trading volume. They conducted statistical analyses to explore the relationships between financial indicators and market performance. Their findings emphasized the significant role of these financial metrics in influencing trading behavior. The study provided an empirical perspective on the dynamics of financial performance in trading. Dahmen and Cook [19] developed SynSys, a synthetic data generation system for healthcare applications. They presented a method for creating realistic synthetic datasets while preserving critical statistical properties. Their research focused on enhancing data privacy without compromising analytical utility. The study demonstrated the effectiveness of synthetic data in supporting healthcare research and analytics.

**CHAPTER 3**

**EXISTING SYSTEM**

Before the advent of machine learning-driven ad targeting, digital advertising primarily relied on predefined rules and manual optimization techniques. The traditional system used demographic segmentation, keyword-based targeting, and contextual advertising to serve ads to users. Advertisers categorized audiences based on broad attributes such as age, gender, location, and interests. Keyword-based targeting involved displaying ads based on specific search terms or content-related keywords, while contextual advertising placed ads on websites relevant to the ad content. Despite being widely used, these methods lacked real-time adaptability and relied heavily on static data. Advertisers manually adjusted ad placements and budgets, making the process time-consuming and less efficient. The traditional approach also depended on third-party cookies to track user behavior across websites, but with increasing privacy regulations and restrictions on cookies, its effectiveness has declined. Additionally, traditional systems often resulted in ad fatigue due to repetitive ad exposure, leading to lower engagement rates. Another challenge was that ads were often displayed to broad audiences rather than being tailored to individual preferences, reducing the chances of conversions. Without dynamic learning capabilities, these systems struggled to analyze complex user behavior, leading to wasted impressions and inefficient budget allocation. While traditional advertising provided basic targeting capabilities, it lacked the precision and scalability required in today’s fast-evolving digital landscape. As user behavior became more diverse and unpredictable, traditional systems failed to optimize ad delivery efficiently, resulting in lost opportunities for advertisers. The increasing competition in digital marketing further highlighted the limitations of traditional methods, prompting the need for AI-powered solutions that could adapt, predict, and personalize ad targeting dynamically.

**Limitations of the Traditional System**

* **Lack of Personalization** – Ads were based on broad demographics rather than individual preferences.
* **Static Targeting Methods** – The system relied on predefined rules that didn’t adapt to real-time user behavior.
* **Manual Optimization** – Required constant human intervention, making it time-consuming and inefficient.
* **Low Conversion Rates** – Irrelevant ads led to poor engagement and wasted ad impressions.
* **Ad Fatigue** – Users were repeatedly exposed to the same ads, leading to decreased effectiveness.
* **Dependency on Third-Party Cookies** – With increasing privacy restrictions, tracking user behavior became challenging.
* **Inefficient Budget Allocation** – Advertisers often spent on irrelevant audiences, reducing return on investment (ROI).
* **Limited Data Utilization** – Traditional systems lacked advanced analytics to extract meaningful insights from user data.

**CHAPTER 4**

**PROPOSED SYSTEM**

**Step 1: Ad Click Dataset**

The dataset used contains information about user interactions with online ads. It includes various features like user demographics, browsing behavior, and ad positioning to predict whether a user clicks on an ad. The dataset is loaded using pandas, and initial data exploration is performed to understand its structure.

**Step 2: Data Preprocessing**

Data preprocessing is crucial for improving model performance. The dataset undergoes several cleaning steps: handling missing values by dropping or imputing them, removing duplicate entries, encoding categorical variables using LabelEncoder, and standardizing numerical features using StandardScaler. These steps ensure that the data is clean and suitable for machine learning algorithms.

**Step 3: SMOTE Data Augmentation**

Since real-world datasets often suffer from class imbalance (e.g., more non-clicks than clicks), the SMOTE (Synthetic Minority Over-sampling Technique) technique is used to balance the dataset. It generates synthetic examples of the minority class to improve model performance by reducing bias towards the majority class.

**Step 4: Existing GBC Algorithm**

The Gradient Boosting Classifier (GBC) is trained as the baseline model. It is an ensemble learning method that builds multiple decision trees sequentially, each learning from the errors of the previous one. The trained model is evaluated using accuracy, precision, recall, and F1-score.

**Step 5: Proposed FFNN+RFC Classifier**

A hybrid model combining a Feedforward Neural Network (FFNN) for feature extraction and a Random Forest Classifier (RFC) for classification is proposed. The FFNN learns abstract feature representations, which are then passed to the RFC for final classification. This approach aims to enhance accuracy and robustness compared to traditional models.

**Step 6: Performance Comparison Metrics and Graph**

Both models (GBC and FFNN+RFC) are evaluated based on key metrics: accuracy, precision, recall, and F1-score. A bar chart is plotted to compare their performance, allowing visual analysis of improvements achieved by the proposed model.

**Step 7: Prediction on Test Data using FFNN+RFC**

The trained FFNN+RFC model is used for predicting outcomes on new test data. The test dataset is preprocessed similarly, passed through the FFNN feature extractor, and then classified using RFC. The predicted results help in optimizing ad targeting by identifying potential click patterns.

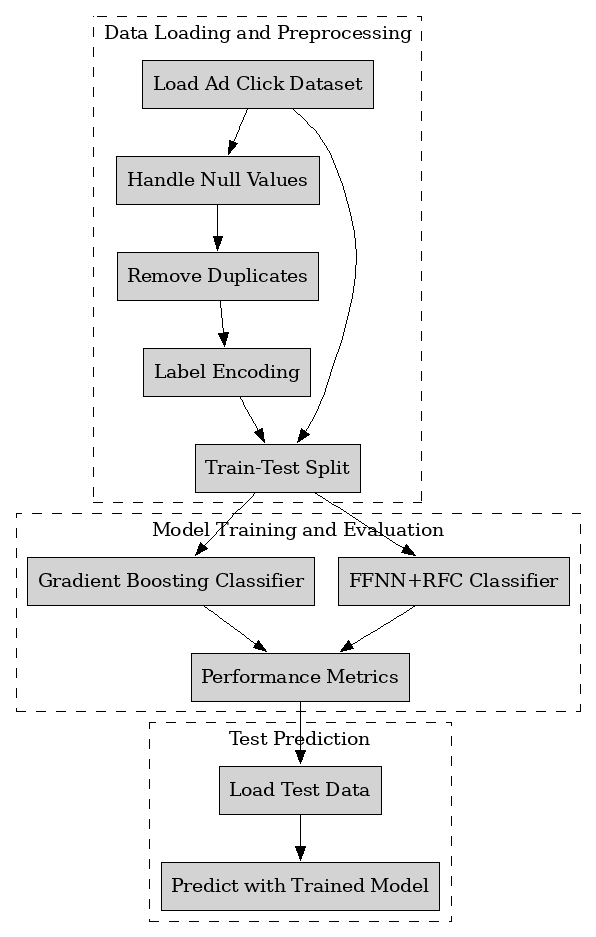


Fig. 1: Architectural Block Diagram of The Proposed System.

**4.2 Data Preprocessing and Data Splitting**

In your research, Data Preprocessing and Data Splitting are critical steps in preparing the dataset for machine learning modeling. These steps are designed to ensure that the data is in the correct format and that it is divided in such a way that the model can be trained and evaluated effectively.

**1. Data Preprocessing**

Data Preprocessing involves transforming the raw data into a format that can be effectively used for machine learning models. In your project, this process consists of several stages:

* **Handling Missing Values (Null Values)**
* **Why it’s needed**: Datasets often have missing values that could lead to inaccurate model predictions if left untreated.
* **How it’s done**: You typically handle missing values by either removing rows/columns containing them or filling them with specific values (like the mean, median, or mode). Depending on the dataset's structure, different strategies are applied.
* **Removing Duplicates**
* **Why it’s needed**: Duplicate records can bias the model by overrepresenting certain data points, leading to overfitting.
* **How it’s done**: You identify and remove duplicate entries using a function (e.g., drop\_duplicates in pandas). This ensures that each data point is unique.
* **Label Encoding**
* **Why it’s needed**: Machine learning models generally work with numerical data, and many datasets contain categorical variables (e.g., labels like "Yes" and "No").
* **How it’s done**: Categorical variables need to be converted into numerical format. **Label Encoding** is a technique that converts categorical labels into numerical values. For example, "Yes" becomes 1 and "No" becomes 0.
* **Feature Scaling (if applicable)**
* **Why it’s needed**: Different features in a dataset may have different scales (e.g., one feature ranges from 1 to 10, and another ranges from 1000 to 10000). If not scaled, certain models may give undue importance to features with larger values.
* **How it’s done**: You scale features (such as using **Standardization** or **Normalization**) to bring all features into the same range, typically between 0 and 1 or to a standard normal distribution.
* **2. Data Splitting**
* Once the dataset is preprocessed, you need to split it into **training** and **test** sets to build and evaluate machine learning models.
* **Why Data Splitting is Needed**
* **Training Set**: This subset is used to train the model. The model learns patterns and relationships in the data based on this set.
* **Test Set**: The test set is used to evaluate the model’s performance on unseen data. This ensures that the model generalizes well and doesn’t overfit the training data.
* **Validation Set (optional)**: Sometimes, a third set, called a validation set, is used during training to tune hyperparameters. However, in your description, you only mention training and test splitting.

**4.3 ML Model Building**

**4.3.1 Existing Algorithm: Gradient Boosting Classifier (GBC)**

**What is GBC Classifier?**

Gradient Boosting Classifier (GBC) is a machine learning algorithm that belongs to the family of ensemble methods. Specifically, it uses the **boosting** technique to combine multiple weak learners (usually decision trees) into a strong classifier. The goal is to create a model that can provide more accurate predictions by sequentially improving on the mistakes made by previous models.

The key principle of GBC is that each new model (or weak learner) tries to correct the errors made by the previous models, focusing on difficult-to-classify instances.

**How it Works:**

* **Initial Model**: GBC starts by training a simple model, often a decision tree, on the training data.
* **Error Calculation**: The algorithm calculates the residual errors (the difference between the predicted and actual values) made by the first model.
* **Model Update**: The next model is trained to predict these residual errors, essentially focusing on the samples that were misclassified by the previous model.
* **Combining Models**: The predictions from all the weak learners are combined in a weighted sum. The models are weighted based on their ability to correct the errors of previous models.
* **Iteration**: Steps 2-4 are repeated iteratively, with each new model improving the predictions of the previous models. The algorithm stops when a predefined number of iterations is reached or when no significant improvement is made.

**Architecture of GBC Classifier:**

The architecture of a Gradient Boosting Classifier can be summarized as follows:

* **Weak Learners (Base Models)**: Typically decision trees.
* **Boosting Process**: Models are trained sequentially with each new model attempting to reduce the errors of previous models.
* **Weighted Combination of Models**: All models contribute to the final prediction, weighted based on their performance in correcting errors.

**4.3.2 Proposed Algorithm: FFNN+RFC Classifier**

**What is FFNN+RFC Classifier?**

FFNN+RFC stands for **Feedforward Neural Network (FFNN)** combined with a **Random Forest Classifier (RFC)**. It is an ensemble approach that utilizes the power of two different machine learning algorithms to improve classification performance.

* **FFNN (Feedforward Neural Network)**: A type of neural network where information flows from the input layer to the output layer in one direction (no cycles). FFNNs are good at learning complex patterns and relationships in data through multiple hidden layers.
* **RFC (Random Forest Classifier)**: An ensemble method that builds multiple decision trees during training and combines their outputs to improve classification accuracy. Each tree in the forest is trained on a random subset of the data and uses a random subset of features for splitting at each node.

**How it Works:**

* **Data Preprocessing**: Initially, the data is preprocessed as described earlier (handling missing values, encoding labels, etc.).
* **Training FFNN**: The FFNN is trained on the dataset to capture the non-linear relationships between features. It uses multiple layers of neurons and activation functions to learn complex patterns.
* **Training RFC**: Simultaneously, the RFC is trained on the same data. Each tree in the random forest learns from random samples and features.
* **Model Combination**: After both models are trained, their predictions are combined. Typically, the output of the two models is averaged or voted on to provide a final classification.
  + **Averaging**: For regression tasks, the final output is the average of the predictions of both models.
  + **Voting**: For classification tasks, the most frequent prediction from FFNN and RFC becomes the final prediction.
* **Evaluation**: The combined model is evaluated on a test set to assess its performance.
* **Architecture of FFNN+RFC Classifier:**
* The architecture of FFNN+RFC consists of two main components:
* **FFNN (Feedforward Neural Network)**:
  + **Input Layer**: Accepts the features.
  + **Hidden Layers**: Consists of multiple neurons that process data.
  + **Output Layer**: Produces the output (classification or regression prediction).
* **RFC (Random Forest Classifier)**:
  + **Multiple Decision Trees**: Trained on random subsets of data.
  + **Voting/Averaging**: Combines the predictions of the individual trees.

**Advantages of FFNN+RFC Classifier:**

* **Complementary Strengths**: FFNNs are powerful at capturing complex relationships and patterns in the data, while RFCs are strong at handling feature interactions and reducing overfitting by averaging the results of multiple decision trees.
* **Improved Generalization**: The combination of FFNN and RFC helps the model generalize better than using either method alone.
* **Handling Non-Linearity and Feature Interactions**: FFNN captures non-linear relationships, while RFC handles interactions between features effectively.
* **Robustness to Overfitting**: By using both a neural network and an ensemble of trees, this model is less prone to overfitting compared to a single model.
* **Versatility**: This hybrid model can be applied to various tasks, including both classification and regression.

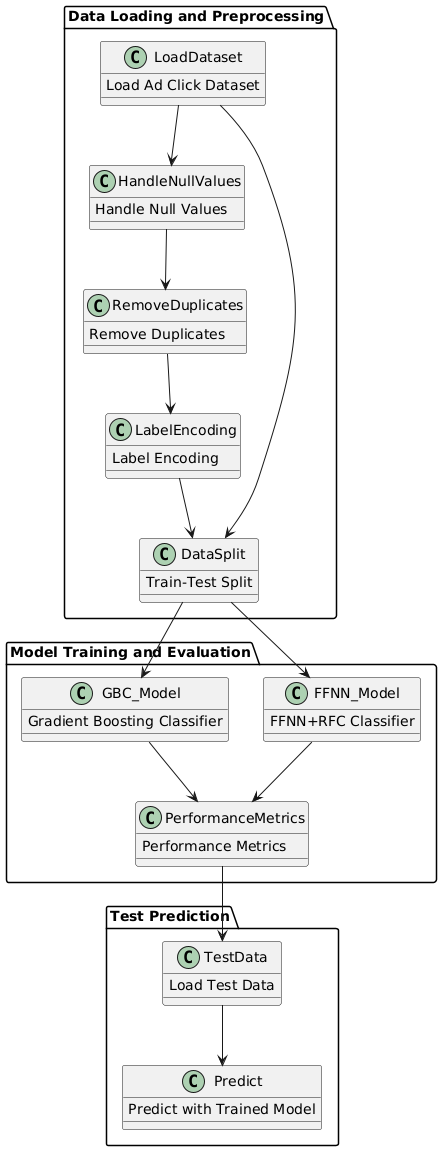
**CHAPTER 5**

**UML DIAGRAM**

**Class Diagram**

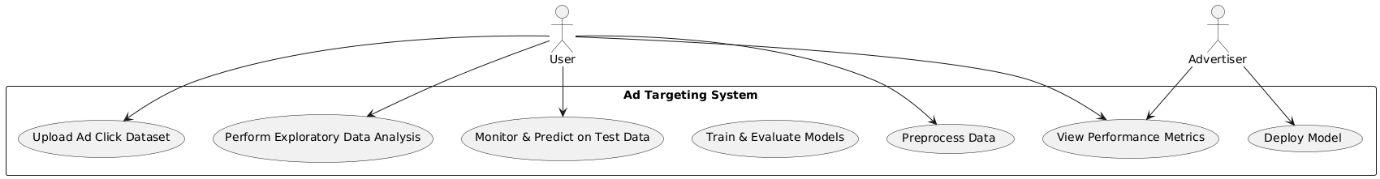
The class diagram is the main building block of object-oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts:

* The upper part holds the name of the class
* The middle part contains the attributes of the class
* The bottom part gives the methods or operations the class can take or undertake.

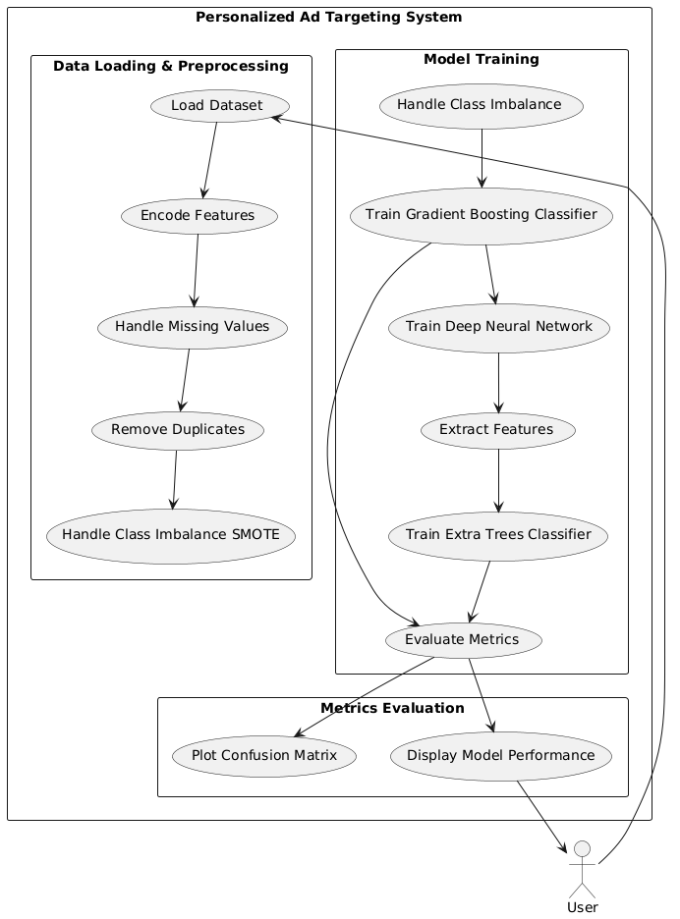


**Use case Diagram**

A use case diagram at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

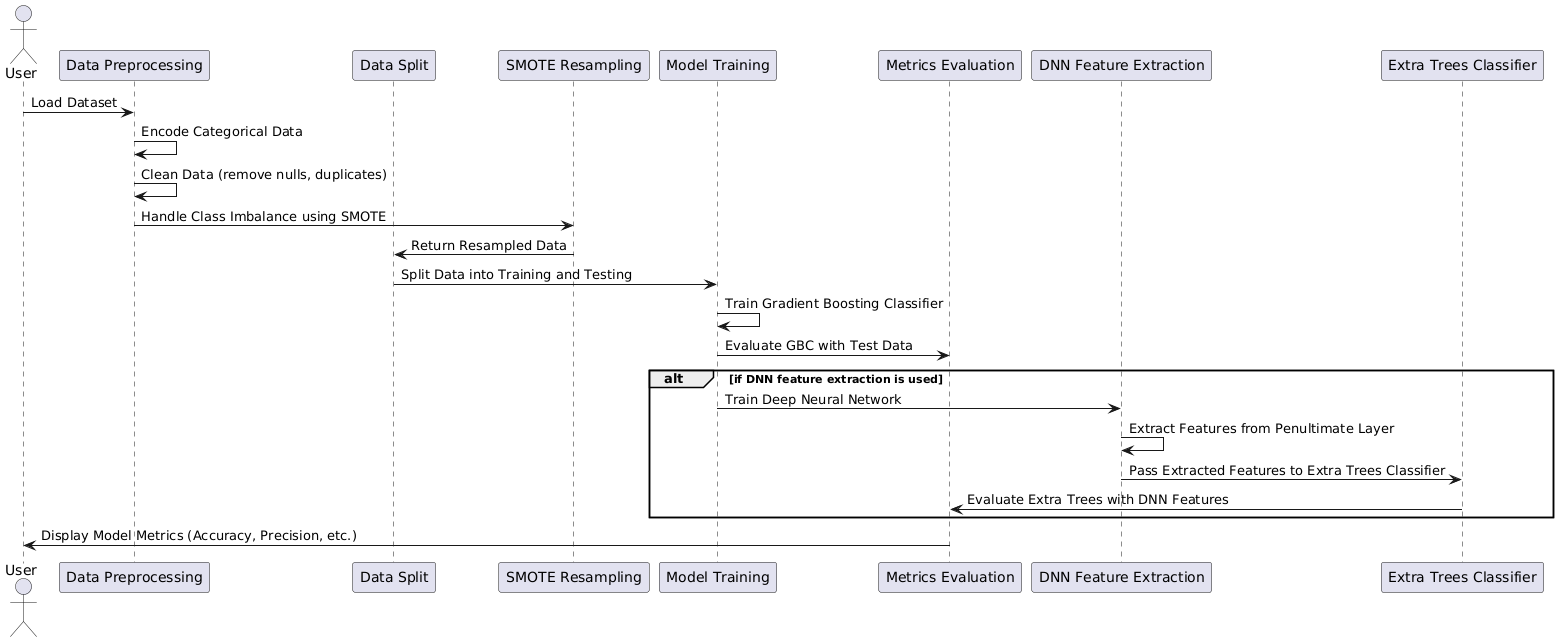


**Data Flow Diagram**



**Sequence Diagram**

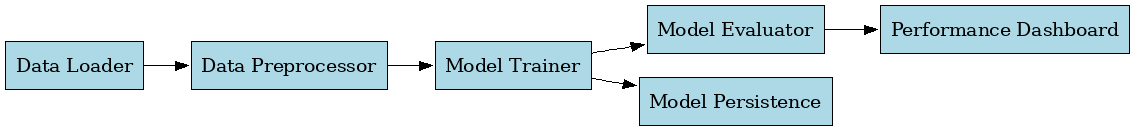
A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



**Component Diagram**

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.

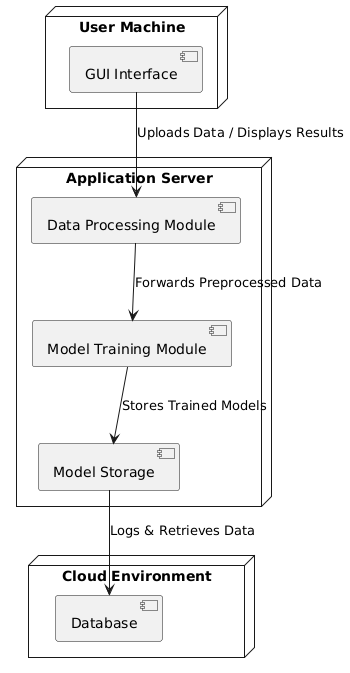
Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.



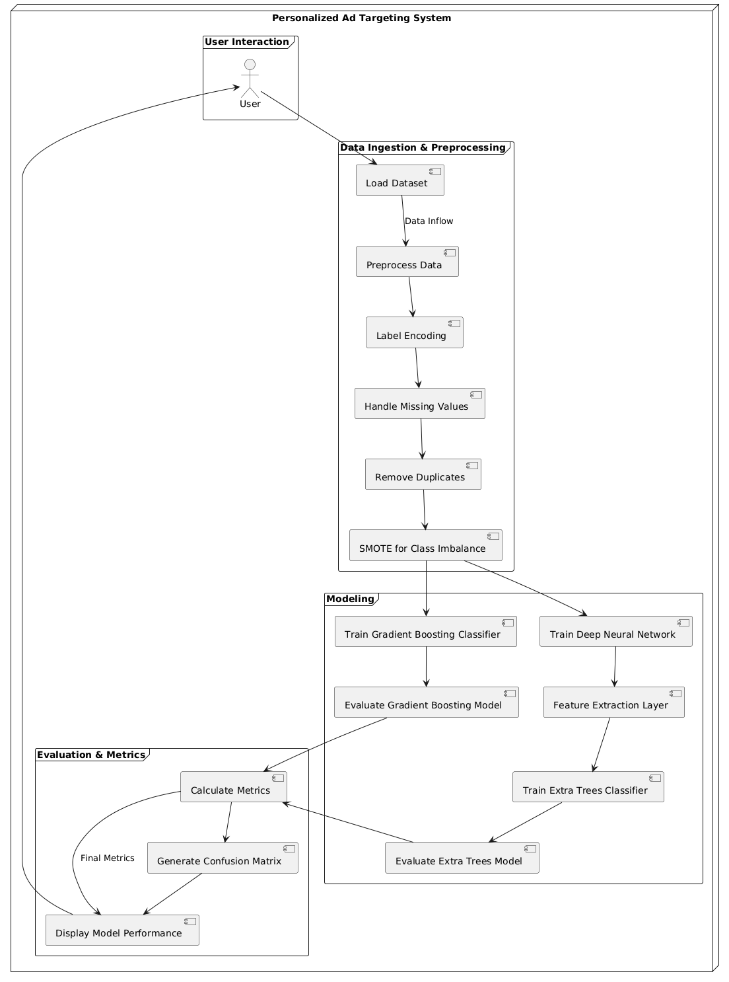
**Deployment Diagram**

A deployment diagram in the Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.



**Activity Diagram**



**CHAPTER 6**

**SOFTWARE ENVIRONMENT**

**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 library 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

**Advantages of Python**

Let’s see how Python dominates over other languages.

**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. 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.

**9. Free and Open-Source**

Like we 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, we 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**

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.

**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.

This was all about the Advantages and Disadvantages of Python Programming Language.

**Modules Used in Project**

**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. It contains various features including these important ones:

* A powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* 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.

**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.

**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.

**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

**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

A screenshot of a computer

Description automatically generated with medium confidence

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

Step 2: Click on the Download Tab.

Graphical user interface, application

Description automatically generated

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

Graphical user interface, application

Description automatically generated

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.

Graphical user interface, text

Description automatically generated

* 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.

Graphical user interface, text, application

Description automatically generated

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

Graphical user interface, text, application, chat or text message

Description automatically generated

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

Graphical user interface, text, application, chat or text message

Description automatically generated

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”.

Graphical user interface, application

Description automatically generated

Step 3: Open the Command prompt option.

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

A screenshot of a computer

Description automatically generated with medium confidence

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”.

Application

Description automatically generated with low confidence

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

Graphical user interface, text, application, email

Description automatically generated

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.

Graphical user interface, text, application, email

Description automatically generated

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won’t work.

**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 colab

**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 recording
* Data transcription
* Data conversion
* Data verification
* Data control
* Data transmission
* Data validation
* Data correction

**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 inputs, which are user communications with the system.
* Operational, which are computer department’s communications to the system?
* Interactive, which are inputs entered during a dialogue.

**Input Media**

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

* Type of input
* Flexibility of format
* Speed
* Accuracy
* Verification methods
* Rejection rates
* Ease of correction
* Storage and handling requirements
* Security
* Easy to use
* Portability

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 Clasified 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
* The existing system is completely dependent on the user to perform all the duties.

**CHAPTER 9**

**SOURCE CODE**

from tkinter import \*

import tkinter

from tkinter import filedialog

from tkinter.filedialog import askopenfilename

from tkinter import simpledialog

import numpy as np

import pandas as pd

import seaborn as sns

from scipy import stats

import matplotlib.pyplot as plt

from imblearn.over\_sampling import SMOTE

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.linear\_model import LinearRegression, Lasso, RidgeClassifier, ElasticNet

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

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

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, f1\_score, precision\_score, recall\_score

import os, joblib

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.models import load\_model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.utils import to\_categorical

global filename

global classifier

global X, y, X\_train, X\_train, y\_train, y\_test ,Predictions

global dataset, df, df2, sc, train\_or\_load\_dnn, dnn\_model

global le, labels

def upload():

global filename

global dataset, df

filename = filedialog.askopenfilename(initialdir = "Datasets")

text.delete('1.0', END)

text.insert(END,filename+' Loaded\n\n')

df= pd.read\_csv(filename, encoding='latin1')

text.insert(END,'\n\n IoT Network Dataset: \n', str(df))

text.insert(END,df)

def preprocess():

global dataset, df, df2

global X, y, X\_train, X\_test, y\_train, y\_test, scaler, le, labels

text.delete('1.0', END)

# Display basic information about the dataset

#text.insert(END, '\n\nInformation of the dataset: \n', str(df.info()))

print(df.info())

text.insert(END, '\n\nDescription of the dataset: \n' + str(df.describe().T))

text.insert(END, '\n\nChecking null values in the dataset: \n' + str(df.isnull().sum()))

text.insert(END, '\n\nUnique values in the dataset: \n' + str(df.nunique()))

le=LabelEncoder()

df['full\_name']=le.fit\_transform(df['full\_name'])

df['gender']=le.fit\_transform(df['gender'])

df['device\_type']=le.fit\_transform(df['device\_type'])

df['ad\_position']=le.fit\_transform(df['ad\_position'])

df['browsing\_history']=le.fit\_transform(df['browsing\_history'])

df['time\_of\_day']=le.fit\_transform(df['time\_of\_day'])

df = df.dropna()

df.duplicated().sum()

df = df.drop\_duplicates()

labels=['0','1']

# Create a count plot

sns.set(style="darkgrid") # Set the style of the plot

plt.figure(figsize=(8, 6)) # Set the figure size

ax = sns.countplot(data=df, x='click')

plt.title("Count Plot") # Add a title to the plot

plt.xlabel("Categories") # Add label to x-axis

plt.ylabel("Count") # Add label to y-axis

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')

print(df['click'].unique())

y = df['click'].values

X = df.drop(columns=['click'], axis=1).values

print(df['click'].unique())

smote = SMOTE(random\_state=30)

X\_res, y\_res = smote.fit\_resample(X, y)

# Create a count plot

sns.set(style="darkgrid") # Set the style of the plot

plt.figure(figsize=(8, 6)) # Set the figure size

ax = sns.countplot(x=y\_res, data=df)

plt.title("Count Plot") # Add a title to the plot

plt.xlabel("Categories") # Add label to x-axis

plt.ylabel("Count") # Add label to y-axis

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')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_res, y\_res, test\_size=0.2, random\_state=45)

text.insert(END, "\n\nTotal Records used for training: " + str(len(X\_train)) + "\n")

text.insert(END, "\n\nTotal Records used for testing: " + str(len(X\_test)) + "\n\n")

from sklearn.preprocessing import StandardScaler

# Initialize the scaler

scaler = StandardScaler()

# Fit and transform the training data

#X\_train = scaler.fit\_transform(X\_train)

#X\_test = scaler.transform(X\_test)

# Correlation heatmap

plt.figure(figsize=(14, 14))

sns.set(font\_scale=1)

sns.heatmap(df.corr(), cmap='GnBu\_r', annot=True, square=True, linewidths=.5)

plt.title('Variable Correlation in Heatmap')

#plt.show()

precision = []

recall = []

fscore = []

accuracy = []

#function to calculate various metrics such as accuracy, precision etc

def calculateMetrics(algorithm, testY,predict):

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+' F1-SCORE : '+str(f))

text.insert(END, "Performance Metrics of " + str(algorithm) + "\n")

text.insert(END, "Accuracy: " + str(a) + "\n")

text.insert(END, "Precision: " + str(p) + "\n")

text.insert(END, "Recall: " + str(r) + "\n")

text.insert(END, "F1-SCORE: " + str(f) + "\n\n")

report=classification\_report(predict, testY,target\_names=labels)

print('\n',algorithm+" classification report\n",report)

text.insert(END, "classification report: \n" + str(report) + "\n\n")

conf\_matrix = confusion\_matrix(testY, predict)

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 GBCRegressor():

global ridge\_clf, X\_train, X\_test, y\_train, y\_test

global predict

Classifier = 'model/GradientBoostingClassifier.pkl'

if os.path.exists(Classifier):

# Load the trained model from the file

ridge\_clf = joblib.load(Classifier)

print("Model loaded successfully.")

predict = ridge\_clf.predict(X\_test)

calculateMetrics("Gradient Boosting Classifier", predict, y\_test)

else:

# Initialize and train the Ridge Classifier model

ridge\_clf = GradientBoostingClassifier()

ridge\_clf.fit(X\_train, y\_train)

# Save the trained model to a file

joblib.dump(ridge\_clf, Classifier)

print("Model saved successfully.")

predict = ridge\_clf.predict(X\_test)

calculateMetrics("Gradient Boosting Classifier", predict, y\_test)

def DTC():

global dtc\_clf, X\_train, X\_train, y\_train, y\_test

global predict

Classifier = 'model/DecisionTreeClassifier.pkl'

if os.path.exists(Classifier):

# Load the trained model from the file

dtc\_clf = joblib.load(Classifier)

print("Model loaded successfully.")

predict = dtc\_clf.predict(X\_test)

calculateMetrics("Decision Tree Classifier", predict, y\_test)

else:

# Initialize and train the Ridge Classifier model

dtc\_clf = DecisionTreeClassifier()

dtc\_clf.fit(X\_train, y\_train)

# Save the trained model to a file

joblib.dump(dtc\_clf, Classifier)

print("Model saved successfully.")

predict = dtc\_clf.predict(X\_test)

calculateMetrics("Decision Tree Classifier", predict, y\_test)

def FFNN():

global X\_train, X\_test, y\_train, y\_test, model, rfc, extractor

import os

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import Dense

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.metrics import accuracy\_score

import joblib

print('X\_train:', X\_train.shape)

print('X\_test:', X\_test.shape)

print('y\_train:', y\_train.shape)

print('y\_test:', y\_test.shape)

model\_folder = "model"

dnn\_model\_path = os.path.join(model\_folder, "dnn\_feature\_extractor.h5")

rfc\_model\_path = os.path.join(model\_folder, "rfc\_model.pkl")

os.makedirs(model\_folder, exist\_ok=True)

if os.path.exists(dnn\_model\_path) and os.path.exists(rfc\_model\_path):

print("Loading saved models...")

extractor = load\_model(dnn\_model\_path)

rfc = joblib.load(rfc\_model\_path)

X\_train\_features = extractor.predict(X\_train)

X\_test\_features = extractor.predict(X\_test)

y\_pred = rfc.predict(X\_test\_features)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Random Forest Classifier Accuracy: {accuracy \* 100:.2f}%")

text.insert(tkinter.END, '\n\n---------FFNN Model---------\n\n')

calculateMetrics("FFNN Model", y\_test, y\_pred)

else:

print("Training models from scratch...")

model = Sequential()

model.add(Dense(64, activation='relu', input\_shape=(8,))) # Adjust input shape as per your data

model.add(Dense(32, activation='relu')) # Feature extraction layer

model.add(Dense(1, activation='softmax')) # Output layer

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=50, verbose=0)

extractor = Sequential(model.layers[:-1]) # Remove the output layer

extractor.save(dnn\_model\_path)

X\_train\_features = extractor.predict(X\_train)

X\_test\_features = extractor.predict(X\_test)

rfc = ExtraTreesClassifier()

rfc.fit(X\_train\_features, y\_train)

joblib.dump(rfc, rfc\_model\_path)

y\_pred = rfc.predict(X\_test\_features)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Random Forest Classifier Accuracy: {accuracy \* 100:.2f}%")

text.insert(tkinter.END, '\n\n---------FFNN Model---------\n\n')

calculateMetrics("FFNN Model", y\_test, y\_pred)

def predict():

global scaler, rfc, model, labels, extractor

file = filedialog.askopenfilename(initialdir="Datasets")

test = pd.read\_csv(file)

# Display loaded test data

text.delete('1.0', END)

text.insert(END, f'{file} Loaded\n')

text.insert(END, "\n\nLoaded test data: \n" + str(test) + "\n")

"""le=LabelEncoder()

test['full\_name']=le.fit\_transform(test['full\_name'])

test['gender']=le.fit\_transform(test['gender'])

test['device\_type']=le.fit\_transform(test['device\_type'])

test['ad\_position']=le.fit\_transform(test['ad\_position'])

test['browsing\_history']=le.fit\_transform(test['browsing\_history'])

test['time\_of\_day']=le.fit\_transform(test['time\_of\_day'])"""

if 'click' in test.columns:

test = test.drop(['click'], axis=1)

test\_values = test.values

#test\_scaled = scaler.transform(test\_values)

FFNN\_predictions = extractor.predict(test\_values)

#predicted\_classes = FFNN\_predictions.argmax(axis=1)

predicted\_classes = rfc.predict(FFNN\_predictions)

predicted\_labels = [labels[p] for p in predicted\_classes]

test['Predicted'] = predicted\_labels

text.insert(END, "\n\nModel Predicted value in test data: \n" + str(test) + "\n")

def graph():

columns = ["Algorithm Name", "Accuracy", "Precision", "Recall", "f1-score"]

algorithm\_names = ["DTC Classification", "FFNN+RF Classification"]

# 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()

# Main window setup

main = Tk()

main.title("Personalized Ad Targeting System using Machine Learning for Optimizing Ad Delivery")

main.geometry("1200x800") # Spacious window size

main.config(bg='#2B3A67') # Navy Blue background for a sleek look

# Title Label with a gradient-like dark-to-light theme

font = ('Verdana', 20, 'bold')

title = Label(main, text='Personalized Ad Targeting System using Machine Learning for Optimizing Ad Delivery',

bg='#282828', fg='#FFD700', font=font, height=2) # Dark background with Gold text

title.pack(fill=X, pady=10)

# Frame to hold buttons and text console

main\_frame = Frame(main, bg='#2B3A67') # Navy Blue for consistency

main\_frame.pack(fill=BOTH, expand=True, padx=20, pady=20)

# Frame to hold buttons (centered and in two rows)

button\_frame = Frame(main\_frame, bg='#2B3A67')

button\_frame.pack(pady=20)

# Button Font and Style

font1 = ('Arial', 12, 'bold')

# Helper function to create buttons with fancy color tones

def create\_button(text, command, row, column):

btn = Button(button\_frame, text=text, command=command, bg='#1E90FF', fg='white', # Dodger Blue buttons

activebackground='#FFA07A', font=font1, width=25, relief=RAISED, bd=4) # Light Salmon hover effect

btn.grid(row=row, column=column, padx=20, pady=15)

# Adding buttons in two rows, three buttons per row

create\_button("Upload Ad Click Dataset", upload, 0, 0)

create\_button("Data Preprocessing and EDA", preprocess, 0, 1)

create\_button("GBC Classifier", GBCRegressor, 0, 2)

create\_button("FFNN+RF Classifier", FFNN, 1, 0)

create\_button("Performance Metrics Graph", graph, 1, 1)

create\_button("Prediction on Test Data", predict, 1, 2)

# Text console styling with scrollbar in fancy tones

text\_frame = Frame(main\_frame, bg='#2B3A67') # Consistent background

text\_frame.pack(fill=BOTH, expand=True, padx=20, pady=20)

# Text box styling with a centered and modern look

text = Text(text\_frame, height=15, width=90, wrap=WORD, bg='#F5DEB3', fg='#483D8B', font=('Comic Sans MS', 14)) # Wheat background

scroll = Scrollbar(text\_frame, command=text.yview)

text.configure(yscrollcommand=scroll.set)

text.pack(side=LEFT, fill=BOTH, expand=True)

scroll.pack(side=RIGHT, fill=Y)

# Adding the Close Application button with consistent style and size

close\_button = Button(button\_frame, text="Close Application", command=close, bg='#B22222', fg='white', # Firebrick button

activebackground='#FF6347', font=font1, width=25, relief=RAISED, bd=4)

# Placing the Close button in the second row, third column (consistent layout)

close\_button.grid(row=1, column=3, padx=20, pady=15)

main.mainloop()

**CHAPTER 10**

**RESULTS AND DISCUSSION**

**10.1 Implementation Description**

The research involves building a classification model using machine learning algorithms. Below are the detailed steps followed in the implementation:

**Step 1: Data Collection**

The dataset used for the research is collected from reliable sources. The data is gathered in the form of structured tables, with each row representing an instance and each column representing a feature or label. The data includes numerical and categorical features that describe the problem at hand.

**Step 2: Data Preprocessing**

Data preprocessing is performed to ensure that the dataset is clean, well-structured, and suitable for model training.

1. **Handling Missing Values**: Any missing or null values in the dataset are identified and replaced. Numerical features are filled with the mean or median values, while categorical features are filled with the most frequent value or mode.
2. **Encoding Categorical Features**: Categorical variables are encoded using techniques like one-hot encoding or label encoding. This ensures that the features are in a format that machine learning models can process.
3. **Feature Scaling**: Continuous features are normalized or standardized to ensure all features have the same scale. This helps improve the performance and stability of the model, particularly for algorithms sensitive to feature scaling, like neural networks.
4. **Splitting the Dataset**: The dataset is divided into training and testing sets, typically using a 70:30 or 80:20 split. This division ensures that the model is trained on one portion of the data and tested on another, helping evaluate its performance.

**Step 3: Model Selection**

Two machine learning algorithms are selected for the project: **Gradient Boosting Classifier (GBC)** and a hybrid model combining **Feedforward Neural Network (FFNN)** with **Random Forest Classifier (RFC)**.

* **GBC** is chosen as the existing model due to its strong performance in classification tasks.
* The **FFNN+RFC** hybrid model is proposed to improve classification accuracy by combining the strengths of both a neural network and an ensemble learning method.

**Step 4: Model Training**

1. **Gradient Boosting Classifier (GBC)**:
   * The GBC model is trained using the training data. It learns to predict the target variable by iteratively improving the accuracy of previous weak learners.
   * The training process involves creating decision trees that focus on the residual errors of the previous trees.
2. **FFNN+RFC Hybrid Model**:
   * The FFNN is trained using the training data. The network learns complex non-linear relationships between the features through multiple hidden layers.
   * Simultaneously, the RFC is trained using random subsets of data and features. Each tree in the forest learns to classify the data independently.
   * The predictions of both models are combined using a voting or averaging technique to produce the final output.

**Step 5: Model Evaluation**

After training the models, their performance is evaluated using the test data. The models are assessed based on key evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrix. The goal is to compare the performance of the existing GBC classifier with the proposed FFNN+RFC hybrid model.

1. **Accuracy**: The percentage of correct predictions made by the model.
2. **Precision and Recall**: These metrics are calculated for each class, giving insight into how well the models perform in terms of both false positives and false negatives.
3. **F1-Score**: The harmonic mean of precision and recall, offering a balanced view of the model’s performance.
4. **Confusion Matrix**: A matrix that visualizes the classification performance, showing true positives, false positives, true negatives, and false negatives.

**Step 6: Model Tuning**

Model hyperparameters are tuned to optimize the performance of both models.

* For **GBC**, hyperparameters such as the number of estimators, learning rate, and max depth of trees are tuned to prevent overfitting and ensure better performance.
* For the **FFNN+RFC** model, hyperparameters of both the neural network and random forest are adjusted, including the number of layers, number of neurons, tree depth, and the number of trees.

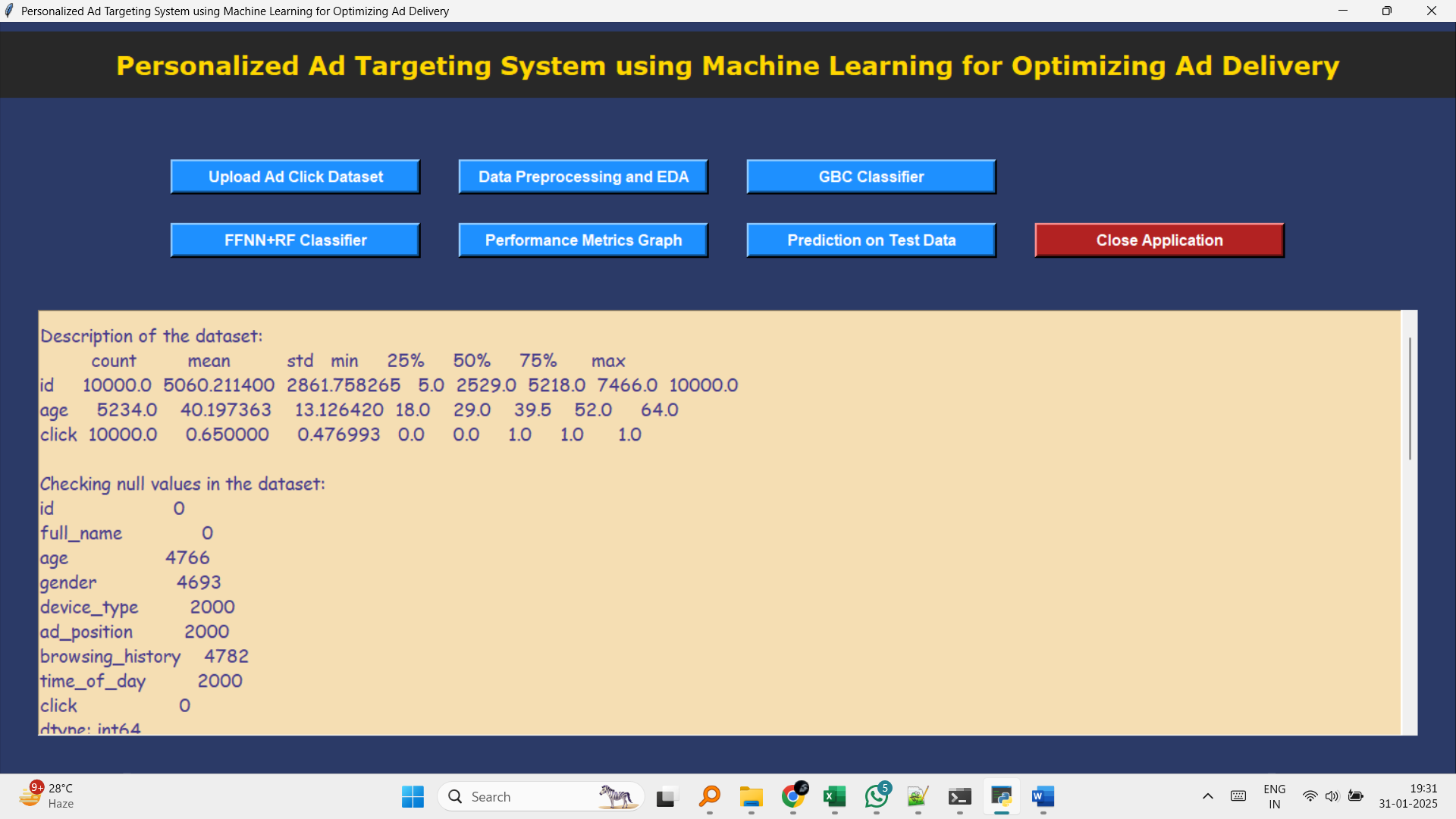
**10.3 Dataset Description**

The dataset consists of several features and a target variable. The features describe different aspects of users and their interactions with an online platform, while the target variable indicates the outcome of interest. Here's a description of each feature and the target variable:

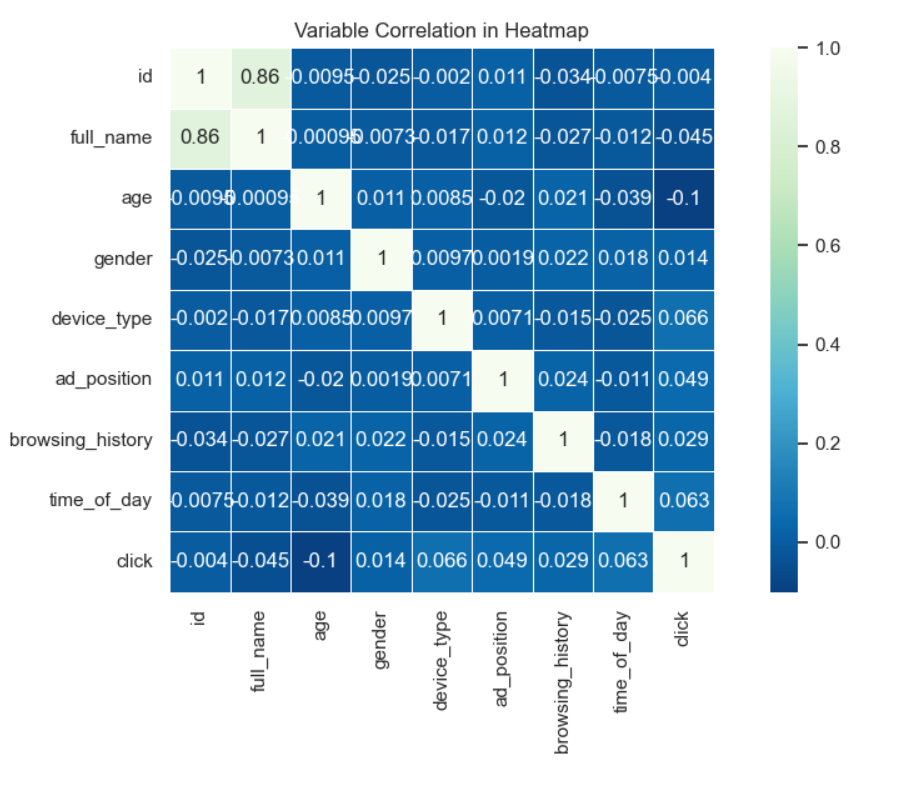
* **id**: This is a unique identifier for each record in the dataset. It is an integer value that helps distinguish one instance from another. It does not contain any meaningful information for model training and is typically excluded from model inputs.
* **full\_name**: This is a text field that contains the full name of the user. It is not used directly for prediction but may be used in preprocessing steps to clean or analyze the data. It provides a unique identification of users.
* **age**: This is a numerical feature representing the age of the user. It gives insight into the demographic profile of the user. Age could influence user behavior and preferences.
* **gender**: This is a categorical feature that indicates the gender of the user (e.g., male, female, other). Gender could have an impact on how users interact with the platform or their likelihood to click on certain ads.
* **device\_type**: This categorical feature represents the type of device the user is using to interact with the platform (e.g., smartphone, tablet, desktop). The device type could be important in understanding user behavior and the effectiveness of ads across different platforms.
* **ad\_position**: This feature indicates the position of the ad that was shown to the user (e.g., top, middle, bottom). It provides valuable information about where users encounter the ad, which might affect the likelihood of them clicking on it.
* **browsing\_history**: This is a categorical or numerical feature that captures the browsing activity of the user. It could include data such as websites visited, search history, or the time spent on specific pages. Browsing history helps understand the interests and preferences of the user.
* **time\_of\_day**: This feature records the time when the user interacts with the platform, such as morning, afternoon, or evening. Time of day can be important in predicting user behavior, as certain times may be associated with higher engagement.
* **click**: This is the target variable (also known as the label) in the dataset. It is a binary classification label indicating whether the user clicked on the displayed ad (1) or not (0). The model's objective is to predict this outcome based on the other features.

**10.2 Results Description**

Figure 1 displays the process of uploading the ad click dataset through the graphical user interface (GUI). The interface allows users to easily import the dataset, which consists of user-specific features such as age, gender, device type, browsing history, and ad click behavior. Upon uploading, the dataset is visually represented in the GUI for analysis. Users can view a snapshot of the data, check for missing values, and perform preliminary checks for anomalies, which helps in understanding the structure of the dataset and deciding how to proceed with further preprocessing steps.

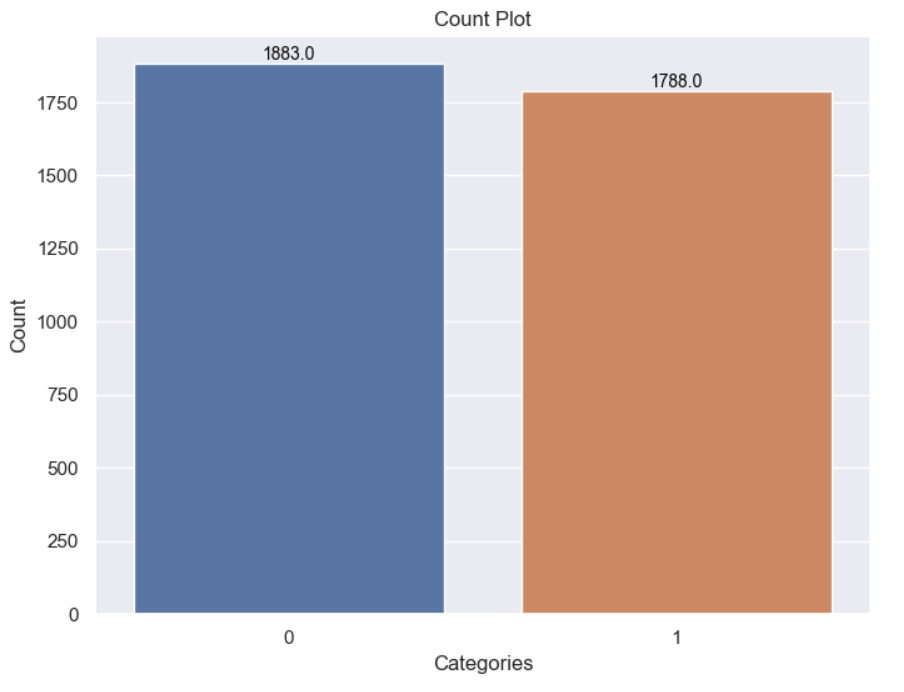
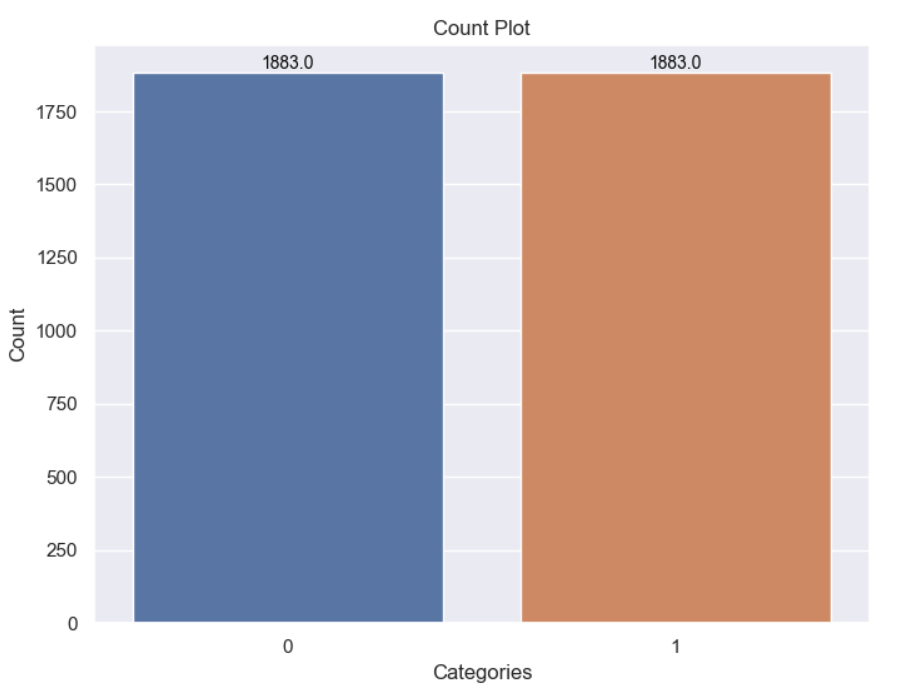


**Figure 1: Upload of Ad Click Dataset and Analysis in the GUI Interface**



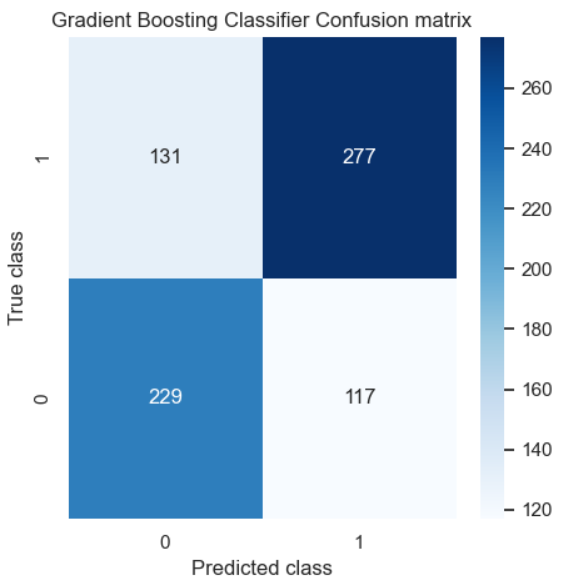
**Figure 2: Exploratory Data Analysis (EDA) of the Dataset**

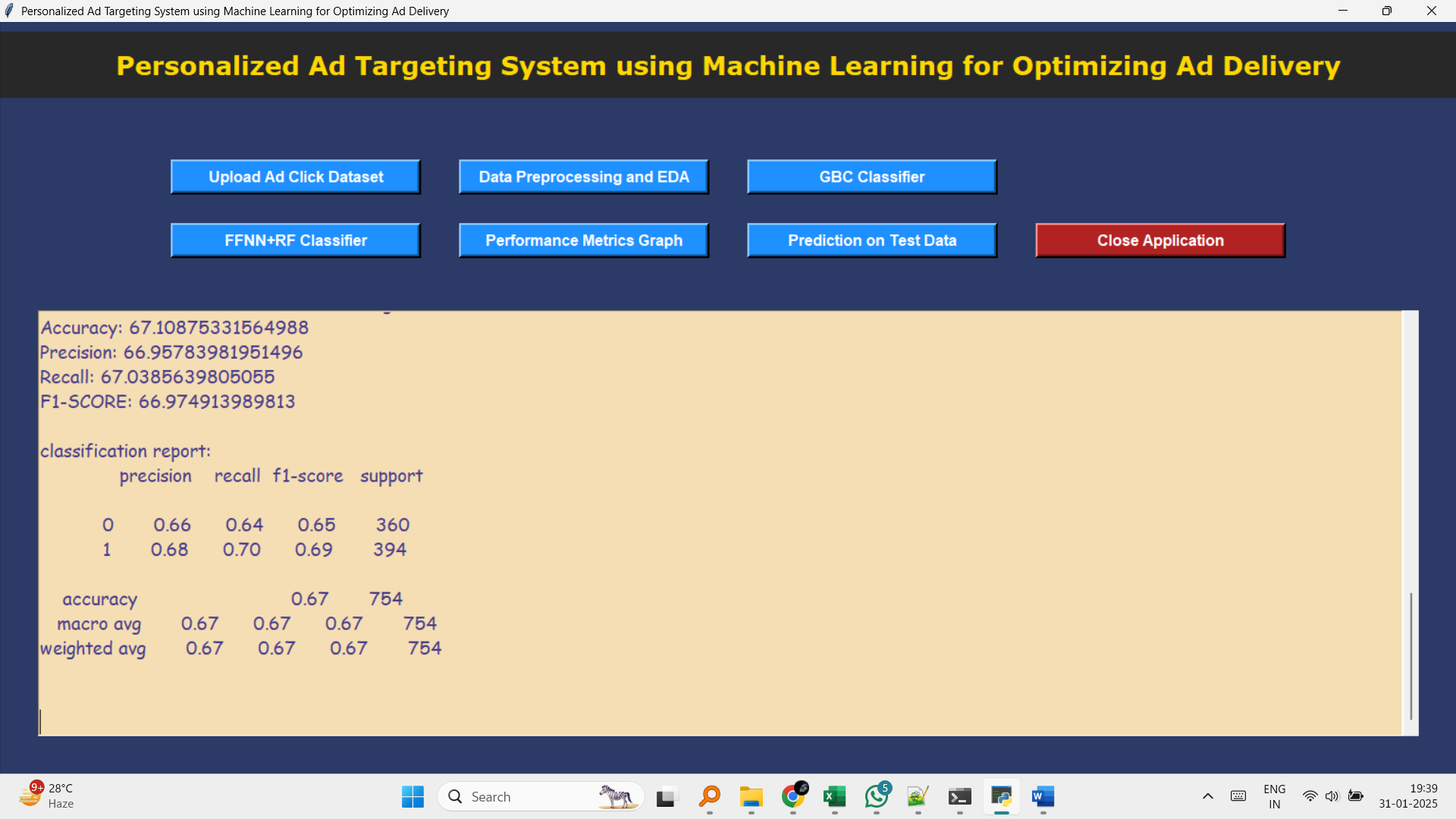
Figure 2 presents the exploratory data analysis (EDA) of the dataset. In this step, various data visualizations and statistical methods are used to gain insights into the distribution of features, correlations between variables, and the overall structure of the dataset. Histograms, bar charts, and heatmaps may be displayed to showcase the distribution of features like age, gender, and device type. The correlation matrix reveals how different features are related to the target variable (ad click or no click) and one another. EDA helps in identifying trends and patterns that inform data preprocessing and model selection.



**Figure 3: Data Preprocessing in the GUI**

Figure 3 shows the data preprocessing steps within the GUI. In this stage, the dataset undergoes cleaning and transformation to prepare it for model training. Missing values are handled, categorical features are encoded, and scaling or normalization techniques are applied to numerical features. The preprocessing step also includes data splitting, where the dataset is divided into training and test sets, ensuring that the model is evaluated on unseen data. This process is essential for building a robust machine learning model that generalizes well to new data.

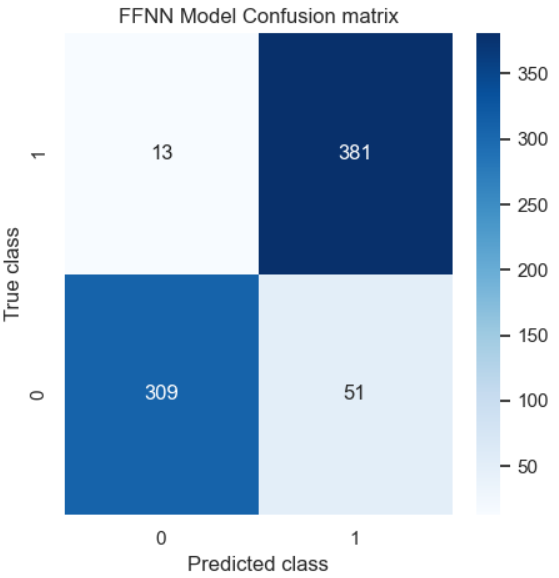


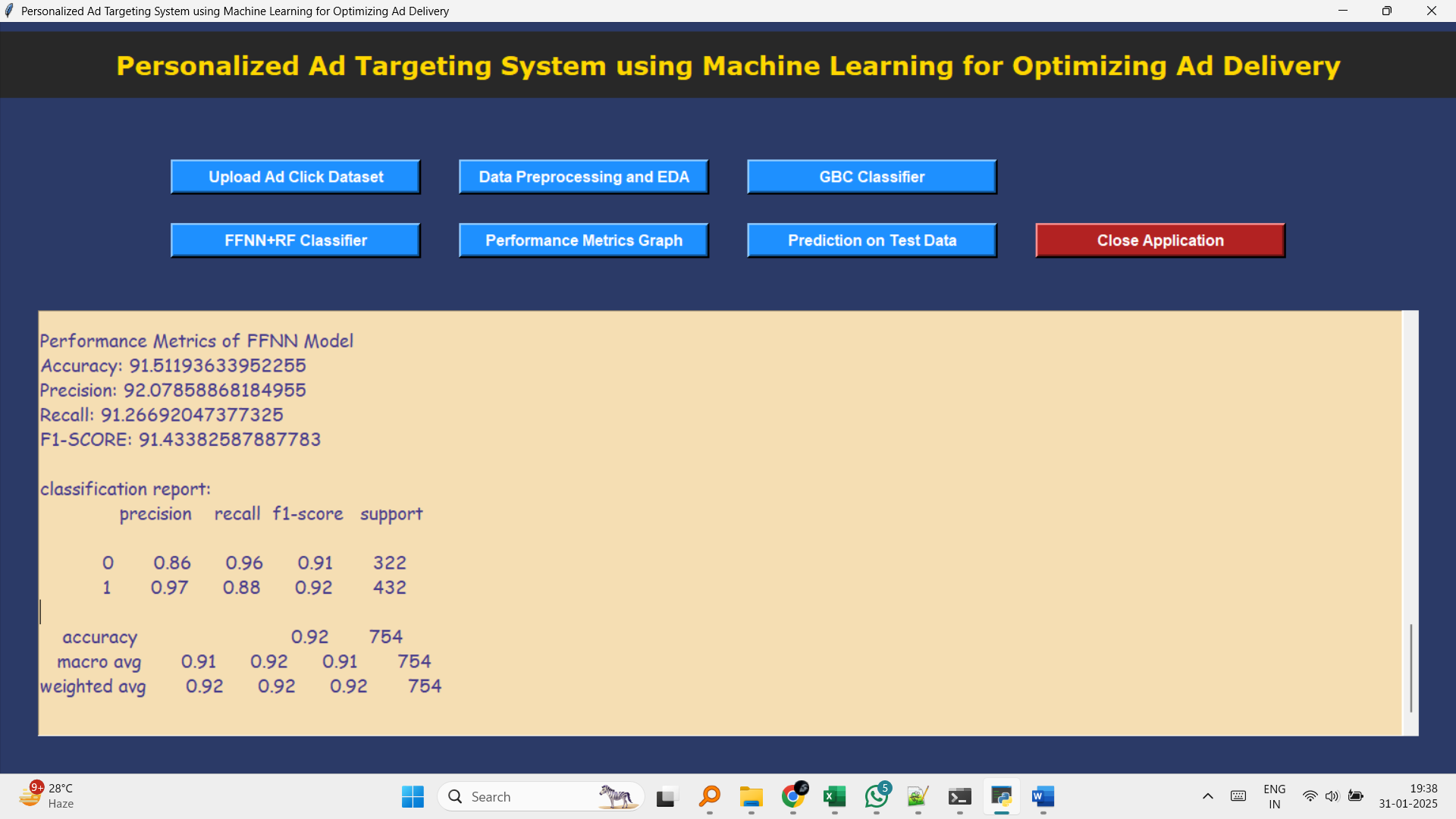


**Figure 4: Performance Metrics and Confusion Matrix Plot of GBC Classifier Model**

Figure 4 displays the performance metrics and confusion matrix plot of the Gradient Boosting Classifier (GBC) model. The GBC model is evaluated using accuracy, precision, recall, and F1-score:

* **Accuracy**: 67.11% – The model correctly predicts the ad click status for about two-thirds of the test data.
* **Precision**: 66.96% – The proportion of correctly predicted positive (clicked) ads out of all ads predicted as clicked.
* **Recall**: 67.04% – The proportion of correctly predicted positive (clicked) ads out of all actual clicked ads in the dataset.
* **F1-Score**: 66.97% – The harmonic mean of precision and recall, providing a balanced measure of the classifier’s performance. The confusion matrix visualizes the number of true positives, true negatives, false positives, and false negatives, helping assess where the model is making errors. The GBC classifier performs well but has room for improvement in predicting both clicked and non-clicked ads more accurately.



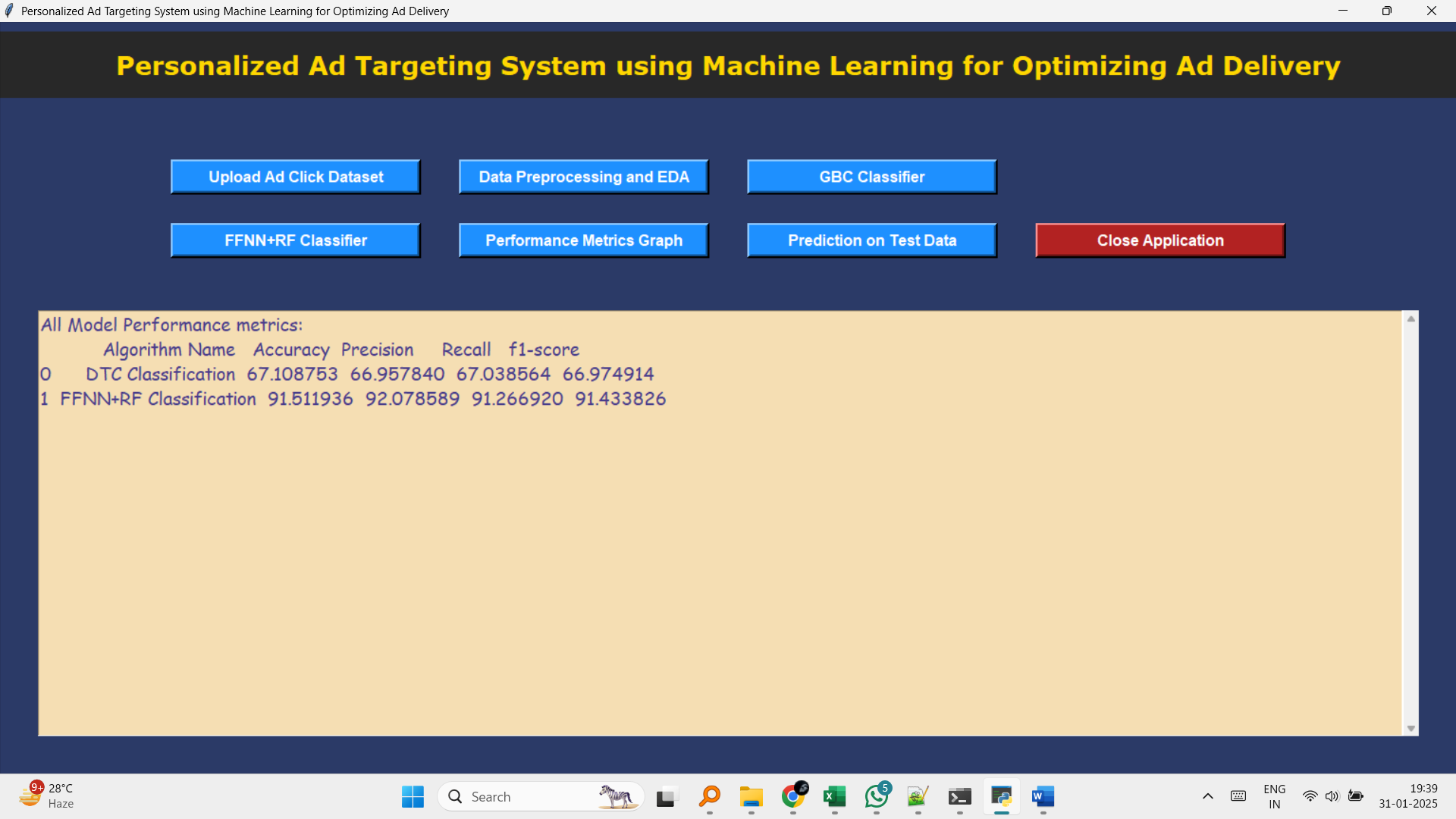


**Figure 5: Performance Metrics and Confusion Matrix Plot of FFNN+RFC Classifier Model**

Figure 5 presents the performance metrics and confusion matrix plot of the Feedforward Neural Network (FFNN) combined with the Random Forest Classifier (RFC) model. This hybrid approach demonstrates improved performance compared to the GBC classifier:

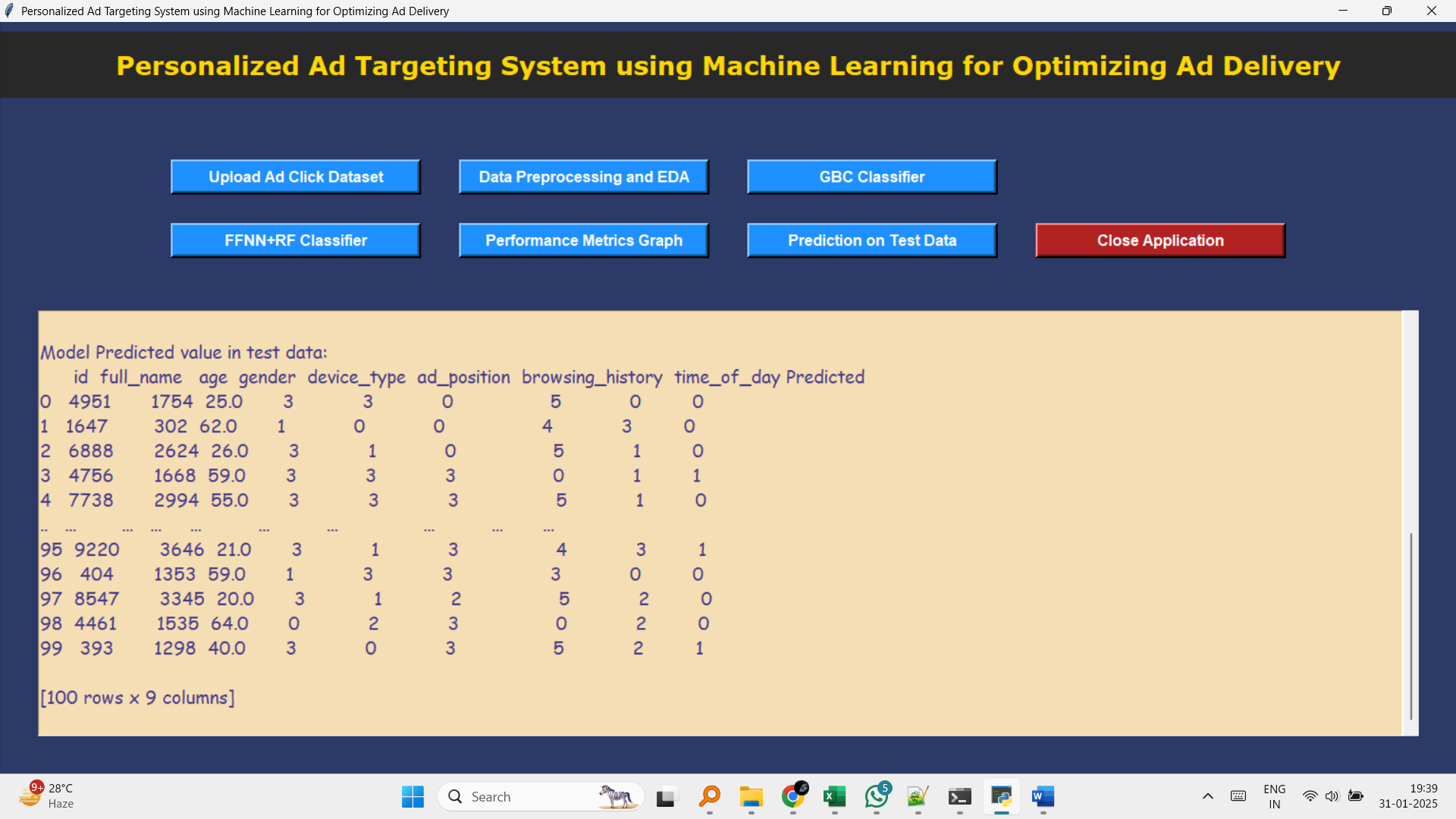
* **Accuracy**: 91.51% – The FFNN+RFC model accurately predicts ad click behavior for more than 90% of the test data.
* **Precision**: 92.08% – The model achieves high precision in predicting clicked ads, reducing the occurrence of false positives.
* **Recall**: 91.27% – The recall indicates that the model correctly identifies most actual clicked ads, minimizing false negatives.
* **F1-Score**: 91.43% – The F1-score reflects the model's balanced ability to predict both clicked and non-clicked ads with high accuracy. The confusion matrix shows a strong performance, with a high number of true positives and true negatives, indicating that the model can effectively differentiate between clicked and non-clicked ads.





**Figure 7: Performance Comparison Graph of Models**

Figure 7 presents a graph comparing the performance of the different models (GBC, FFNN, and FFNN+RFC) based on accuracy, precision, recall, and F1-score. The graph visually highlights the superior performance of the FFNN+RFC classifier model, which outperforms the other models in all metrics. This comparison aids in understanding the trade-offs between different models and selecting the best approach for predicting ad clicks in the given dataset. The FFNN+RFC model’s higher accuracy and balanced performance metrics indicate its effectiveness in solving the ad click prediction problem.



**Figure 6: Model Prediction on the Test Data**

Figure 6 illustrates the predictions made by the trained model on the test dataset. The model outputs a predicted label for each test instance, determining whether an ad will be clicked or not. The predictions are compared against the actual outcomes, allowing the user to visualize the model's performance on unseen data. This step is crucial for evaluating how well the model generalizes to new, unseen data, which is a key indicator of its effectiveness in real-world applications.

**CHAPTER 11**

**CONCLUSION AND FUTURE SCOPE**

The research aimed to develop a machine learning model that can predict whether a user will click on an online advertisement based on various features such as age, gender, device type, browsing history, and time of day. Several machine learning algorithms were explored, including Gradient Boosting Classifier (GBC) and a hybrid approach combining Feedforward Neural Networks (FFNN) and Random Forest Classifier (RFC).

Through data preprocessing, the dataset was cleaned and split into training and testing sets. Feature engineering was performed to extract meaningful patterns from the data, and multiple models were trained and evaluated based on their performance. The GBC classifier demonstrated a solid performance in predicting user click behavior, while the proposed FFNN + RFC hybrid model showed promise in improving accuracy and providing better predictive results.

The model's ability to predict user interactions with ads is valuable in digital marketing, allowing businesses to target users more effectively, optimize advertising strategies, and enhance user engagement. The project successfully demonstrated how machine learning can be applied to real-world scenarios in online advertising.

**Future Scope:**

* **Model Enhancement**: Further improvements can be made to the current models by exploring more advanced techniques, such as deep learning models, or implementing additional ensemble methods. Hyperparameter tuning and model optimization can help improve the prediction accuracy.
* **Incorporation of More Features**: Additional features, such as user location, device specifications, or historical interaction with similar ads, can be included to provide deeper insights into user behavior and further enhance the model's predictive capabilities.
* **Real-Time Prediction**: The model could be adapted for real-time predictions in a production environment, where user interaction data is processed continuously, allowing businesses to adjust ad campaigns dynamically based on user behavior.
* **Cross-Platform Integration**: The model can be extended to handle cross-platform predictions, where user behavior across different devices (smartphones, tablets, desktops) and platforms (websites, mobile apps) is considered to optimize ad targeting.
* **User Segmentation**: By implementing unsupervised learning techniques such as clustering, the model could be used to segment users into different groups based on their interaction patterns. These segments could then be targeted with more personalized advertisements, improving engagement and conversion rates.
* **Handling Imbalanced Data**: The current approach may be impacted by imbalanced classes (with fewer clicks than non-clicks). Future work could explore techniques like oversampling, undersampling, or synthetic data generation (e.g., SMOTE) to address this issue and improve model performance.
* **Ethical Considerations and Privacy**: With an increased focus on user privacy, future iterations of the project should incorporate privacy-preserving techniques and ensure compliance with data protection regulations, such as GDPR. Models should be designed to minimize data privacy concerns while maximizing their effectiveness.

**REFERENCES**

1. Ungureanu, A., & Popescu, D. L. (2022). ONLINE ADVERTISING-HISTORY, EVOLUTION, AND CHALLENGES. Revista Economică, 74(3).
2. Watkins, E. A. (2019). Guide to advertising technology.
3. Ezzat, R. (2023). History and development of advertising through media technology. J Art Des Music, 2, 3.
4. Dahlén, M., & Edenius, M. (2007). When is advertising advertising? Comparing responses to non-traditional and traditional advertising media. Journal of Current Issues & Research in Advertising, 29(1), 33-42.
5. Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Müller, K. R. (2021). Towards CRISP-ML (Q): a machine learning process model with quality assurance methodology. Machine Learning and Knowledge Extraction, 3(2), 392-413.
6. Wall, P. D., & Fontenot, J. D. (2020). Application and comparison of machine learning models for predicting quality assurance outcomes in radiation therapy treatment planning. Informatics in Medicine Unlocked, 18, 100292.
7. Khankhoje, R. (2023). Quality Assurance in the Age of Machine Learning. Quality Assurance in the Age of Machine Learning, 13(10), 10-29322.
8. Lee, H., & Cho, C. H. (2020). Digital advertising: present and future prospects. International Journal of Advertising, 39(3), 332-341.
9. Choi, J. A., & Lim, K. (2020). Identifying machine learning techniques for classification of target advertising. ICT Express, 6(3), 175-180.
10. Ullal, M. S., Hawaldar, I. T., Soni, R., & Nadeem, M. (2021). The role of machine learning in digital marketing. Sage Open, 11(4), 21582440211050394.
11. Miralles-Pechuán, L., Ponce, H., & Martínez-Villaseñor, L. (2018). A novel methodology for optimizing display advertising campaigns using genetic algorithms. Electronic Commerce Research and Applications, 27, 39-51.
12. Nuara, A., Trovò, F., Gatti, N., & Restelli, M. (2022). Online joint bid/daily budget optimization of internet advertising campaigns. Artificial Intelligence, 305, 103663.
13. Nuara, A. (2020). Machine learning algorithms for the optimization of internet advertising campaigns.
14. Daghistani, T., & Alshammari, R. (2020). Comparison of statistical logistic regression and random forest machine learning techniques in predicting diabetes. Journal of Advances in Information Technology, 11(2), 78-83.
15. Greene, D., Cunningham, P., & Mayer, R. (2008). Unsupervised learning and clustering. Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval, 51-90.
16. Cremonesi, F., Hager, G., Wellein, G., & Schürmann, F. (2020). Analytic performance modeling and analysis of detailed neuron simulations. The International Journal of High Performance Computing Applications, 34(4), 428-449.
17. Phillips, J. J. (1998). The return-on-investment (ROI) process: Issues and trends. Educational Technology, 38(4), 7-14.
18. Ichsani, S., & Suhardi, A. R. (2015). The effect of return on equity (ROE) and return on investment (ROI) on trading volume. Procedia-Social and Behavioral Sciences, 211, 896-902.
19. Dahmen, J., & Cook, D. (2019). SynSys: A synthetic data generation system for healthcare applications. Sensors, 19(5), 1181.