PREDICTIVE ANALYTICS ON RETAIL BANKING

Project Report submitted to Alagappa University in partial fulfillment of the requirements for the Award of the degree of

MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Submitted by

C.MAHESWARI
(REG.NO:2023557016)

Under the Guidance of
Dr.P.ESWARAN, M.Sc, M.Tech., Ph.D.,
Assistant Professor

DEPARTMENT OF COMPUTER APPLICATIONS

ALAGAPPA UNIVERSITY

(A State University Accredited with 'A++' Grade by NAAC (CGPA:3.59)

in the Fourth Cycle, Under Dual Mode Category)

KARAIKUDI-630003

MAY-2025

CERTIFICATE

This is to certify that the Main project entitled, **"PREDICTIVE ANALYTICS ON RETAIL BANKING"** is a Bonafide work done by **C.MAHESWARI (REG.NO:2023557016)** is submitted in partial fulfilment of the requirement for the award of the degree of MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE AND DATA SCIENCE has been carried out under our supervision and guidance from December 2024 to May 2025.

HEAD OF THE DEPARTMENT

Dr.V. PALANISAMY, MCA,M.Tech.,Ph.D.,
Senior Professor & Head,
Dept. of Computer Applications,
Alagappa University,
Karaikudi.

INTERNAL GUIDE

Dr.P.ESWARAN, M.Sc,M.Tech.,Ph.D.,
Assistant Professor,
Dept. of Computer Applications,
Alagappa University,
Karaikudi.

Submitted for Viva-Voce Examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINAR

ACKNOWLEDGEMENT

I acknowledge my sincere gratitude to the Almighty who sustained me with his powerful blessings to complete various stages of this project work.

I wish to extend my sincere thanks and gratitude to **Dr.V.PALANISAMY, MCA, M.Tech., Ph.D., Senior Professor and Head,** Department of Computer Applications for permitting me to take up this project.

I wish to extend my sincere thanks to my guide **Dr.P.ESWARAN, M.Sc., M.Tech., Ph.D., Assistant Professor,** Department of Computer Applications, Alagappa University Karaikudi, for his enthusiastic, Valuable support and timely suggestions in indispensable situation during my project work.

I would also like to extend my sincere thanks to all staff members of the Department of Computer Applications who encouraged me to finish this project.

Finally, I wish to thank my family members and friends for their constant encouragement and valuable suggestions.

C.MAHESWARI

ABSTRACT

In today's highly competitive financial landscape, customer acquisition and retention are critical to the success of retail banks. Traditional marketing campaigns often lack precision in targeting the right audience, resulting in inefficient use of resources and missed growth opportunities. This project, titled Predictive Analytics for Retail Banking, employs machine learning techniques to forecast whether a customer is likely to subscribe to a term deposit product. Using a real-world dataset derived from past marketing campaigns, the system performs extensive data preprocessing, feature engineering, and model training using multiple machine learning algorithms, including Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors, and XGBoost. The model achieving the highest predictive performance is selected and saved for deployment. The primary objective is to enable data-driven decision-making by predicting customer behaviour based on various features such as age, occupation, loan history, and contact method. This predictive capability empowers banks to design more effective marketing strategies, reduce operational costs, and enhance customer conversion rates. The entire project is implemented in Python, developed using Google Colab, and integrated with a Streamlit frontend for user interaction. A PostgreSQL backend is used to store the selected model and manage customer data, enabling real-time predictions and a seamless user experience.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF PROJECT

The banking sector generates massive volumes of customer data daily, yet much of this data remains underutilized. One of the significant challenges retail banks face is predicting customer interest in financial products, such as term deposits, and targeting them effectively. This project, *Predictive Analytics for Retail Banking*, focuses on applying machine learning techniques to predict whether a customer will subscribe to a term deposit based on historical marketing campaign data.

The system uses a real-world dataset containing various customer attributes such as age, job, marital status, education, loan status, and contact type. Machine learning models are trained to classify customers into two categories: those who are likely to subscribe to a term deposit and those who are not. This prediction enables banks to make informed decisions and design efficient marketing strategies, thereby improving their overall performance and customer engagement.

In model building, the project emphasizes deployment and usability. The best-performing model is integrated with a user-friendly Streamlit web interface, allowing users to input customer data and receive real-time predictions. The backend storage using PostgreSQL ensures data persistence, making the system suitable for practical use in financial institutions. Ultimately, the project aims to bridge the gap between data analysis and actionable insights in the retail banking domain.

1.2 SYSTEM ENVIRONMENT

The system environment defines the platform, tools, and technologies used throughout the development and execution of the project. A well-defined environment ensures consistency, compatibility, and efficient performance.

This project **Predictive Analytics for Retail Banking** was developed using a cloud-based environment combined with modern data science tools and frameworks to support data processing, machine learning model development, and user interface integration.

1.2.1 HARDWARE REQUIREMENTS

PROCESSOR	INTEL CORE i7
CPU CLOCK SPEED	2.80 GHz
RAM	16GB
HARD DISC CAPACITY	500GB
DISPLAY TYPE	INTEGRATED GPU
INTERNET CONNECTION	STABLE INTERNET CONNECTION

1.2.2 SOFTWARE REQUIREMENTS

PROGRAMMING LANGUAGE	PYTHON
OPERATING SYSTEM	WINDOWS 11
FRONT-END FRAMEWORK	STREAMLIT
DEVELOPING ENVIRONMENT	GOOGLE COLABATORY
DATASET	MICROSOFT EXCEL
TEXT EDITOR	VISUAL STUDIO

1.3 SOFTWARE SPECIFICATION

1.3.1 FRONT-END

1. PROGRAMMING LANGUAGE – PYTHON

Python is an open-source, high-level programming language known for its simplicity and versatility. Created by Guido van Rossum in 1991, Python supports multiple paradigms, including procedural, object-oriented, and functional programming. It is dynamically typed, interpreted, and widely used in areas such as web development, automation, artificial intelligence, and data science.

REASONS FOR CHOOSING PYTHON:

- **Ease of Use:** Clean syntax reduces development time.
- **Rich Libraries:** Libraries like pandas, numpy, scikit-learn, matplotlib simplify machine learning workflows.
- **Strong Community:** Vast resources, forums, and third-party tools.
- **Fast Prototyping:** Suitable for iterative development, ideal for ML projects.

PYTHON IN DATA SCIENCE:

- **Data Manipulation:** pandas allows efficient data wrangling.
- **Computation:** numpy and scipy handle numerical tasks.
- **Machine Learning:** scikit-learn and xgboost support various algorithms.
- **Visualization:** matplotlib, seaborn, and plotly provide visual insights.
- **Integration:** Works well with tools like Google Colab, Streamlit, and PostgreSQL.

IN THIS PROJECT:

Python is used throughout the machine learning pipeline from data preprocessing and model training to evaluation and deployment.

2. DEVELOPING ENVIRONMENT: GOOGLE COLABORATORY

INTRODUCTION TO GOOGLE COLABORATORY:

Google Colaboratory, commonly known as **Google Colab**, is a free, cloud-based development environment provided by Google. It is built on top of Jupyter Notebook and allows users to write and execute Python code in a web-based interface. It allows running Python code with zero setup and provides access to GPUs/TPUs. Google Colab is widely used in the fields of machine learning, deep learning, and data science due to its flexibility, ease of use, and seamless integration with the Google ecosystem.

REASONS FOR CHOOSING GOOGLE COLAB OVER JUPYTER NOTEBOOK:

- **No Installation Required**: Colab runs entirely on the cloud. There's no need to install Python, packages, or set up environments on your local machine.
- **Free GPU/TPU Access**: Users can take advantage of Google's hardware to accelerate heavy computations like model training at no cost.
- **Cloud Storage Integration**: Colab is tightly integrated with **Google Drive**, making it easy to save and load datasets or models directly from the cloud.
- **Easy Sharing**: Colab notebooks can be shared just like Google Docs, enabling smooth collaboration with peers or instructors.
- **Pre-installed Libraries**: Common data science and machine learning libraries like numpy, pandas, scikit-learn, and matplotlib are pre-installed, saving time during setup.

IN THIS PROJECT:

Colab was used for data preprocessing, model training, visualization, and testing, enabling efficient experimentation without local hardware dependency.

3. FRONT-END FRAMEWORK: STREAMLIT

INTRODUCTION TO STREAMLIT

Streamlit is an open-source Python library that enables developers and data scientists to create interactive web applications for data science and machine learning projects—with minimal effort and zero front-end knowledge required. Unlike traditional web development frameworks that require HTML, CSS, or JavaScript, Streamlit allows you to build intuitive dashboards and user interfaces entirely using Python.

It's specifically designed for fast prototyping and sharing of machine learning models and data analytics workflows, making it a perfect fit for this project.

FEATURES OF STREAMLIT

- Build apps with pure Python.
- Real-time interactivity with widgets.
- Supports data visualization libraries.
- Auto-refresh with every script update.
- Easy deployment options.

IN THIS PROJECT:

Streamlit acts as the **frontend interface** that allows end-users (bank officials or analysts) to interact with the machine learning model.

It's used as:

- **User Input Form**: The app includes an input form where users can enter details such as age, job, marital status, contact type, loan status, etc.
- **Prediction Output**: Based on the input, Streamlit sends the data to the trained model and displays the result whether the customer is likely to accept a term deposit offer.
- **Visual Feedback**: Streamlit also shows additional visuals or performance metrics, which help explain how the model is making decisions.
- **Database Integration**: The frontend can communicate with a PostgreSQL backend to store or retrieve customer prediction data, enabling persistent storage of results.

4. BROWSER COMPATIBILITY

GENERAL OVERVIEW:

Browser compatibility refers to how well a web application or interface performs across different internet browsers. Since users may access the application using various platforms—such as Chrome, Firefox, Safari, Edge, or Opera—it's essential that the frontend is responsive, stable, and consistent across all commonly used browsers.

BROWSER COMPATIBILTY IN THIS PROJECT:

The frontend for this project is built using **Streamlit**, which runs on the browser and is known for its excellent cross-browser compatibility. Streamlit applications are rendered as lightweight, server-hosted web apps that can be accessed through any modern web browser without requiring any installation from the user's end.

This project has been tested on commonly used browsers including:

- **Google Chrome**
- **Mozilla Firefox**
- **Microsoft Edge**

The Streamlit interface works smoothly across all these platforms with consistent styling, layout rendering, and interactive functionality.

5. LIBRARIES USED:

NumPy: Used for numerical computing and efficient array operations, essential for data transformation tasks.

Pandas: Provides powerful data structures for data manipulation and analysis, especially DataFrames for tabular data.

Seaborn: A statistical data visualization library that makes it easy to create informative and attractive charts.

Matplotlib: A 2D plotting library used for generating visualizations such as bar graphs, histograms, and line charts.

Scikit-learn: A comprehensive library for machine learning that supports a wide range of algorithms and model evaluation techniques.

XGBoost: An advanced gradient boosting framework for high-performance predictive modeling.

Streamlit: Used to develop an interactive web-based user interface.

Psycopg2: A PostgreSQL database adapter to connect and store prediction results.

Pickle: For saving and loading trained models.

Google Colaboratory: The development environment used to write and execute Python code in the cloud.

6. TEXT EDITOR - VISUAL STUDIO CODE (VS CODE)

Visual Studio Code (VS Code) is a powerful, lightweight, open-source code editor developed by Microsoft. It supports development in multiple programming languages, including Python, which is used in this project. For the Predictive Analytics for Retail Banking project, VS Code plays a vital role in writing, organizing, and running the Streamlit frontend application.

FEATURES OF VS CODE:

- **Integrated Terminal:** Used to run Streamlit commands (streamlit run app.py) directly from the editor.
- **Python Support:** Offers syntax highlighting, IntelliSense, debugging, and linting specifically for Python code.
- **Streamlit Extension Support:** Provides code suggestions and formatting for Streamlit-specific components, which makes frontend development smoother.
- **Version Control Integration:** Git integration allows easy tracking of code changes and collaboration, if needed.
- **Extensions Marketplace:** Useful extensions like *Python*, *Pylance*, *Jupyter*, and *PostgreSQL* enhance productivity during development.

IN THIS PROJECT:

- **Development Environment for Streamlit App:** VS Code is used to build and run the Streamlit user interface that interacts with the trained machine learning model.
- **Database Connection Integration:** The PostgreSQL database connection logic is coded and tested within VS Code.
- **Frontend Testing and Debugging:** The real-time preview and terminal in VS Code allow easy testing and debugging of the frontend.

EXECUTION:

Run locally using the command:

streamlit run app.py

App is hosted on **http://localhost:8501**.

1.3.2 BACK-END

1. DATASET DESCRIPTION

The dataset used in this project forms the foundation for predictive analytics in retail banking. It is provided in **CSV (Comma-Separated Values)** format, which is a widely used and easily accessible data storage format ideal for structured data analysis. It is derived from a Portuguese retail bank's telemarketing campaign data, where the objective was to encourage clients to subscribe to a term deposit product. The dataset provides valuable insight into customer demographics, financial standing, communication details, and past campaign outcomes, all of which are used to predict customer behaviour.

DATASET OVERVIEW

- **Total Records (Rows):** 11,162
- **Total Features (Columns):** 17
- **File Format:** CSV (.csv)
- **Type of Problem:** Binary Classification
- **Target Variable:** deposit (indicates whether the customer subscribed to a term deposit)
- **Source:** Real-world banking campaign data

This structured dataset is highly suitable for developing and testing machine learning models due to its balance of numerical and categorical features, its real-world context, and a clear binary output variable.

DETAILED FEATURE DESCRIPTION

Feature	Description
age	Age of the customer (numeric).
job	Job title or profession (categorical: e.g., admin, technician, services).
marital	Marital status of the customer (e.g., married, single, divorced).
education	Highest level of education attained.
default	Indicates if the customer has credit in default (yes/no).
balance	Account balance in euros (numeric).
housing	Indicates if the customer has a housing loan (yes/no).
loan	Indicates if the customer has a personal loan (yes/no).
contact	Communication type used (e.g., cellular, telephone, unknown).
day	Last contact day of the month (numeric).
month	Last contact month (e.g., may, jun).
duration	Duration of the last contact in seconds (numeric).
campaign	Number of contacts performed during this campaign.
pdays	Days passed since the client was last contacted (-1 means never contacted before).
previous	Number of contacts performed before this campaign.
poutcome	Outcome of the previous marketing campaign (e.g., success, failure, unknown).
deposit	**Target variable** – whether the client subscribed to a term deposit (yes/no).

PREPROCESSING:

To ensure the data is ready for machine learning, preprocessing steps included:

- **Converting categorical features** into numerical form using encoding techniques.
- **Handling missing or unknown values** appropriately.
- **Normalizing or scaling** features like balance, duration, and pdays.
- **Splitting the data** into training and testing sets for fair model evaluation.

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes
42	managemen	single	tertiary	no	0	yes	yes	unknown	5	may	562	2	-1	0	unknown	yes
56	managemen	married	tertiary	no	830	yes	yes	unknown	6	may	1201	1	-1	0	unknown	yes
60	retired	divorced	secondary	no	545	yes	no	unknown	6	may	1030	1	-1	0	unknown	yes
37	technician	married	secondary	no	1	yes	no	unknown	6	may	608	1	-1	0	unknown	yes
28	services	single	secondary	no	5090	yes	no	unknown	6	may	1297	3	-1	0	unknown	yes

Sample of the Dataset

2. SAVING THE BEST MODEL – POSTGRESQL

PostgreSQL is a powerful, open-source relational database that is well-suited for structured data storage, and its compatibility with Python makes it an excellent choice for integrating with machine learning workflows.

In this project, PostgreSQL serves as the central database system used to store the best-performing machine learning model and manage its integration with the Streamlit frontend.

ROLE OF POSTGRESQL:

1. Model Storage

After evaluating multiple machine learning models, the best-performing model was serialized and stored securely in the PostgreSQL database. This approach ensures that the model can be retrieved dynamically for predictions without needing to retrain or re-upload it.

2. Streamlit Integration

The Streamlit frontend connects directly to the PostgreSQL database to fetch the saved model and use it for real-time prediction.

3. Efficient and Reliable

PostgreSQL ensures data consistency and security, making it a reliable choice for storing model files and handling any future data, such as prediction logs or user input history.

4. Scalable for Deployment

Using PostgreSQL also prepares the project for potential future scaling, where additional models, user data, or prediction histories might be stored and managed effectively.

CHAPTER 2

SYSTEM ANALYSIS

System analysis involves understanding the current problem, identifying the limitations of the existing approach, and defining the functionalities of the proposed system. The goal is to bridge the gap between raw data and strategic banking decisions through a predictive analytics model. This analysis lays the foundation for a robust solution that transforms data into meaningful predictions, helping banks make informed, cost-effective decisions in their marketing campaigns.

2.1 EXISTING SYSTEM

In most traditional banking environments, marketing teams rely on generic methods like mass email campaigns, cold calls, or broad advertisements to promote financial products such as term deposits. These approaches often treat all customers the same, without truly understanding their individual preferences or behaviours.

As a result, banks end up reaching out to many people who are not even interested in such products. This not only leads to a low response rate but also wastes time, effort, and money. Marketing decisions are usually based on past trends or assumptions rather than real-time data, which limits the effectiveness of campaigns.

There’s no predictive element in place banks are essentially guessing which customers might be interested and hoping for a good outcome. With no intelligent system to analyze patterns or customer history, it becomes difficult to tailor the right product to the right person at the right time.

Additionally, studies show that traditional mass marketing techniques often yield a response rate of less than 2%, while data-driven campaigns can achieve over 10% conversion. This highlights the urgent need for smarter, more targeted approaches in the banking sector.

2.2 PROPOSED SYSTEM

The proposed system brings intelligence and automation into the way banks handle customer targeting. Instead of relying on guesswork, it uses machine learning to analyze past customer behaviour and predict who is most likely to subscribe to a term deposit.

By training various machine learning models—such as Logistic Regression, Random Forest, Decision Tree, K-Nearest Neighbors, and Support Vector Machines—on real marketing campaign data, the system learns to recognize patterns like how age, job, loan status, or contact method can influence a customer's decision. It then uses these patterns to make accurate predictions for new customers.

What makes this system powerful is that it doesn’t just stop at model building. It’s connected to a user-friendly Streamlit web interface, where bank staff or marketing teams can simply input customer details and get instant predictions. Behind the scenes, a PostgreSQL database stores all relevant data and results for future reference or analysis.

Moreover, the system is designed to be scalable and modular. As the bank’s customer base grows, the system can handle larger datasets and be updated with new features or algorithms. It is also flexible enough to support the integration of more advanced models or APIs in the future.

This smarter, data-driven approach helps banks run more targeted campaigns, reduce unnecessary spending, and increase their chances of converting customers making every marketing move more meaningful and effective.

2.3 FEASIBILITY STUDY

2.3.1 TECHNICAL FEASIBILITY

From a technical perspective, the project uses Python a language that's popular, powerful, and has a ton of libraries specifically for data science and machine learning. Development is done in Google Colab, which is free to use and doesn't require any setup. For the user interface, we've chosen Streamlit, which makes it super easy to turn code into a clean, interactive web app. PostgreSQL handles the data storage on the backend efficiently.

All the tools are open-source, well-supported, and widely used in the industry, so there's no need for expensive software or complicated infrastructure.

The only potential challenge could be ensuring data security and model versioning, especially as the system scales. However, these concerns can be addressed by implementing proper encryption for sensitive data and maintaining clear documentation for model updates.

2.3.2 OPERATIONAL FEASIBILITY

From a usability standpoint, the system is designed to be straightforward and practical. It doesn't require users to have a technical background. Anyone from a marketing or banking team can input customer details through the Streamlit interface and instantly get predictions. The process is automated, fast, and intuitive—making it easy to integrate into everyday workflows.

The system also reduces manual effort and decision-making delays, helping teams take action quickly and with more confidence.

In terms of data privacy, the system ensures secure handling of customer data, with all inputs and predictions confined to secure connections and role-based access control if deployed at scale. This makes the system both user-friendly and compliant with data protection standards like GDPR or other local regulations.

2.3.3 ECONOMIC FEASIBILITY

Financially, the system makes a lot of sense. All development tools are free, which keeps costs low. More importantly, by accurately targeting potential customers, banks can save money on ineffective marketing efforts and increase their return on investment (ROI). The system helps focus resources where they'll have the most impact, reducing waste and improving campaign success rates.

For example, if a bank typically spends $100,000 on a generic email marketing campaign and achieves a 2% response rate, they're only reaching 2,000 customers. However, by using the proposed system to identify high-potential leads, the same budget could result in a 10% conversion rate—reaching 10,000 customers and potentially increasing revenue fivefold.

This makes the system not just cost-effective but a strategic asset in long-term marketing planning.

CHAPTER 3

SYSTEM DESIGN

The system design of the Predictive Analytics for Retail Banking project provides a clear framework for how the entire system operates from raw data collection to making intelligent predictions using machine learning. This design acts as the foundation that ensures smooth interaction between various components such as the backend processing, the machine learning pipeline, and the user-facing interface.

The project is designed to forecast whether a customer will subscribe to a term deposit based on features like age, occupation, marital status, loan history, and past marketing outcomes. To achieve this, the system uses real-time user inputs, processes them through a trained ML model, and presents predictions in an intuitive web interface built using Streamlit.

The system is structured in a modular and scalable way, which not only ensures ease of development and debugging but also allows for future enhancements—like integrating new features, models, or real-time bank databases.

3.1 SYSTEM ARCHITETURE

The system architecture for the *Predictive Analytics for Retail Banking* project ensures a smooth data flow from ingestion to prediction. It integrates data science processes with a user-friendly web interface to provide accurate, real-time predictions of customer subscription behaviour. The architecture follows a modular, layered design that supports scalability, maintainability, and future enhancements.

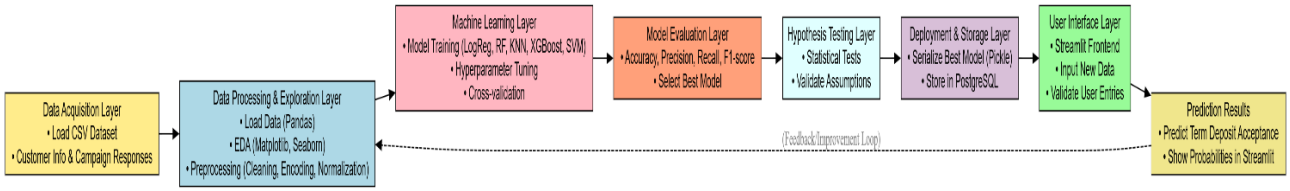


Figure 1: System Architecture

LAYER DESCRIPTIONS

1. DATA ACQUISITION LAYER

Begins with uploading a CSV dataset containing customer details and campaign responses for analysis.

2. DATA PROCESSING & EXPLORATION LAYER

Involves loading the dataset with Pandas, performing EDA using Matplotlib/Seaborn to spot trends and outliers, and preprocessing (cleaning, encoding, normalization).

3. MACHINE LEARNING LAYER

Trains multiple models (Logistic Regression, Random Forest, KNN, XGBoost, SVM) with hyperparameter tuning and cross-validation to optimize performance.

4. MODEL EVALUATION LAYER

Evaluates models based on accuracy, precision, recall, and F1-score. The best-performing model is selected for deployment.

5. HYPOTHESIS TESTING LAYER

Applies statistical tests to validate insights and ensure results are not due to randomness.

6. DEPLOYMENT & STORAGE LAYER

Serializes the best model using Pickle and stores it in PostgreSQL for secure, persistent access.

7. USER INTERFACE LAYER

Uses Streamlit to create a frontend where users can input data and interact with the model. Inputs are validated to ensure accuracy.

8. PREDICTION RESULT

Displays prediction outcomes and probability scores, indicating a customer's likelihood of subscribing to a term deposit.

3.2 INPUT DESIGN

Input Design is a crucial part of system development that focuses on how users interact with the system and how raw data is entered, validated, and processed into useful information. A well-structured input design ensures accuracy, efficiency, and a user-friendly experience.

In the **Predictive Analytics for Retail Banking** project, input design revolves around how data is collected, formatted, and passed into the machine learning model for processing and prediction.

USER INPUT FLOW:

Users interact with the system through the **Streamlit UI**, where they are prompted to input specific customer-related data such as:

Age
Job (e.g., admin, technician, etc.)
Marital Status
Education Level
Default Status
Housing Loan
Personal Loan
Contact Communication Type
Day, Month, Duration
Campaign-related details
Previous outcomes

INPUT VALIDATION:

The system is designed to validate user inputs to ensure:

- Correct data types (e.g., integers for age, binary for loan status)
- Mandatory fields are not left empty
- Proper formatting and logical values (e.g., age must be positive)

This step prevents any inaccurate or harmful data from being fed into the machine learning model.

BACKEND INPUT FLOW:

Input design also includes the **data pipeline**:

- The .csv dataset is loaded and read into a DataFrame.
- Data is explored, cleaned, and preprocessed through encoding, scaling, and handling missing values.
- The refined inputs are then used to train multiple machine learning models.

3.3 OUTPUT DESIGN

Output design is one of the most important phases in system development because it determines how the results of data processing are presented to the users. In this project, the goal of the output design is to deliver the results of predictive analytics in a clear, user-friendly, and interactive manner so that stakeholders can make informed decisions efficiently.

PURPOSE OF OUTPUT DESIGN

- Display whether a customer is likely to subscribe to a term deposit based on the input features.
- Showcase model performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix.
- Present visual analytics like feature importance and model comparisons to help understand the decision process.

IN THIS PROJECT:

1. **Prediction Result**
    - Displays whether the given customer profile is predicted to subscribe to the term deposit (Yes/No).
    - Shown via Streamlit UI after model prediction using the best-performing ML algorithm.

2. **Model Performance Metrics**

    - Includes accuracy score, confusion matrix, precision, recall, F1-score, and ROC-AUC score.

    - Helps to evaluate and compare the effectiveness of different ML models.

3. **Visualizations**

    - Plots such as heatmaps, feature importance graphs, and comparative model performance bar charts are displayed.

    - These visuals support users in understanding data patterns and model reasoning.

4. **User Interface Output**

    - The front-end is developed using Streamlit, which dynamically updates and displays results based on user input.

    - It ensures real-time feedback and an interactive experience.

OUTPUT FORMAT

- Textual: Prediction outputs and performance metrics are shown in text format.
- Graphical: Charts, graphs, and plots are shown for better interpretation.
- Interactive: Users can input new data and instantly receive predictions and insights.

3.4 DATA FLOW DIAGRAM:

The **Data Flow Design** is a critical component of system design that defines the flow of data within the system and its interaction between various modules. In this project, the Data Flow Design outlines how data is collected, processed, and used throughout the system to predict whether a bank customer will accept a term deposit offer. This process is essential to ensure that data moves seamlessly from input to output while maintaining integrity, security, and efficiency.

LEVEL 0 (CONTEXT LEVEL)

The Level 0 Data Flow Diagram (DFD) represents the entire **Predictive Analytics for Retail Banking** system as a single process, showing how external entities interact with it and how data flows between them. This high-level view is essential for understanding the scope of the system and its primary interfaces.

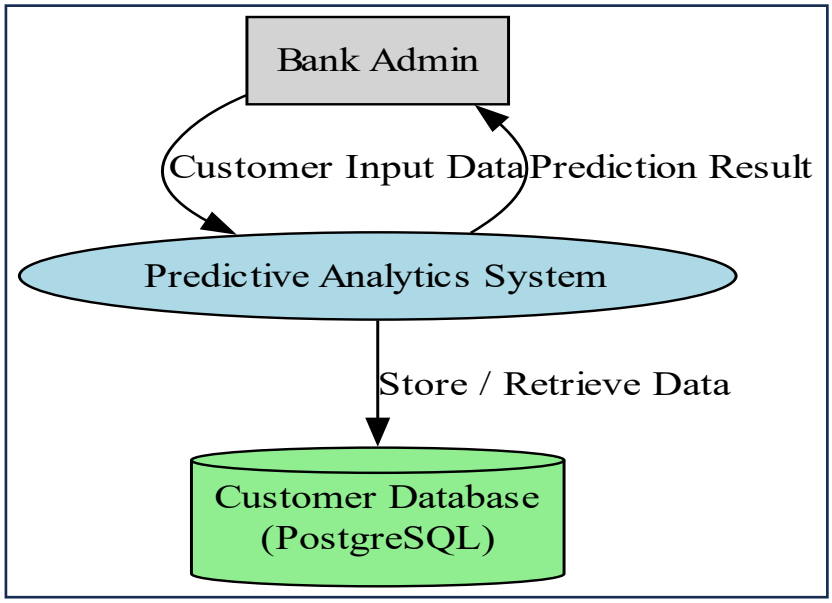


Figure 2: Level 0(Context Level)

EXTERNAL ENTITIES:

1. Bank Admin

The Bank Admin is the primary user of the system. They provide input data regarding customers (e.g., age, income, loan history, etc.) through a user interface (built using Streamlit) and receive prediction results from the system.

2. Customer Database (PostgreSQL)

This is the backend data store that holds historical and new customer data. It is also used to store the results of predictions for future reference and analysis.

MAIN PROCESS:

Predictive Analytics System

This is the central processing unit of the application. It receives customer data, preprocesses it, applies feature selection, uses a trained machine learning model to generate predictions, and then stores or retrieves relevant data from the PostgreSQL database.

DATA FLOWS:

1. **Customer Input Data** (from Bank Admin to Predictive Analytics System) The Bank Admin inputs customer features such as age, job, marital status, loan history, balance, etc., into the system.
2. **Store / Retrieve Data** (between Predictive Analytics System and Database) The system stores new data entries and prediction results in the PostgreSQL database. It may also retrieve historical data for model validation or monitoring purposes.

3. **Prediction Result** (from Predictive Analytics System to Bank Admin) After processing the data through the machine learning model, the system returns the predicted outcome whether the customer is likely to accept a term deposit or not—to the Bank Admin.

LEVEL 1 – DETAILED VIEW

The Level 1 Data Flow Diagram offers a more granular view of the system's operations by decomposing the main process "Prediction Term Deposit Acceptance" into smaller components. This breakdown helps understand how input data is handled, processed, stored, and transformed into useful predictions.

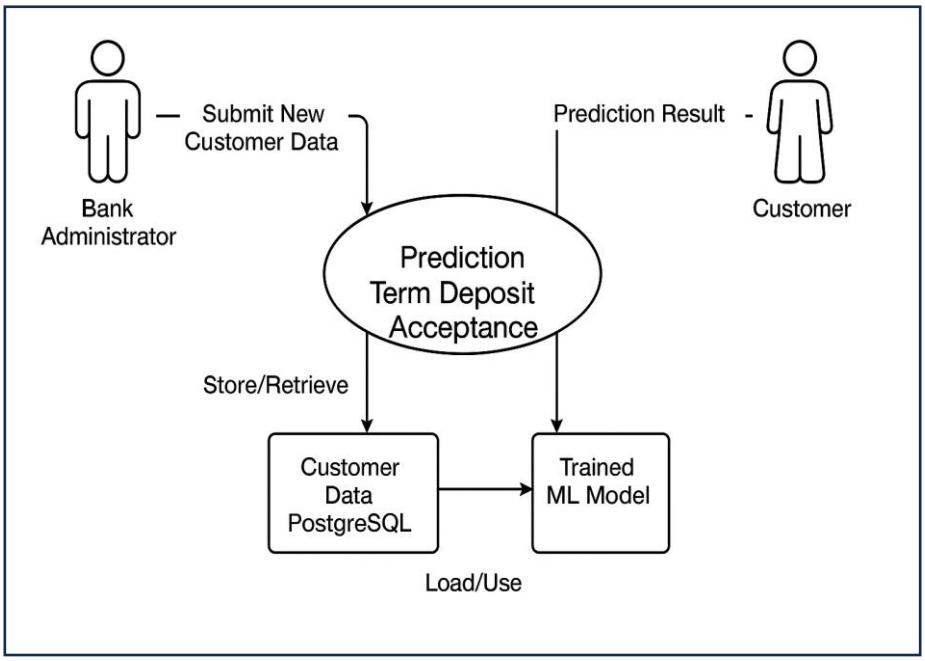


Figure 3: Level 1(Detailed View)

ENTITIES:

1. **Bank Administrator**

The Bank Admin interacts with the system through a user-friendly interface (Streamlit). They enter customer-related data including age, job type, marital status, loan history, and account balances.

2. **Customer**

Though not directly interacting with the system, the customer is the subject of the prediction. The final outcome of the system is focused on determining the customer's likelihood of subscribing to a term deposit.

PROCESSES - Prediction Term Deposit Acceptance

- Receiving customer input data
- Preprocessing and selecting features
- Using a trained ML model for prediction
- Storing and retrieving data in the PostgreSQL database
- Delivering results to the admin

DATA STORES:

1. Customer Data (PostgreSQL Database)

This is the central storage for:

- Historical customer data
- New data entered by the Bank Admin
- Prediction results for tracking and analysis The system performs **Store** operations (writing new entries) and **Retrieve** operations (fetching past data or updating results).

2. Trained ML Model

A pre-trained machine learning model (stored using pickle) is loaded and used to evaluate whether a customer is likely to accept a term deposit offer. This model was built using various supervised learning algorithms during the training phase in Google Colab.

DATA FLOWS:

1. **Submit New Customer Data**

   The Bank Admin submits the required customer data into the system.

2. **Store/Retrieve**

   The system either stores the new customer entry or retrieves historical data for validation or display.

3. **Load/Use ML Model**

   The trained model is loaded into the system and used to process the customer data.

4. Prediction Result

After the model evaluates the input, the system returns a prediction (“Yes” or “No”) indicating the likelihood that the customer will accept the term deposit offer.

3.5 MODULE DESCRIPTION

This project is organized into ten key modules, each building upon the previous to create a complete end-to-end machine learning system for predicting customer responses to term deposit offers. From data exploration to model deployment, each stage is thoughtfully designed to ensure accuracy, reliability, and user accessibility.

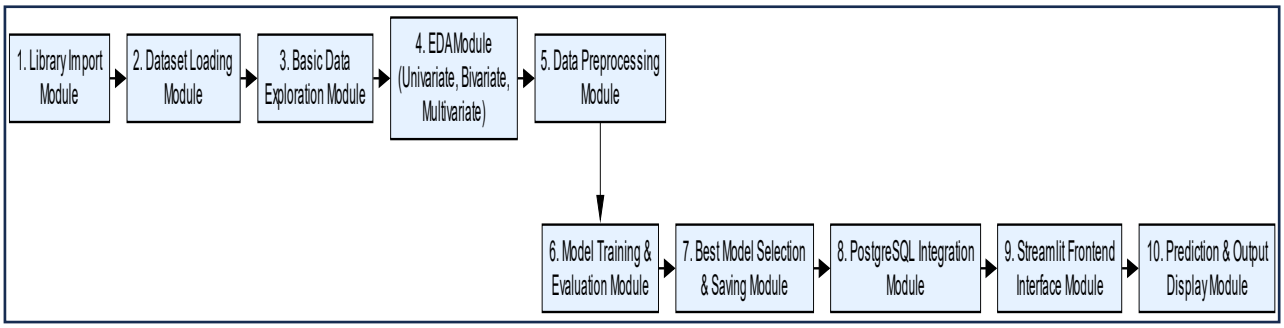


Figure 4: Module Description

1. LIBRARY IMPORT MODULE

This module ensures that all essential Python libraries are loaded before any processing begins. Libraries such as Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, and Streamlit are used for data handling, visualization, modeling, and building the interactive frontend.

2. DATASET LOADING MODULE

The dataset, which includes various customer attributes and previous campaign responses, is loaded into the environment for processing. This module also verifies data format and accessibility.

3. BASIC DATA EXPLORATION MODULE

This module focuses on understanding the structure and quality of the dataset. It involves printing the first few records, checking for data types, missing values, statistical summaries, and identifying duplicates.

KEY ACTIVITIES:

- Viewing sample data
- Getting dataset information and summary statistics
- Checking for unique values in categorical columns
- Identifying and removing duplicate records

4. EXPLORATORY DATA ANALYSIS (EDA) MODULE

EDA is performed to uncover patterns, relationships, and insights within the data.

a) UNIVARIATE ANALYSIS:

Examines each feature individually using histograms and bar charts to observe distributions.

b) BIVARIATE ANALYSIS:

Explores relationships between features using boxplots and correlation heatmaps to detect trends and Outliers.

c) MULTIVARIATE ANALYSIS:

Uses pair plots and visualizes the interaction between multiple variables, especially focusing on how they influence the target variable.

5. DATA PREPROCESSING MODULE

This module cleans and prepares the data for modeling. Tasks include handling missing values, encoding categorical variables, addressing skewed distributions, managing outliers, and scaling features to ensure model performance.

TECHNIQUES:

- Label Encoding and One-Hot Encoding
- StandardScaler or MinMaxScaler for normalization
- Outlier detection and correction
- Skewness correction for numerical stability

6. MODEL TRAINING & EVALUATION MODULE

In this critical module, the cleaned dataset is split into training and testing sets. Multiple machine learning algorithms such as Logistic Regression, Decision Trees, Random Forest, and others are trained and compared using performance metrics like Accuracy, Precision, Recall, and F1-score.

7. BEST MODEL SELECTION & SAVING MODULE

After evaluating all models, the one with the best performance metrics is selected. It is then serialized using a library like pickle for long-term use.

8. POSTGRESQL INTEGRATION MODULE

This module handles the backend integration by storing the saved model into a PostgreSQL database. This allows for efficient model retrieval and forms the basis for a scalable, database-driven architecture.

9. STREAMLIT FRONTEND INTERFACE MODULE

Streamlit is used to build an interactive and user-friendly web interface. Users can input customer details and get real-time predictions directly from the stored model in PostgreSQL.

FEATURES:

- Dropdowns and text inputs for customer data
- Predict button triggering the model
- Real-time display of prediction results

10. PREDICTION & OUTPUT DISPLAY MODULE

The final module takes the input data from the frontend, processes it in the same way as the training data, retrieves the model from PostgreSQL, and generates a prediction. The result (e.g., whether the customer is likely to accept the offer) is displayed clearly to the user.

11. MACHINE LEARNING ALGORITHM MODEL

1. LOGISTIC REGRESSION

Logistic Regression is a statistical algorithm used to predict a binary outcome (like yes or no, 1 or 0) based on input features. It estimates the probability of a certain event happening.

PURPOSE:

Logistic Regression is used to predict the probability of a binary outcome whether a customer will subscribe to a term deposit or not based on various customer features such as age, job, balance, and past campaign responses. It uses a logistic function to model the probability.

Mathematical Function:

$$f(x) = 1 / (1 + e^{-(\beta_0 + \beta_1 x_1 + ... + \beta_n x_n)})$$

ADVANTAGES:

- Highly interpretable and easy to implement.
- Efficient for large datasets.
- Can handle both continuous and categorical variables.

2. RANDOM FOREST CLASSIFIER

Random Forest is an advanced version of Decision Tree that builds many trees (a "forest") and combines their results for better accuracy and less overfitting.

PURPOSE:

Random Forest is an ensemble learning technique that builds multiple decision trees and aggregates their predictions to improve accuracy and stability. It helps classify customers effectively by capturing nonlinear relationships between features.

ADVANTAGES:

- High accuracy and robustness.
- Can handle missing values and categorical variables.
- Less prone to overfitting compared to individual decision trees.

3. SUPPORT VECTOR MACHINE (SVM)

SVM is an algorithm that finds the best boundary (called a hyperplane) to separate classes (like yes vs. no) in the feature space.

PURPOSE:

SVM is used to classify whether a customer will respond positively to a term deposit offer by finding an optimal hyperplane that separates data points of different classes with the maximum margin.

Mathematical Function:

$$f(x) = w^T x + b$$

ADVANTAGES:

- Effective for high-dimensional feature spaces.
- Good generalization with appropriate kernel.
- Robust to outliers due to margin maximization.

4. XGBOOST CLASSIFIER

XGBClassifier is an advanced machine learning algorithm based on gradient boosting. It builds a series of decision trees, where each tree corrects the mistakes of the previous one. It's known for its speed, accuracy, and performance on structured/tabular data.

PURPOSE:

XGBoost (Extreme Gradient Boosting) is a high-performance gradient boosting algorithm that builds a strong classifier by combining multiple weak learners. It's particularly powerful for tabular data like customer banking records.

ADVANTAGES:

- Extremely fast and scalable.
- Handles missing values and unbalanced datasets well.
- Built-in cross-validation and regularization.

5. K-NEAREST NEIGHBORS (KNN)

KNN is a simple algorithm that looks at the ‘K’ nearest data points to make a prediction. It assumes similar things exist close together.

PURPOSE:

KNN classifies customers by comparing their features with the 'k' most similar instances in the training set. It assumes that similar instances exist in close proximity.

ADVANTAGES:

- Simple and easy to understand.
- Works well with multi-class problems.
- No assumptions about data distribution (non-parametric).

CHAPTER 4

SYSTEM TESTING

Testing is a crucial phase in the development of any software system, especially one that involves predictive analytics and handles sensitive user data. In this project, we conducted several types of testing to ensure that the **Retail Banking Prediction System** functions as expected, delivers accurate results, and provides an excellent user experience. The primary testing methods applied are:

1. Functional Testing
2. Model Testing
3. End-to-End Testing
4. Usability Testing

4.1.1. FUNCTIONAL TESTING

PURPOSE:

To verify that all functions within the Retail Banking Prediction System operate according to the specified requirements and perform their intended tasks correctly.

DESCRIPTION:

Functional testing ensured that each module and feature of the system worked as expected. Key features, such as the data input forms, prediction button, and database interaction, were tested to make sure they performed seamlessly when triggered by the user.

IN THIS PROJECT:

- **Input Handling:** Ensured that the user inputs (such as customer data) were accepted correctly by the system.
- **Prediction Execution:** Verified that clicking the "Predict" button triggered the correct machine learning model and displayed the appropriate results.
- **Database Integration:** Checked that data was correctly pushed to PostgreSQL and retrieved without errors.
- **UI Interactions:** Ensured all frontend elements (buttons, dropdowns, text boxes) interacted as expected and provided real-time feedback.

4.1.2. MODEL TESTING

PURPOSE:

To evaluate the performance of each machine learning algorithm used in the project and determine which model provides the best predictions for the given task.

DESCRIPTION:

Model testing was performed to ensure that the five different algorithms (Logistic Regression, Random Forest, SVM, XGBoost, KNN) were properly trained and evaluated based on standard performance metrics. These metrics included accuracy, precision, recall, F1-score, and AUC-ROC. The goal was to determine which model would best predict whether a bank customer would accept a term deposit offer.

IN THIS PROJECT:

- **Model Training:** Ensured that each model was trained on the dataset, with proper hyperparameter tuning.
- **Performance Evaluation:** Measured each model's accuracy, precision, recall, and F1-score using cross-validation and confusion matrices.
- **Model Comparison:** Compared the performance of all five models to identify the most reliable one for deployment.

4.1.3. END-TO-END TESTING

PURPOSE:

To ensure that the entire prediction system, from data input to model prediction and result display, works seamlessly as a unified system.

DESCRIPTION:

End-to-end testing simulated a real-world scenario where a user enters data, triggers the prediction process, and receives the output without any interruptions or failures. This testing was vital in confirming that there were no breaks in the communication between the Streamlit frontend, the machine learning model, and the PostgreSQL backend.

IN THIS PROJECT:

- **User Input Flow:** Tested that user-entered data was successfully transferred to the model for prediction.
- **Model Prediction Flow:** Ensured that the model was invoked correctly, the prediction was made, and results were returned.
- **Backend-Frontend Communication:** Checked if PostgreSQL correctly handled the saving of predictions and model responses.
- **UI Updates:** Verified that the frontend updated properly after receiving the output from the model.

4.1.4. USABILITY TESTING

PURPOSE:

To assess the ease of use, accessibility, and user experience of the system, ensuring that non-technical users can interact with the application effectively.

DESCRIPTION:

Usability testing focused on how user-friendly the Streamlit interface was, especially for individuals without technical expertise. The goal was to identify any potential barriers that could confuse or frustrate users when interacting with the tool.

IN THIS PROJECT:

- **Navigation Simplicity:** Checked if users could intuitively navigate through the input forms and prediction features.
- **Interface Responsiveness:** Tested if the UI responded quickly to user inputs and displayed results in a timely manner.
- **Accessibility Features:** Verified font size, color contrast, and screen reader compatibility to ensure accessibility for users with disabilities.

4.2 SYSTEM IMPLEMENTATION

System implementation is the crucial phase where all the designed components—data, machine learning models, backend storage, and the user interface—come together to form a fully functional application. In this project, we transitioned from analysis and modeling to an interactive prediction system that can be used in real-time.

The implementation integrates a trained machine learning model into a **Streamlit-based web interface**, backed by a **PostgreSQL** database for secure model and data storage. **Visual Studio Code (VS Code)** served as the development environment for writing, testing, and running the application, ensuring a seamless workflow from backend development to frontend deployment.

This phase focuses on making the solution accessible and practical—allowing end users to input customer details and receive predictions instantly, while also ensuring that the system remains efficient, scalable, and secure.

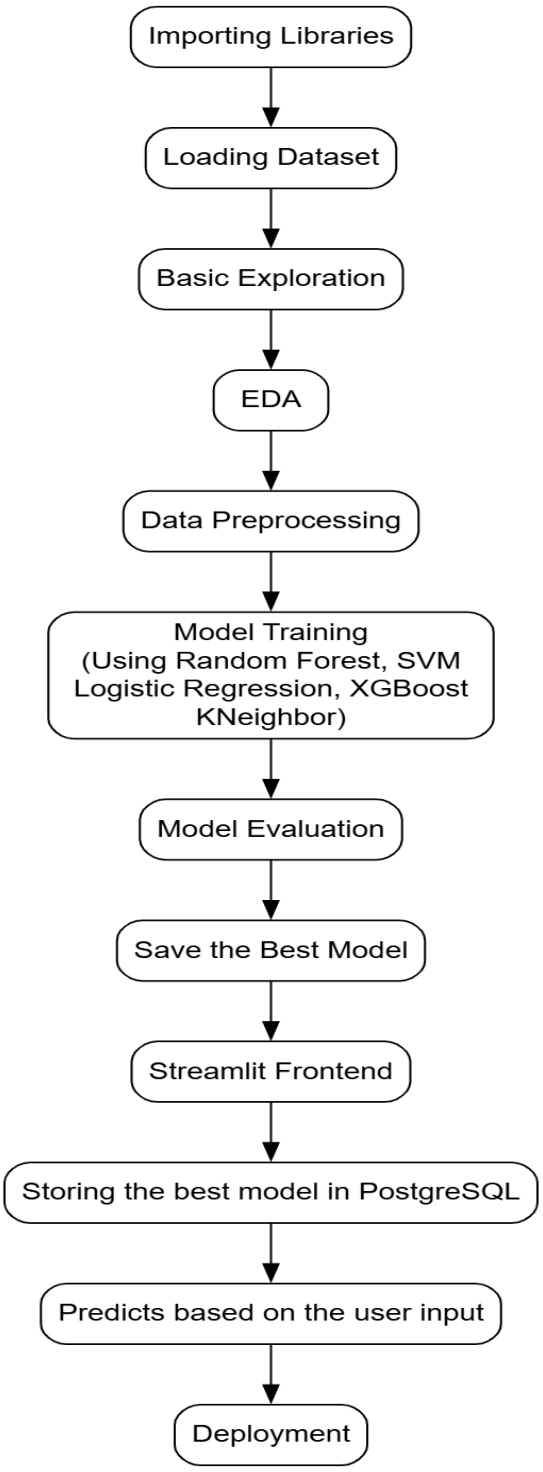


Figure 5: System Implementation

4.2.1 IMPORTING NECESSARY LIBRARIES

The project begins with importing all the essential Python libraries required for data manipulation, visualization, machine learning, and frontend development. Libraries like pandas and numpy are used for data handling, matplotlib and seaborn for visualizations, and sklearn, xgboost, and pickle for building and saving machine learning models. Streamlit is also imported to build the interactive frontend interface.

1. **Data Handling:** pandas, numpy
2. **Visualization:** matplotlib, seaborn
3. **ML Modeling:** sklearn, xgboost
4. **Persistence & Deployment:** pickle
5. **Web UI:** streamlit
6. **Database Connectivity:** psycopg2

4.2.2. LOADING THE DATASET

The dataset, which contains customer information relevant to banking services, is loaded using **pandas**. It includes attributes like age, job type, marital status, balance, loan details, etc., which are crucial in predicting whether a customer will subscribe to a term deposit. This step ensures the dataset is accessible for further analysis and modeling.

4.2.3. BASIC DATA EXPLORATION

This phase involves an initial look at the dataset to understand its structure and contents:

- **Printing the first five rows** helps us get a snapshot of the data.
- **Information about the dataset** shows the number of entries, column names, data types, and non-null values.
- **Statistical summary** (via .describe()) gives insights into the distribution of numerical variables.
- **Checking unique values in categorical columns** helps identify categorical features and their distinct values.
- **Checking duplicate values** ensures data quality by identifying any redundant entries.

4.2.4. EXPLORATORY DATA ANALYSIS (EDA)

EDA allows us to visualize and understand data patterns, trends, and relationships.

a) Univariate Analysis

- **Histograms** for numerical features (e.g., age, balance) reveal distribution and skewness.
- **Bar plots** for categorical features (e.g., job, education, marital status) show frequency distributions.

b) Bivariate Analysis

- **Boxplots** help detect outliers by comparing numerical variables across categorical groups.
- **Correlation heatmap** identifies relationships between numerical features and helps with feature selection.

c) Multivariate Analysis

- **Pair plots** provide a visual overview of multiple feature relationships.
- **Target variable analysis** (e.g., how different features influence the likelihood of deposit subscription) gives valuable predictive insights.

4.2.5. DATA PREPROCESSING

To prepare the data for modeling, several preprocessing techniques are applied:

- **Handling missing values** by imputation or removal ensures clean input.
- **Encoding categorical variables** using Label Encoding or One-Hot Encoding converts textual data into numerical format.
- **Checking and fixing skewness** ensures normally distributed data, improving model performance.
- **Handling outliers** (if needed) to prevent model bias.
- **Feature scaling** (standardization or normalization) ensures uniformity, especially for distance-based algorithms.

4.2.6. MODEL TRAINING & EVALUATION

At this stage, the data is ready to be used for building predictive models.

- **Splitting data** into training and testing sets (80:20) ensures fair evaluation.
- **Training multiple ML models** such as Logistic Regression, Random Forest, SVM, XGBoost, and K-Nearest Neighbors.
- **Evaluating models** using performance metrics like Accuracy, Precision, Recall, and F1-score helps identify the most effective algorithm.
- **Hyperparameter tuning** is done using techniques like RandomizedSearchCV to improve model accuracy.

4.2.7. SAVING THE BEST MODEL

After model comparison, the algorithm with the highest overall performance is chosen. This model is **saved using Pickle**, making it ready for deployment in the prediction system. This saved model is later used for generating real-time predictions in the frontend.

4.2.8. FRONTEND INTEGRATION WITH STREAMLIT

To make the system user-friendly, a web-based interface is built using **Streamlit**. The frontend allows users to input customer data through a form and view instant predictions. The interface also includes result display and basic visual elements to improve usability and interaction.

4.2.9. CONNECTING WITH POSTGRESQL

In this project, **PostgreSQL** is used to:

- Store the **best-performing model** securely.
- Keep a record of user inputs and prediction results. This connection is established using psycopg2 allowing the Streamlit frontend to interact with the database seamlessly.

4.2.10. PREDICTING THE RESULTS

The final stage allows users to make predictions in real-time. Users input the required features via the frontend, which are processed and passed to the saved ML model. The prediction result—whether the customer is likely to accept the term deposit—is then displayed. All predictions can be logged into the PostgreSQL database for future reference or analysis.

CHAPTER 5

CONCLUSION

The *Retail Banking Prediction System* successfully demonstrates how machine learning and data science techniques can be applied to solve real-world banking challenges—specifically, predicting whether a customer is likely to subscribe to a term deposit. From understanding the dataset to deploying an interactive web-based application, each phase of the project contributed to building a reliable and data-driven decision support tool.

Through careful data exploration, preprocessing, and the application of multiple machine learning algorithms, we identified the best-performing model based on key evaluation metrics like accuracy, precision, recall, and F1-score. By integrating this model with a user-friendly Streamlit frontend, we transformed a complex machine learning process into a simple and intuitive experience for end-users.

The use of PostgreSQL for storing the best model and logging prediction data adds an essential layer of scalability and security, ensuring that the system can grow and be maintained efficiently. Tools like VS Code streamlined the development process, making it easier to write, debug, and manage both backend and frontend components.

This project not only highlights the potential of predictive analytics in the financial sector but also reflects a complete machine learning lifecycle from raw data to a deployable product. Moving forward, this system can be expanded by integrating additional features such as customer segmentation, churn prediction, or even personalized marketing strategies, making it a valuable asset for data-driven banking operations.

CHAPTER 6

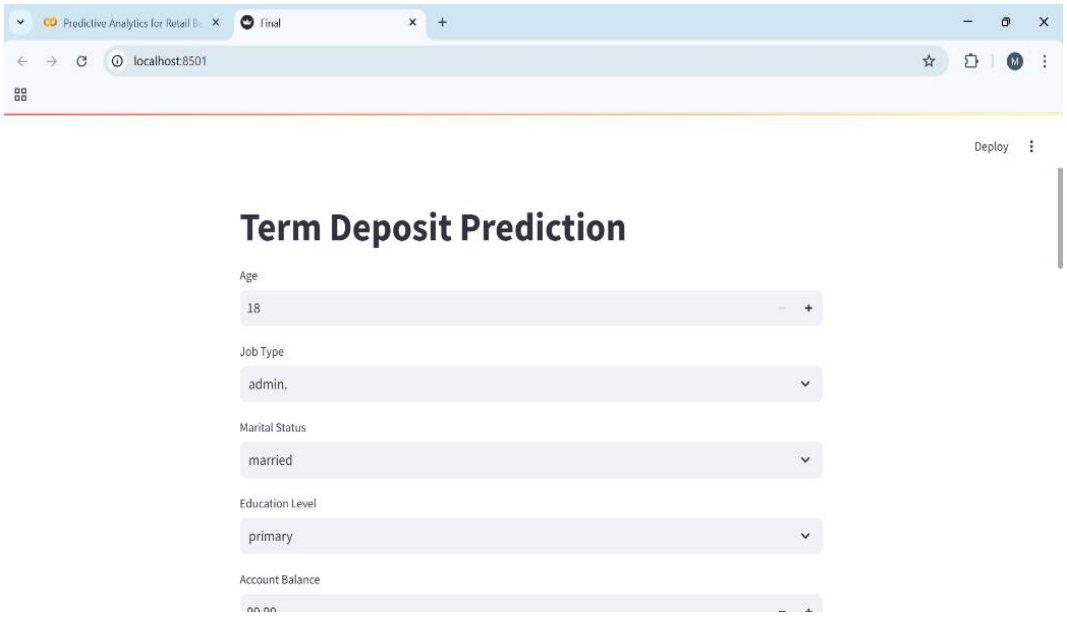
FUTURE ENCHANCEMENT

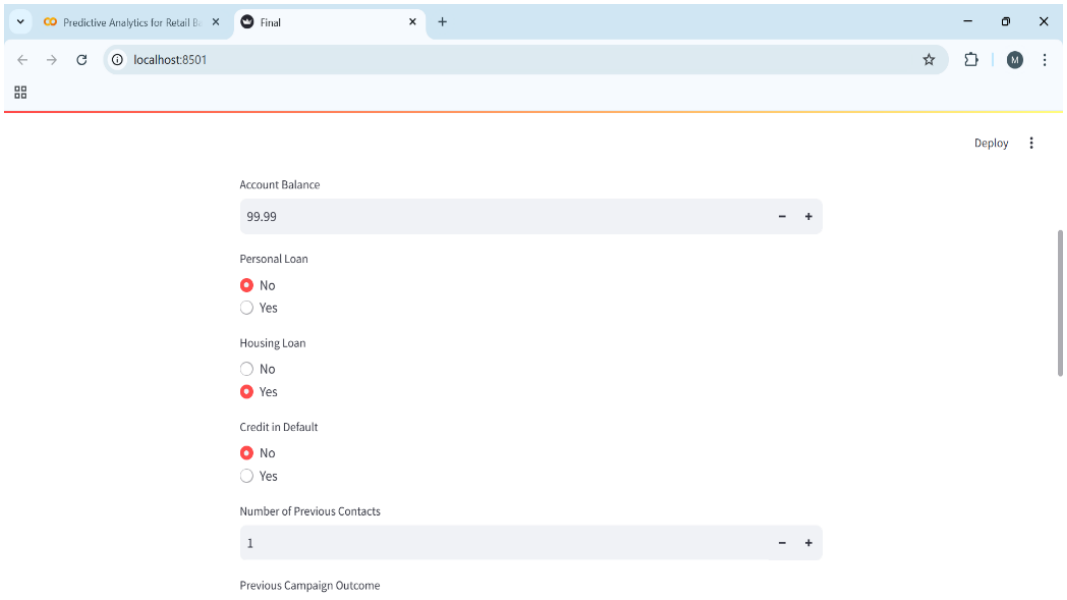
The current implementation of the *Retail Banking Prediction System* delivers strong predictive capabilities using machine learning techniques; however, there are several future enhancements that can be considered to elevate the system to a production-ready solution. One of the primary areas of improvement is the integration of real-time data. Enhancing feature engineering by incorporating behavioural data, transaction frequency, and customer engagement history can lead to deeper insights and improved model accuracy. Security can also be strengthened by adding user authentication and role-based access, ensuring that sensitive data is only accessible to authorized personnel. The development of interactive dashboards using tools like Streamlit or Power BI can support real-time visualization of model outcomes and performance metrics, making the system more user-friendly and insightful for business users. A mobile-responsive design can further enhance accessibility, allowing stakeholders to use the system conveniently across various devices. In the future, adopting an ensemble learning approach—where multiple models are combined for predictions can lead to better generalization and robustness. Furthermore, integrating the system with Customer Relationship Management (CRM) tools or core banking platforms can support seamless deployment in real-world banking environments. These enhancements, when implemented, will significantly expand the system's usability, security, and effectiveness, making it a comprehensive tool for predictive analytics in retail banking.

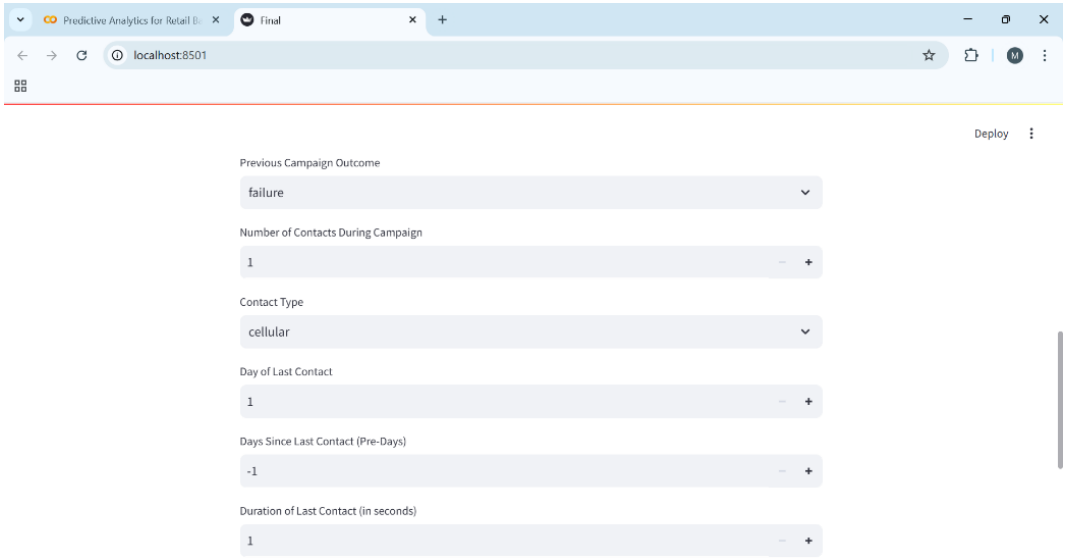
CHAPTER 7

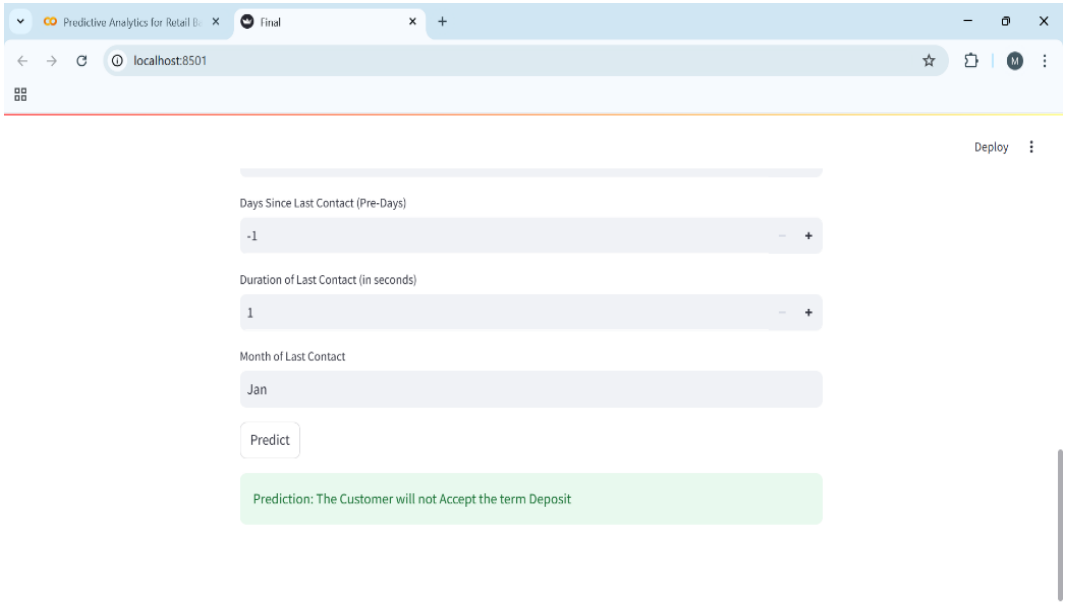
APPENDIX

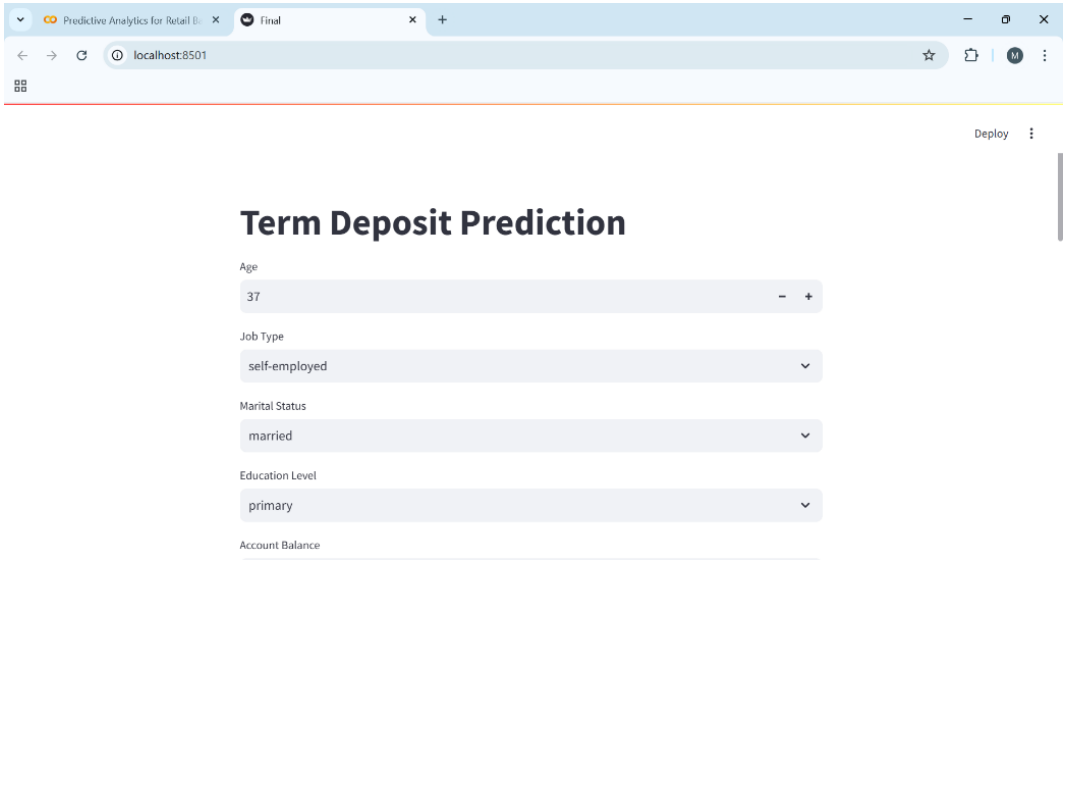
7.1 SCREENSHOTS

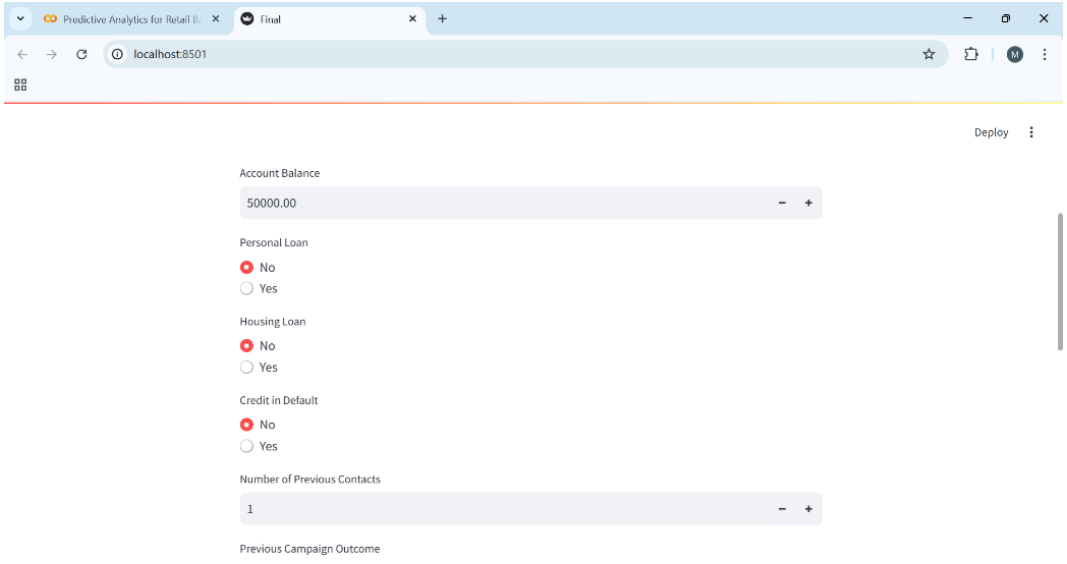


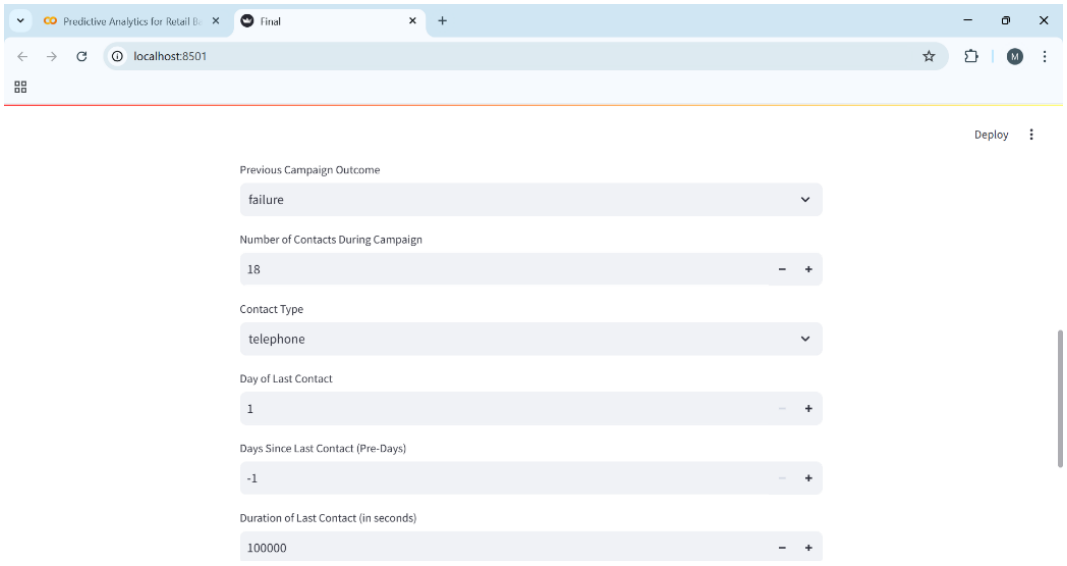


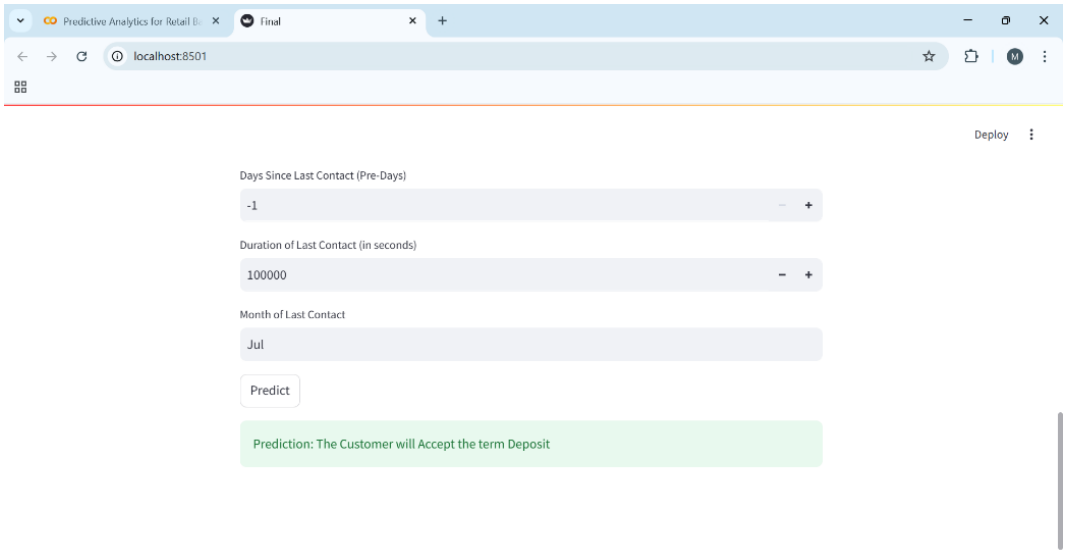

Predictive Analytics for Retail B
Final
localhost:8501
Deploy
Previous Campaign Outcome
failure
Number of Contacts During Campaign
1
Contact Type
cellular
Day of Last Contact
1
Days Since Last Contact (Pre-Days)
-1
Duration of Last Contact (in seconds)
1


Predictive Analytics for Retail B
Final
localhost:8501
Deploy
Days Since Last Contact (Pre-Days)
-1
Duration of Last Contact (in seconds)
1
Month of Last Contact
Jan
Predict
Prediction: The Customer will not Accept the term Deposit


Predictive Analytics for Retail B
Final
localhost:8501
Deploy
Term Deposit Prediction
Age
37
Job Type
self-employed
Marital Status
married
Education Level
primary
Account Balance


Predictive Analytics for Retail B
Final
localhost:8501
Deploy
Account Balance
50000.00
Personal Loan
No
Yes
Housing Loan
No
Yes
Credit in Default
No
Yes
Number of Previous Contacts
1
Previous Campaign Outcome


Predictive Analytics for Retail B
Final
localhost:8501
Deploy
Previous Campaign Outcome
failure
Number of Contacts During Campaign
18
Contact Type
telephone
Day of Last Contact
1
Days Since Last Contact (Pre-Days)
-1
Duration of Last Contact (in seconds)
100000


Predictive Analytics for Retail B
Final
localhost:8501
Deploy
Days Since Last Contact (Pre-Days)
-1
Duration of Last Contact (in seconds)
100000
Month of Last Contact
Jul
Predict
Prediction: The Customer will Accept the term Deposit

7.2 SOURCE CODE:

```
import pandas as pd

import numpy as np

import math

from sklearn.preprocessing import LabelEncoder, StandardScaler,
PowerTransformer

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, auc

from scipy.stats import ttest_ind, chi2_contingency

import pickle

import psycopg2
```

```
df = pd.read_csv("bank.csv")

#Printing the first five rows of the dataset

df.head()

#Information about the Dataset

print("Initial Data Info:")

print(df.info())

#Statistical Summary of the Dataset

print("\nStatistical Summary:")

print(df.describe())

#Checking Duplicate Rows

print(f"Duplicate Rows: {df.duplicated().sum()}")

#Check Unique Values in Categorical Columns

categorical_cols = df.select_dtypes(include=['object']).columns

for col in categorical_cols:

    print(f"Unique values in {col}: {df[col].unique()}")

# Select numerical features

numerical_features=df.select_dtypes(include=['int64',

'float64']).columns
```

```
# Plot histograms for all numerical features
df[numerical_features].hist(figsize=(12, 8), bins=20, color='skyblue',
edgecolor='black')
plt.suptitle("Distribution of Numerical Features")
plt.show()
plt.figure(figsize=(15, 8))
categorical_cols = df.select_dtypes(include=['object']).columns
num_cols = len(categorical_cols)  # Total categorical features
# Determine the number of rows and columns dynamically
rows = (num_cols // 3) + (num_cols % 3 > 0)  # Ensures all subplots
fit
for i, col in enumerate(categorical_cols, 1):
    plt.subplot(rows, 3, i)
        sns.countplot(x=df[col],   hue=df[col],   palette='coolwarm',
legend=False)
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.xticks(rotation=45)
plt.tight_layout()
```

```
plt.show()
plt.figure(figsize=(12, 6))
for i, col in enumerate(numerical_features):
     plt.subplot(2, 4, i+1)  # Adjust based on number of numerical
features
    sns.boxplot(y=df[col], color='lightblue')
    plt.title(f"Boxplot of {col}")
plt.tight_layout()
plt.show()
from sklearn.preprocessing import LabelEncoder
df_encoded = df.copy()
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col])
# Plot heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(df_encoded.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Heatmap")
plt.show()
```

```
sns.pairplot(df, diag_kind='kde', corner=True)

plt.show()

plt.figure(figsize=(6, 4))

sns.countplot(x=df['deposit'], hue=df['deposit'], palette='coolwarm',
legend=False)

plt.title("Distribution of Target Variable")

plt.xlabel("Target Variable")

plt.ylabel("Count")

plt.show()

from scipy.stats import ttest_ind

import seaborn as sns

import matplotlib.pyplot as plt

# Split balance into two groups

balance_accepted = df[df['deposit'] == 'yes']['balance']

balance_rejected = df[df['deposit'] == 'no']['balance']

# Perform Independent T-Test

t_stat, p_value = ttest_ind(balance_accepted, balance_rejected,
equal_var=False)

print(f"T-Test Statistic: {t_stat:.4f}")

print(f"P-Value: {p_value:.4f}")
```

```
# Interpretation
if p_value < 0.05:
    print("Reject the Null Hypothesis: Balance significantly affects deposit acceptance.")
else:
     print("Fail to Reject the Null Hypothesis: No significant difference in balance between groups.")
# Create box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x="deposit", y="balance", hue="deposit", data=df, palette=['red', 'green'], legend=False)
# Labels & Title
plt.xlabel("Deposit Accepted (yes,no)", fontsize=12)
plt.ylabel("Balance", fontsize=12)
plt.title("Balance Distribution for Accepted vs. Rejected Deposits", fontsize=14)
# Show plot
plt.show()
from scipy.stats import chi2_contingency
import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Identify categorical features dynamically

categorical_features                                                =
df.select_dtypes(include=['object']).columns.tolist()

# Store Chi-Square results

chi2_results = {}

# Perform Chi-Square test for each categorical variable

for feature in categorical_features:

    contingency_table = pd.crosstab(df[feature], df['deposit'])

    chi2, p, dof, expected = chi2_contingency(contingency_table)

    chi2_results[feature] = chi2  # Store Chi-Square statistic

# Convert to DataFrame for visualization

chi2_df = pd.DataFrame(list(chi2_results.items()), columns=['Feature',
'Chi-Square Statistic'])

chi2_df     =     chi2_df.sort_values(by='Chi-Square     Statistic',
ascending=False)

# Plot the results

plt.figure(figsize=(10, 6))

sns.barplot(x='Chi-Square  Statistic',  y='Feature',  hue='Feature',
data=chi2_df, dodge=False, legend=False, palette='coolwarm')
```

```
plt.xlabel("Chi-Square Statistic", fontsize=14)
plt.ylabel("Categorical Feature", fontsize=14)
plt.title("Chi-Square Test Results for Categorical Features",
fontsize=16)
plt.show()
#Data Preprocessing
print("\nMissing Values:")
print(df.isnull().sum())
missing_values = df.isnull().sum()
if missing_values.sum() == 0:
    print("No missing values found in the dataset.")
else:
    df.dropna(inplace=True)  # Only drop if missing values exist
    print("Missing values handled.")
# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
print("\nCategorical Columns:", categorical_cols)
print("\nCategorical Variables Before Encoding:")
print(df.head())
# Applying Label Encoding to categorical variables
```

```
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
     df[col] = le.fit_transform(df[col])  # Transforming categories
into numerical values
    label_encoders[col] = le  # Storing encoders for later use
print("\nCategorical Variables After Encoding:")
print(df.head())  # Checking the transformation
# Using IQR (Interquartile Range) to remove outliers
def remove_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
for col in numerical_features:
    df = remove_outliers(df, col)
num_features = len(numerical_features)
rows = math.ceil(num_features / 4)
```

```
# ---- Visualizing Outliers After Handling ----
plt.figure(figsize=(15, rows * 4))
for i, col in enumerate(numerical_features, 1):
    plt.subplot(rows, 4, i)
    sns.boxplot(y=df[col], color='lightgreen')
    plt.title(f"Boxplot of {col} (After Handling)")
plt.tight_layout()
plt.show()
X = df.drop(columns=['deposit']) #Separate features (X) and target (y)
y = df['deposit']
scaler = StandardScaler() # Apply scaling only to features
X_scaled = scaler.fit_transform(X)
print("\nFeature Scaling Applied (Standardization).")
# Define features (X) and target variable (y)
X = df.drop(columns=['deposit'])  # Features
y = df['deposit']  # Target variable
# Split the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
# Standardizing the feature values for models that require scaling
```

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)  # Fit and transform
training data

X_test_scaled = scaler.transform(X_test)  # Transform test data (using
same scaler)

print("Data Splitting Completed: 80% Train - 20% Test")

from sklearn.linear_model import LogisticRegression

from      sklearn.model_selection      import      cross_val_score,
cross_val_predict

from    sklearn.metrics    import    accuracy_score,    recall_score,
roc_auc_score, confusion_matrix, classification_report, roc_curve

import matplotlib.pyplot as plt

# Initialize Logistic Regression

lr = LogisticRegression(max_iter=500)

lr.fit(X_train_scaled, y_train)

lr_acc   =   cross_val_score(lr,   X_train_scaled,   y_train,   cv=3,
scoring='accuracy', n_jobs=-1)

print("\n Logistic Regression Cross-validation Accuracy:", lr_acc)

y_pred_lr = lr.predict(X_test_scaled)

y_proba_lr = lr.predict_proba(X_test_scaled)[:, 1]
```

```
# Accuracy, Recall, and ROC Score

acc_lr = accuracy_score(y_test, y_pred_lr) * 100

rec_lr = recall_score(y_test, y_pred_lr) * 100

roc_lr = roc_auc_score(y_test, y_proba_lr) * 100

print(f"\n Logistic Regression Accuracy: {acc_lr:.2f}% | Recall:
{rec_lr:.2f}% | ROC AUC: {roc_lr:.2f}%")

print("\nConfusion Matrix:") # Confusion Matrix

print(confusion_matrix(y_test, y_pred_lr))

print("\nClassification Report:")  # Classification Report

print(classification_report(y_test, y_pred_lr))

fpr, tpr, _ = roc_curve(y_test, y_proba_lr) # Plot ROC Curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, linewidth=2, label=f"AUC: {roc_lr:.2f}", color='b')

plt.xlabel('False Positive Rate', fontsize=14)

plt.ylabel('True Positive Rate', fontsize=14)

plt.title('ROC Curve: Logistic Regression', fontsize=14)

plt.legend()

plt.show()

from sklearn.model_selection import cross_val_score
```

```
from sklearn.metrics import accuracy_score, recall_score,
roc_auc_score, confusion_matrix, classification_report
from sklearn.metrics import roc_curve
# Initialize Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)
# Cross-validation accuracy
rf_acc = cross_val_score(rf, X_train_scaled, y_train, cv=3,
scoring='accuracy', n_jobs=-1)
print("\n Random Forest Cross-validation Accuracy:", rf_acc)
y_pred_rf = rf.predict(X_test_scaled) # Predict on test data
y_proba_rf = rf.predict_proba(X_test_scaled)[:, 1]
# Accuracy, Recall, and ROC Score
acc_rf = accuracy_score(y_test, y_pred_rf) * 100
rec_rf = recall_score(y_test, y_pred_rf) * 100
roc_rf = roc_auc_score(y_test, y_proba_rf) * 100
print(f"\n Random Forest Accuracy: {acc_rf:.2f}% | Recall:
{rec_rf:.2f}% | ROC AUC: {roc_rf:.2f}%")
print("\nConfusion Matrix:") # Confusion Matrix
print(confusion_matrix(y_test, y_pred_rf))
```

```
print("\nClassification Report:") # Classification Report
print(classification_report(y_test, y_pred_rf))
fpr, tpr, _ = roc_curve(y_test, y_proba_rf) # Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linewidth=2, label=f"AUC: {roc_rf:.2f}", color='b')
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('ROC Curve: Random Forest', fontsize=14)
plt.legend()
plt.show()
from sklearn.svm import SVC
# Initialize SVM with probability=True for ROC Curve
svm = SVC(probability=True)
svm.fit(X_train_scaled, y_train)
# Cross-validation accuracy
svm_acc = cross_val_score(svm, X_train_scaled, y_train, cv=3,
scoring='accuracy', n_jobs=-1)
print("\n SVM Cross-validation Accuracy:", svm_acc)
```

```
y_pred_svm = svm.predict(X_test_scaled) # Predict on test data

y_proba_svm = svm.predict_proba(X_test_scaled)[:, 1]

# Accuracy, Recall, and ROC Score

acc_svm = accuracy_score(y_test, y_pred_svm) * 100

rec_svm = recall_score(y_test, y_pred_svm) * 100

roc_svm = roc_auc_score(y_test, y_proba_svm) * 100

print(f"\n SVM Accuracy: {acc_svm:.2f}% | Recall: {rec_svm:.2f}% |
ROC AUC: {roc_svm:.2f}%")

print("\nConfusion Matrix:") # Confusion Matrix

print(confusion_matrix(y_test, y_pred_svm))

print("\nClassification Report:") # Classification Report

print(classification_report(y_test, y_pred_svm))

fpr, tpr, _ = roc_curve(y_test, y_proba_svm) # Plot ROC Curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, linewidth=2, label=f"AUC: {roc_svm:.2f}",
color='b')

plt.xlabel('False Positive Rate', fontsize=14)

plt.ylabel('True Positive Rate', fontsize=14)

plt.title('ROC Curve: SVM', fontsize=14)

plt.legend()
```

```
plt.show()

from xgboost import XGBClassifier

# Initialize XGBoost

xgb = XGBClassifier(eval_metric='logloss', random_state=42)

xgb.fit(X_train_scaled, y_train)

# Cross-validation accuracy

xgb_acc = cross_val_score(xgb, X_train_scaled, y_train, cv=3,
scoring='accuracy', n_jobs=-1)

print("\n XGBoost Cross-validation Accuracy:", xgb_acc)

y_pred_xgb = xgb.predict(X_test_scaled) # Predict on test data

y_proba_xgb = xgb.predict_proba(X_test_scaled)[:, 1]

# Accuracy, Recall, and ROC Score

acc_xgb = accuracy_score(y_test, y_pred_xgb) * 100

rec_xgb = recall_score(y_test, y_pred_xgb) * 100

roc_xgb = roc_auc_score(y_test, y_proba_xgb) * 100

print(f"\n XGBoost Accuracy: {acc_xgb:.2f}% | Recall: {rec_xgb:.2f}%
| ROC AUC: {roc_xgb:.2f}%")

print("\nConfusion Matrix:") # Confusion Matrix

print(confusion_matrix(y_test, y_pred_xgb))
```

```
print("\nClassification Report:") # Classification Report
print(classification_report(y_test, y_pred_xgb))
fpr, tpr, _ = roc_curve(y_test, y_proba_xgb) # Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linewidth=2, label=f"AUC: {roc_xgb:.2f}",
color='b')
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('ROC Curve: XGBoost', fontsize=14)
plt.legend()
plt.show()
from sklearn.neighbors import KNeighborsClassifier
# Initialize KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
# Cross-validation accuracy
knn_acc = cross_val_score(knn, X_train_scaled, y_train, cv=3,
scoring='accuracy', n_jobs=-1)
print("\n KNN Cross-validation Accuracy:", knn_acc)
```

```
# Predict on test data
y_pred_knn = knn.predict(X_test_scaled)
acc_knn = accuracy_score(y_test, y_pred_knn) * 100 # Accuracy
print(f"\n KNN Accuracy: {acc_knn:.2f}%")
print("\nConfusion Matrix:") # Confusion Matrix
print(confusion_matrix(y_test, y_pred_knn))
# Store accuracy scores in a dictionary
model_results = {
    "Logistic Regression": acc_lr,
    "Random Forest": acc_rf,
    "SVM": acc_svm,
    "XGBoost": acc_xgb,
    "KNN": acc_knn
}
# Find the best model
best_model = max(model_results, key=model_results.get)
# Print the best model
print(f"\n Best Model: {best_model} with Accuracy:
{model_results[best_model]:.2f}%")
```

```
# Save the trained Random Forest model to a binary file
with open('random_forest_model.pkl', 'wb') as f:
    pickle.dump(rf, f)
print(type(rf))
def convert_model_to_binary(file_path):
    with open(file_path, "rb") as file:
        binary_data = file.read()
    return binary_data
# Convert the saved model file to binary
binary_model = convert_model_to_binary("random_forest_model.pkl")
def plot_roc_curves(models, X_test, y_test):
    plt.figure(figsize=(10, 6))
    for name, model in models.items():
        X_test_np = X_test.values
        y_probs = model.predict_proba(X_test_np)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_probs)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.2f})")
     plt.plot([0, 1], [0, 1], 'k--', label="Random Guessing (AUC =
0.50)")
```

```
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve Comparison")
    plt.legend(loc="lower right")
    plt.show()
trained_models = {
    "Logistic Regression": lr,
    "Random Forest": rf,
    "SVM": svm,
    "XGBoost": xgb,
    "KNN": knn
}
plot_roc_curves(trained_models, X_test, y_test)
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
from sklearn.metrics import accuracy_score
# Define the Random Forest model
rf = RandomForestClassifier(random_state=42)
```

```
param_dist = {
    'n_estimators': randint(100, 200),
    'max_depth': randint(10, 30),
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 'log2'],  # Corrected max_features
    'bootstrap': [True, False]
}
random_search_rf = RandomizedSearchCV(estimator=rf,
param_distributions=param_dist,
n_iter=20, cv=3, n_jobs=-1, verbose=2, random_state=42)
random_search_rf.fit(X_train, y_train)
print("Best Parameters for Random Forest:",
random_search_rf.best_params_)
best_rf = random_search_rf.best_estimator_
y_pred_rf = best_rf.predict(X_test)
# Calculate accuracy of the best Random Forest model
rf_accuracy = accuracy_score(y_test, y_pred_rf) * 100
print(f"Random Forest Model Accuracy: {rf_accuracy:.2f}")
```

STREAMLIT CODE:

```
import streamlit as st

import numpy as np

import psycopg2

import pickle

def load_model():

    try:

        conn = psycopg2.connect(

            dbname="banking_predictions",

            user="postgres",

            password="mahe06",

            host="localhost",

            port="5432"

        )

        cur = conn.cursor()

        cur.execute("SELECT model FROM models WHERE model_name = %s",
('Random Forest',))

        model_data = cur.fetchone()

        if model_data:
```

```
            print(f"Model data fetched. Size: {len(model_data[0])}
bytes")
            model = pickle.loads(model_data[0])
            print("Model loaded successfully.")
            cur.close()
            conn.close()
            return model
        else:
            print("Model not found in the database.")
            cur.close()
            conn.close()
            return None
 except psycopg2.Error as e:
        print(f"PostgreSQL error: {e}")
        return None
    except pickle.UnpicklingError as e:
        print(f"Pickle error: {e}")
        return None
    except Exception as e:
        print(f"General error: {e}")
```

```
        return None

st.title("Bank Term Deposit Prediction")

age = st.number_input("Age")

job = st.selectbox("Job Type", ['admin.', 'blue-collar',
'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed',
'services', 'student', 'technician', 'unemployed', 'unknown'])

marital = st.selectbox("Marital Status", ['married', 'single',
'divorced'])

education = st.selectbox("Education Level", ['primary', 'secondary',
'tertiary', 'unknown'])

balance = st.number_input("Account Balance")

loan = st.radio("Personal Loan", ['No', 'Yes'])

housing = st.radio("Housing Loan", ['No', 'Yes'])

default = st.radio("Credit in Default", ['No', 'Yes'])

previous = st.number_input("Number of Previous Contacts", min_value=0,
step=1)

poutcome = st.selectbox("Previous Campaign Outcome", ['failure',

'nonexistent', 'success'])

campaign = st.number_input("Number of Contacts During Campaign",
min_value=1, step=1)

mapping = {'Yes': 1, 'No': 0}
```

```
loan = mapping[loan]

housing = mapping[housing]

default = mapping[default]

job_mapping = {j: i for i, j in enumerate(['admin.', 'blue-collar',
'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed',
'services', 'student', 'technician', 'unemployed', 'unknown'])}

marital_mapping = {'married': 0, 'single': 1, 'divorced': 2}

education_mapping = {'primary': 0, 'secondary': 1, 'tertiary': 2,
'unknown': 3}

poutcome_mapping = {'failure': 0, 'nonexistent': 1, 'success': 2}

job = job_mapping[job]

marital = marital_mapping[marital]

education = education_mapping[education]

poutcome = poutcome_mapping[poutcome]

input_features = np.array([[age, job, marital, education, balance,
loan, housing, default, previous, poutcome, campaign]])

model = load_model()

if st.button("Predict"):

    if model:
```

```
        prediction = model.predict(input_features)
        result = "Accepted" if prediction[0] == 1 else "Not Accepted"
        st.success(f"Prediction: {result}")
    else:
        st.error("Error loading the model.")
```

CHAPTER 8

BIBILOGRAPHY

1. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.
2. Raschka, S., & Mirjalili, V. (2019). *Python Machine Learning* (3rd ed.). Packt Publishing.
3. Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer.
4. McKinney, W. (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (2nd ed.). O'Reilly Media.
5. Probst, P., Wright, M. N., & Boulesteix, A. L. (2019). *Hyperparameter tuning in machine learning*. *Journal of Machine Learning Research*, 20(53), 1-32.
6. Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. *Journal of Machine Learning Research*, 12, 2825–2830.
7. PostgreSQL Global Development Group. (2024). *PostgreSQL Documentation*. Retrieved from: https://www.postgresql.org/docs/
8. Streamlit Inc. (2024). *Streamlit Documentation*. Retrieved from: https://docs.streamlit.io/
9. UCI Machine Learning Repository. (n.d.). *Bank Marketing Data Set*. Retrieved from: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing
10. Seaborn Documentation. (2024). *Statistical Data Visualization in Python*. Retrieved from: https://seaborn.pydata.org/