CUSTOMER CHURN PREDICTION PROJECT

Final Project Report

Submitted by: Rasib Hassan

10Pearls Shine Internship

Date: 15/11/24

1. Data Preprocessing and Exploratory Data Analysis (EDA)

Data Preprocessing

The data preprocessing stage involved preparing the raw data for machine learning by performing a series of steps to clean, transform, and format the dataset.

1. **Data Cleaning**:
   - **Handling Missing Values**: In the dataset, missing values were checked and imputed where necessary.
   - **Encoding Categorical Variables**: Since many machine learning models require numerical inputs, categorical features such as gender, contract type payment method and etc were encoded. One-hot encoding was used categories, while label encoding was applied for EDA ensuring the data was in a machine-readable format.
   - **Feature Scaling**: Numerical features were scaled to normalize their ranges, which is essential for models sensitive to feature magnitudes, such as K-Nearest Neighbors (KNN) and gradient boosting models. StandardScaler was applied to standardize or normalize values between 0 and 1.
2. **Class Balancing**:
   - **Dealing with Imbalanced Target Variable**: Since churn datasets are often imbalanced (e.g., fewer customers churn compared to those who don't), techniques like Synthetic Minority Over-sampling Technique (SMOTE) were applied to balance the classes and prevent the model from being biased toward the majority class.

Exploratory Data Analysis (EDA)

In the EDA phase, the dataset was explored to gain insights into the factors that contribute to customer churn.

1. **Univariate Analysis**:
   - **Target Variable Analysis**: The distribution of the target variable (churn) was analyzed to confirm class imbalance.
   - **Feature Distributions**: Key features like tenure, monthly charges, total charges, and customer demographics (age, gender) were visualized using histograms and density plots to understand their distribution and detect any skewness.
2. **Bivariate Analysis**:
   - **Churn vs. Numerical Features**: Boxplots and scatter plots were used to compare churn status with numerical variables such as monthly charges and tenure. For example, customers with shorter tenures may have a higher churn rate.
   - **Churn vs. Categorical Features**: Churn rates were analyzed across categories such as contract type (monthly, one-year, two-year), payment method, and internet service type. These visualizations helped in understanding which customer segments are more likely to churn.

3. **Correlation Analysis**:

   - **Heatmap of Feature Correlations**: A correlation matrix was generated to identify relationships between features. Highly correlated features (above a certain threshold) were noted for potential removal to reduce redundancy and multicollinearity in the dataset. Correlations between the target variable (churn) and numerical features were assessed to see which factors had the strongest relationships with churn, providing insights for model feature importance.

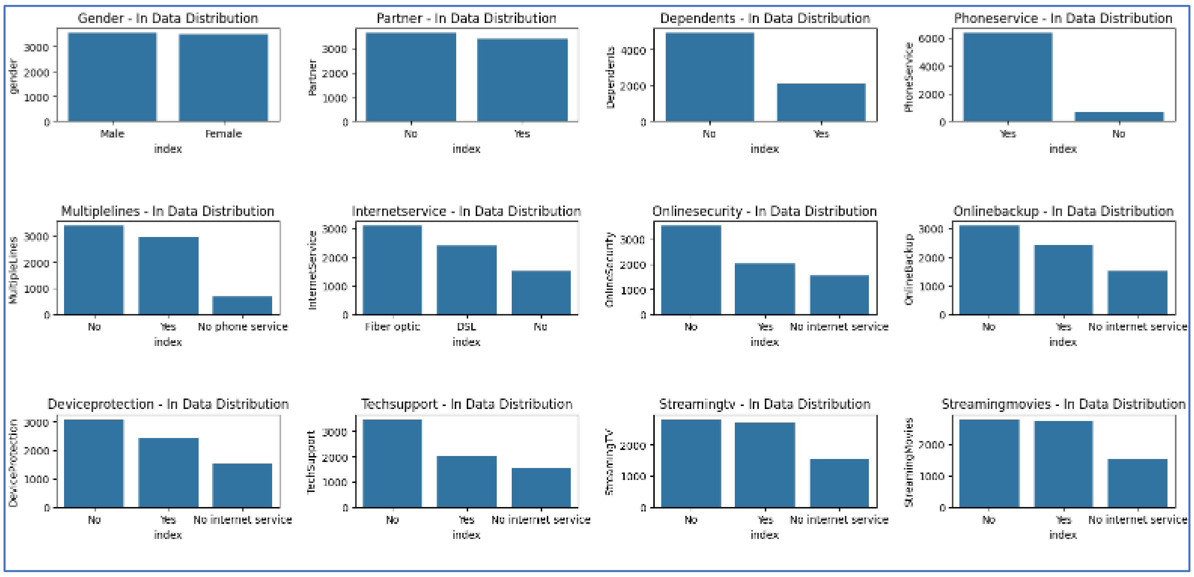
4. **Insights Gained**:

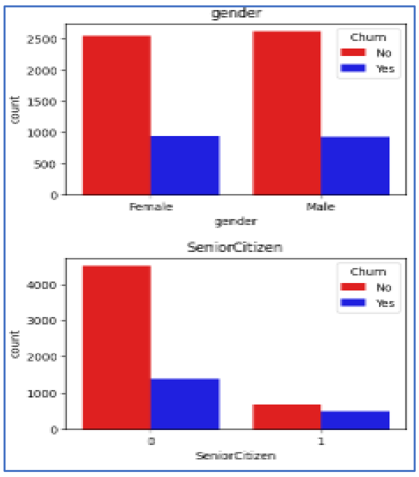
   - Certain categories, such as month-to-month contracts and electronic check payment methods, were identified as having higher churn rates.

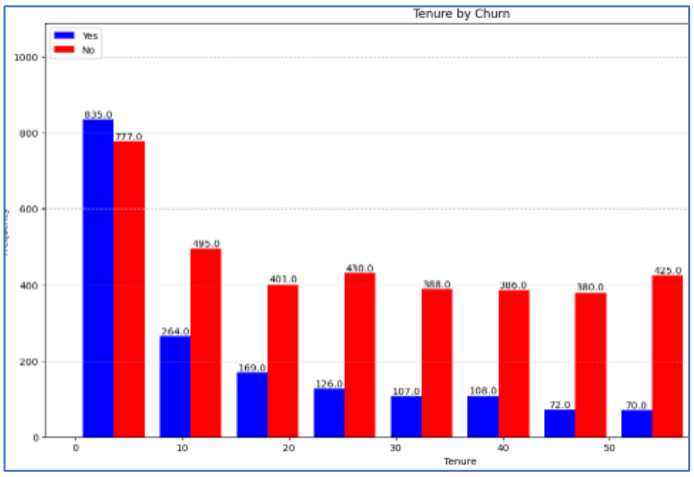
   - Customers with shorter tenures and higher monthly charges were more likely to churn, suggesting that pricing and contract length play crucial roles in customer retention.

Outcome: This data preprocessing and EDA phase resulted in a clean, balanced, and feature-engineered dataset, revealing key patterns and correlations related to customer churn, which informed model selection and training strategies.

Here Are some snapshots of visualizations provided valuable insights:







2. Model Training and Evaluation

Model Selection and Training Process

In the model training stage, several algorithms were evaluated for predicting customer churn. The following models were trained and evaluated:

1. **Random Forest**:
    - **Description**: Random Forest is an ensemble method that builds multiple decision trees and aggregates their predictions, enhancing performance and reducing overfitting.
    - **Results**: Achieved a cross-validation accuracy of **0.842150** and a test accuracy of **0.782115**. The model had a balanced F1 score of **0.621455** and an AUC-ROC of **0.747025**, indicating that it can effectively separate churn and non-churn classes.
2. **Gradient Boosting**:
    - **Description**: Gradient Boosting builds sequential trees, where each tree corrects the errors of the previous ones, making it effective in handling complex patterns.
    - **Results**: The model reached a cross-validation accuracy of **0.828623** and a test accuracy of **0.763940**, with an F1 score of **0.624277** and AUC-ROC of **0.753617**. Although its accuracy was slightly lower, its AUC-ROC was competitive.
3. **Logistic Regression**:
    - **Description**: Logistic Regression is a simple, interpretable model often used as a baseline in classification problems. It provides a probabilistic view of predictions.
    - **Results**: The model performed well, with a cross-validation accuracy of **0.817633** and a test accuracy of **0.779276**. Its F1 score and AUC-ROC were **0.622069** and **0.746790**, respectively, indicating a reasonable balance between recall and precision.
4. **K-Nearest Neighbors (KNN)**:
    - **Description**: KNN is a distance-based model that classifies data points based on their proximity to other data points, which can be sensitive to scaling.
    - **Results**: This model underperformed with a test accuracy of **0.699787** and a low F1 score of **0.520951**, suggesting that KNN may not capture the complexities of the churn dataset effectively.
5. **Decision Tree**:
    - **Description**: Decision Tree is a simple, interpretable model that splits data into nodes based on feature importance. It can easily overfit, but it provides insight into important features.
    - **Results**: With a test accuracy of **0.767921** and an F1 score of **0.608333**, this model showed moderate performance, achieving an AUC-ROC of **0.739054**.

6. **XGBoost**:

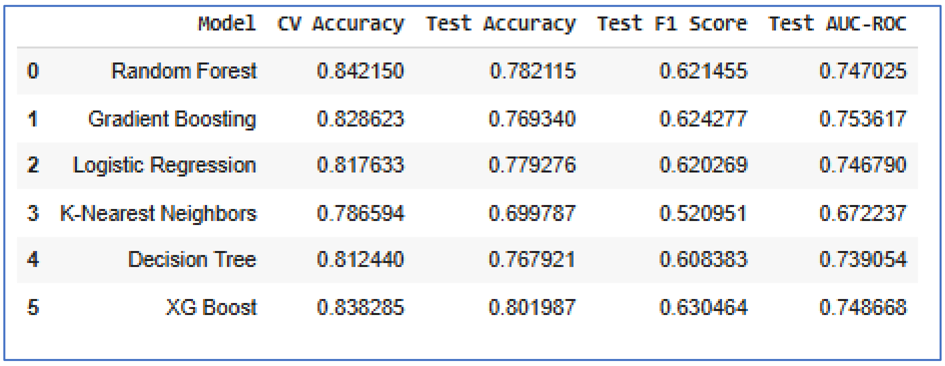
   - **Description**: XGBoost is an advanced gradient-boosting algorithm known for its speed and performance, using regularization to control overfitting.
   - **Results**: XGBoost emerged as the top-performing model, with a cross-validation accuracy of **0.838285** and the highest test accuracy at **0.801987**. Its F1 score was **0.630464**, and the AUC-ROC of **0.748668** highlighted its effectiveness in identifying churn customers.

Model Evaluation and Selection

The models were evaluated based on several metrics:

- **Cross-Validation Accuracy**: Used to assess model stability across multiple training sets.
- **Test Accuracy**: Direct measure of model accuracy on the unseen test data.
- **F1 Score**: Provides a balance between precision and recall, critical in imbalanced datasets like churn prediction.
- **AUC-ROC**: Evaluates the model's ability to distinguish between churn and non-churn classes.

Outcome: XGBoost was chosen as the final model for deployment due to its superior test accuracy and AUC-ROC. Its robustness in handling class imbalance and feature interactions made it the best choice for this project.



	Model	CV Accuracy	Test Accuracy	Test F1 Score	Test AUC-ROC
0	Random Forest	0.842150	0.782115	0.621455	0.747025
1	Gradient Boosting	0.828623	0.769340	0.624277	0.753617
2	Logistic Regression	0.817633	0.779276	0.620269	0.746790
3	K-Nearest Neighbors	0.786594	0.699787	0.520951	0.672237
4	Decision Tree	0.812440	0.767921	0.608383	0.739054
5	XG Boost	0.838285	0.801987	0.630464	0.748668

3. Designing a MySQL Database

This module aimed to design a database in MySQL for analyzing customer churn at a telecom company. By structuring data into related tables and performing advanced SQL analyses, the module identifies patterns that can inform retention strategies.

Objectives

- **Database Schema Design**: Create tables for customers, billing, services, and churn predictions with primary and foreign keys.
- **Data Ingestion**: Import preprocessed data, ensuring integrity with MySQL Workbench.
- **Data Querying & Reporting**: Generate insights on churn patterns by querying based on contract type, tenure, and billing details.

3. Key Queries and Analysis

- **Churn by Contract Type**: Identified that month-to-month contracts had a higher churn rate.
- **Churn by Tenure Range**: Customers with shorter tenure (less than a year) showed higher churn rates.
- **Churn by Monthly Charges**: Higher monthly charges correlated with increased churn, suggesting pricing sensitivity.

4. Insights and Findings

Here's a summary of the insights provided by each query:

1. **Churn Count by Status**

```
SELECT churn, COUNT(customerID) AS CustomerCount

FROM telco_churn.Churnpredictions

GROUP BY churn;
```

This query shows the total count of customers who have churned versus those who haven't. It provides a high-level view of the overall churn rate, which is useful for understanding the scale of customer retention challenges.

2. **Churn by Contract Type**

```
SELECT b.Contract, cp.churn, COUNT(cp.customerID) AS CustomerCount

FROM telco_churn.Churnpredictions AS cp

JOIN telco_churn.Billing AS b ON cp.customerID = b.customerID

GROUP BY b.Contract, cp.churn

ORDER BY b.Contract, cp.churn;
```

This query examines the relationship between contract type and churn. It reveals which contract types (e.g., month-to-month, one-year, or two-year) have higher churn rates, indicating that shorter contracts may lead to higher churn and that incentives for longer contracts could improve retention.

3. **Customer Count by Tenure Segment**

```
SELECT
  CASE
    WHEN tenure < 12 THEN 'Short-term'
    WHEN tenure BETWEEN 12 AND 24 THEN 'Mid-term'
    ELSE 'Long-term'
  END AS TenureSegment,
  COUNT(c.customerID) AS CustomerCount
FROM telco_churn.Customers AS c
GROUP BY TenureSegment;
```

This query groups customers by tenure (short-term, mid-term, and long-term) to identify how many customers fall into each segment. It helps to see if customers with shorter tenures are more common, which may indicate a need for stronger engagement for new customers.

4. **Churn by Contract Type**

```
SELECT b.Contract, cp.churn, COUNT(cp.customerID) AS CustomerCount
FROM telco_churn.Churnpredictions AS cp
JOIN telco_churn.Billing AS b ON cp.customerID = b.customerID
GROUP BY b.Contract, cp.churn
ORDER BY b.Contract, cp.churn;
```

This query examines the relationship between contract type and churn. It reveals which contract types (e.g., month-to-month, one-year, or two-year) have higher churn rates, indicating that shorter contracts may lead to higher churn and that incentives for longer contracts could improve retention.

5. **Churn by Internet Service Type**

```
SELECT
  s.InternetService, cp.churn, COUNT(cp.customerID) AS CustomerCount
FROM telco_churn.Churnpredictions AS cp
JOIN telco_churn.Services AS s ON cp.customerID = s.customerID
GROUP BY s.InternetService, cp.churn
ORDER BY s.InternetService, cp.churn;
```

This query evaluates churn rates based on the type of internet service customers have (e.g., DSL, Fiber optic, or None). It provides insight into whether certain service types are associated with higher churn, suggesting areas where service quality improvements might reduce churn.

4. API Development with Flask and Testing with Postman

the process of developing a customer churn prediction API using Flask, a lightweight web framework for Python. The API receives customer data, processes it, and returns a prediction on whether the customer is likely to churn. We then tested the API using Postman, a popular tool for API development and testing.

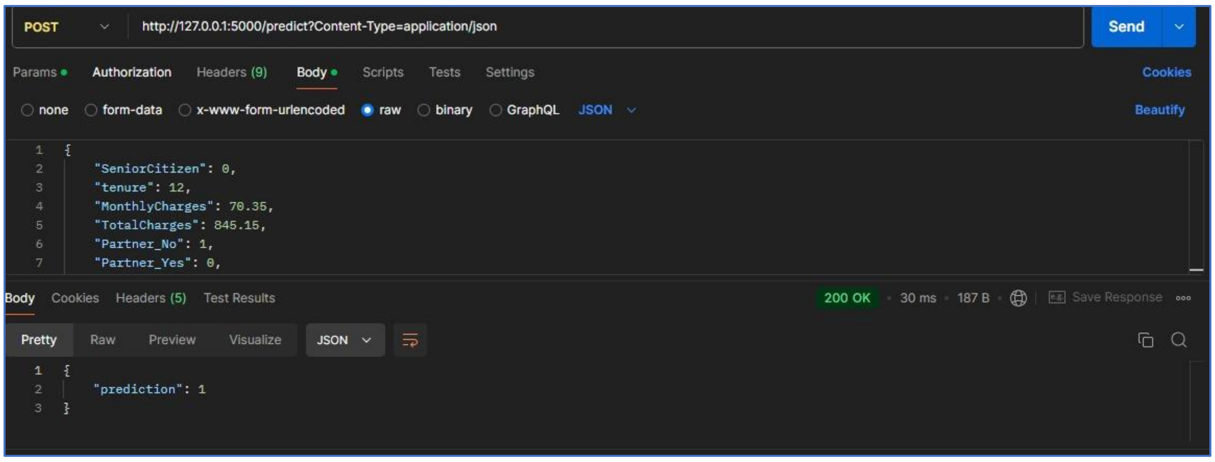
developed a customer churn prediction API using Flask. This API receives customer data in JSON format, processes it, and returns a prediction (1 for churn, 0 for no churn) based on a pre-trained machine learning model.

Development Steps:

1. **Flask Setup**: Created a /predict endpoint.
2. **Model Loading**: Loaded the trained model with pickle.
3. **Data Handling**: Processed JSON input, ensuring all required columns matched the model's expected format.
4. **Prediction and Response**: Made predictions and returned them as JSON.

Testing with Postman:

- In Postman, we set up a POST request to http://127.0.0.1:5000/predict, added sample JSON data in the body, and verified correct predictions in the response.



```
http://127.0.0.1:5000/predict?Content-Type=application/json
```

```
{
    "SeniorCitizen": 0,
    "tenure": 12,
    "MonthlyCharges": 70.35,
    "TotalCharges": 845.15,
    "Partner_No": 1,
    "Partner_Yes": 0,
```

```
200 OK · 30 ms · 187 B
```

```
{
    "prediction": 1
}
```

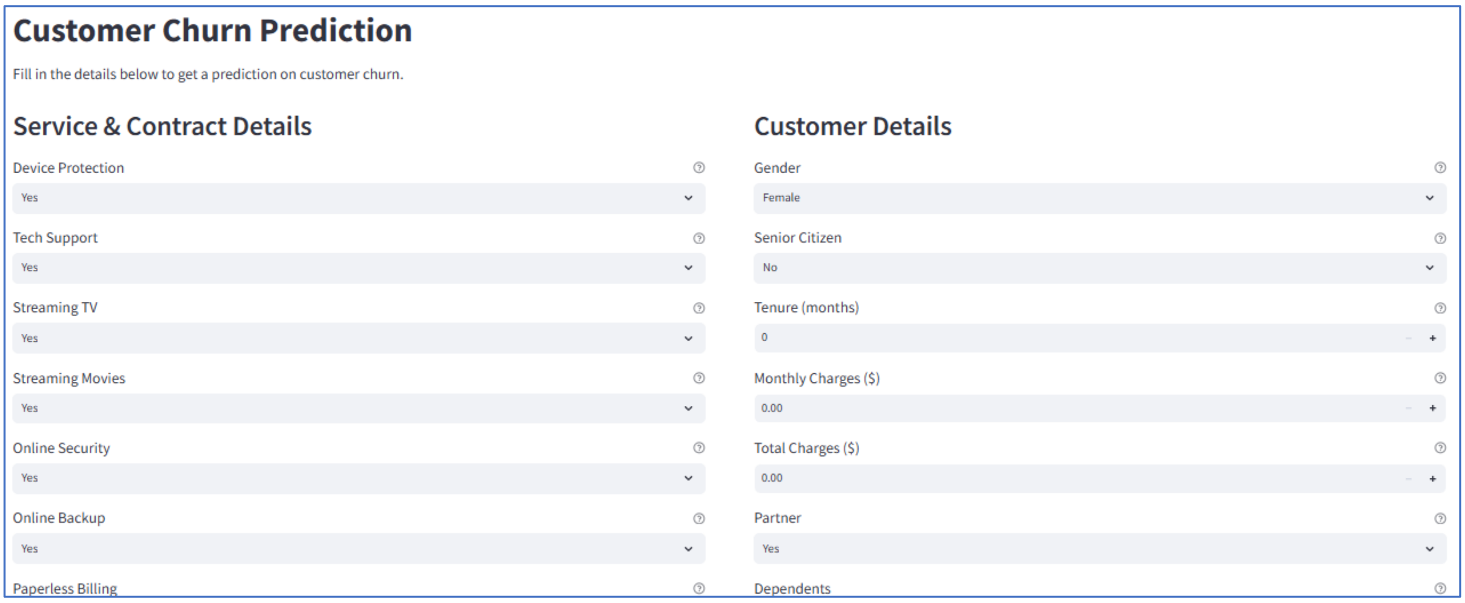
5. Streamlit UI Development

developed a user-friendly interface using Streamlit to predict customer churn based on input data. The interface allows users to enter customer details like tenure, monthly charges, contract type, and various service subscriptions.

Key Features:

1. **Two-Column Layout**: Divided inputs across two columns for clarity and ease of use.
2. **Interactive Input Fields**: Users can enter numerical data, select options for services, and specify customer attributes using dropdowns and number inputs.
3. **Prediction Button**: Once all inputs are filled, users click a "Predict" button to send data to the Flask API and display the churn prediction directly on the page.

This setup provides an intuitive and accessible experience, enabling seamless interaction with the churn prediction model. The attached screenshots illustrate the clean design and simple layout.



Customer Churn Prediction

Fill in the details below to get a prediction on customer churn.

Service & Contract Details

Device Protection: Yes
Tech Support: Yes
Streaming TV: Yes
Streaming Movies: Yes
Online Security: Yes
Online Backup: Yes
Paperless Billing

Customer Details

Gender: Female
Senior Citizen: No
Tenure (months): 0
Monthly Charges ($): 0.00
Total Charges ($): 0.00
Partner: Yes
Dependents

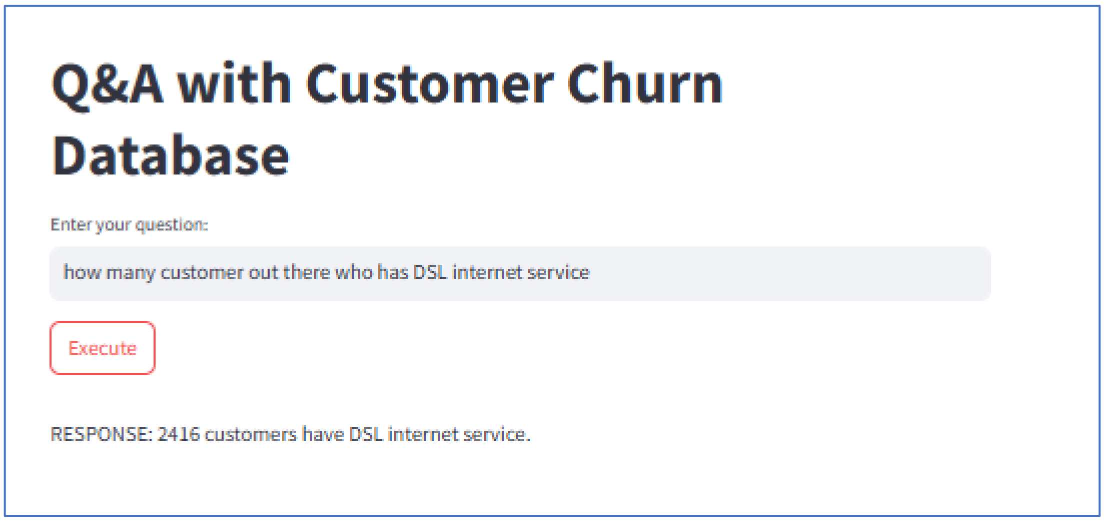
6. RAG Implementation with Gemini LLM for Database Queries

This setup uses Retrieval-Augmented Generation (RAG) with Gemini LLM to let users ask text-based questions about a database, receiving clear, text responses.

How It Works:

1. **User Query:** The user asks a question in natural language.
2. **LLM Interpretation:** Gemini LLM translates this question into an SQL query.
3. **SQL Execution:** The query is run on the backend database.
4. **Answer Generation:** The LLM formats the database response into a readable answer for the user.

This approach makes database insights accessible to non-technical users by transforming text questions into SQL-based answers.


Q&A with Customer Churn Database

Enter your question:

how many customer out there who has DSL internet service

RESPONSE: 2416 customers have DSL internet service.

Conclusion

This project successfully developed a robust customer churn prediction model using machine learning techniques, providing valuable insights into factors that contribute to churn in the telecom industry. By applying thorough data preprocessing and exploratory data analysis (EDA), we identified key patterns and trends, which informed the model training process. Multiple models were evaluated, with XGBoost emerging as the top performer due to its high accuracy and ability to handle class imbalance effectively.

In addition to the predictive model, a comprehensive system was designed, including a MySQL relational database for data storage, a Flask API for real-time predictions, and a Streamlit user interface for accessibility. Furthermore, the integration of Retrieval-Augmented Generation (RAG) with the Gemini LLM enabled intuitive, natural language interactions with the database, making complex insights accessible to non-technical users.