Technical Report: Final Project DS 5110

Introduction to Data Management and Processing

Team Members:
Naga Pavithra Lagisetty, Sunkara Vinisha, Roshitha Tiruveedhula
Khoury College of Computer Sciences
Data Science Program

December 10, 2024

Contents

1 Introduction

Blood donation is a critical component of modern healthcare, with applications including emergency treatment, surgical procedures, chronic illness management, and disaster relief. The increasing demand for blood, combined with the complexities of guaranteeing compatibility and timely delivery, highlights the importance of an effective, scalable, and dependable blood donation management system. Furthermore, handling enormous amounts of donor data, tracking inventory across numerous blood banks, and assuring compliance with safety and regulatory standards are all substantial operational problems.

To solve these difficulties, the project proposes creating a complete Donor and Blood Bank Management System. The system's goal is to improve the way blood donations are managed by integrating cutting-edge technology including advanced data management, and user-friendly interfaces. The main aim of this program is the "Find Nearby Donors" tool, and also "request Blood Tool" which allows hospitals, blood banks, and individuals to rapidly discover and connect with suitable donors and emergecy requests in their area. This feature is especially important in critical situations like accidents, natural disasters, or emergency medical operations, where every second counts.

In addition to real-time donor discovery, the system is intended to maintain an accurate and secure donor database, check the donor eligibility, streamline blood inventory management, and automate regular procedures to minimize the workload on healthcare workers. The integration of user-friendly dashboards, alerting systems, and analytics tools ensures that stakeholders make informed decisions and effectively allocate resources. The goal of the suggested method is to establish a strong and resilient blood donation network by reducing the distance between blood donors and recipients. By raising donor knowledge and participation, this project hopes to promote a voluntary blood donation culture in addition to increasing the operational effectiveness of blood banks. In the end, by guaranteeing that life-saving blood is consistently available, accessible, and given fairly, this program advances the larger goal of saving lives.

2 Literature Review

Recent research in blood donation systems and related technologies has shown significant advancements in various aspects of blood bank management and donor classification. Jaiswal et al. (2022) presented a "Blood Donation System" at the 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), likely focusing on developing a comprehensive system to manage the blood donation process. This system potentially incorporates features such as donor registration, inventory management, and distribution tracking, aiming to streamline the entire blood donation workflow.

In a similar vein, Kaur et al. (2022) introduced a "Web-based Blood Bank System for Managing Records of Donors and Receipts" at the International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES). Their work emphasizes the importance of digital record-keeping in blood donation management. This web-based system likely offers a user-friendly interface for both donors and healthcare professionals to access and manage blood donation information, potentially improving the efficiency and accuracy of record-keeping in blood banks.

Zulfikar et al. (2018) explored an innovative approach to donor classification in their paper ”An Approach to Classify Eligibility Blood Donors Using Decision Tree and Naive Bayes Classifier,” presented at the 6th International Conference on Cyber and IT Service Management (CITSM). Their research employed machine learning techniques to automate the process of determining donor eligibility. This application of artificial intelligence in blood donation could significantly streamline the donation process, reduce human error, and potentially increase the accuracy of donor eligibility assessments.

Addressing the critical issues of data security and operational efficiency, Sandaruwan et al. (2020) presented research ”Towards an Efficient and Secure Blood Bank Management System” at the IEEE 8th R10 Humanitarian Technology Conference (R10-HTC). Their study likely explored advanced security measures for protecting sensitive donor information and optimizing blood bank operations. This research highlights the growing importance of cybersecurity in healthcare information systems, particularly in the context of blood donation management.

While not directly related to blood donation, the work by Qiao et al. (2017) on ”Bird species recognition based on SVM classifier and decision tree” demonstrates the broader application of machine learning techniques in classification tasks. This research, presented at the First International Conference on Electronics Instrumentation & Information Systems (EIIS), could potentially inform future developments in blood type classification or donor matching algorithms within the blood donation field.

3 Methodology

In order to find significant patterns and trends, this project applies a rigorous quantitative research technique that emphasizes the methodical collecting, processing, and analysis of numerical data.

This methodology’s main goals are to use data-driven methods to better understand personal health behaviors, find relationships between important health indicators, and spot patterns between donor eligibility. In addition to making inferences about present health conditions, the research hopes to create predictive models that use past data to check for eligibility of being a donor.

3.1 Data Collection

The data used in this analysis was obtained from Kaggle, a popular platform for data science and machine learning datasets. The specific dataset used is called ”transfusion.csv”, which contains information related to blood donations.

3.1.1 Donor Data

Important information including name, age, gender, contact details, blood type, and the most recent donation date are all included in donor data. Through a systematic registration process, this data is gathered using manual forms. Strict data validation is enforced by the system upon registration to guarantee accuracy and consistency. For instance, contact information formats are vetted to avoid inaccuracies and blood group data are cross-checked against specified categories. Additionally, duplicate records are found and removed using unique identifiers like phone numbers or email addresses. In an emergency, it can be simpler to find nearby donors because to sophisticated features like GPS tagging, which can help link donors to particular locales.

3.1.2 Blood Bank Data

Blood banks provide vital information, such as their name, address, geolocation, contact information, and blood inventory status. The inventory is organized by blood type, quantity, and expiration date, with regular updates to assure real-time accuracy. This information is critical for efficient resource allocation since it enables the system to deliver accurate inventory summaries and notifications for low stock or expiring units.

3.1.3 Blood Request Data

A blood request system is implemented where users submit urgent requests for specific blood groups. These requests would be displayed on a real-time dashboard, alerting potential donors and blood banks to critical needs. This system would streamline the process of matching urgent blood requirements with available donors or supplies.

3.2 Data Preprocessing

The data cleansing procedure is critical to ensuring that the system runs smoothly, with accurate and reliable data guiding all decisions made inside the Donor and Blood Bank Management System. By eliminating inconsistencies, decreasing redundancy, and correcting errors, this comprehensive cleaning approach improves the dataset's reliability and integrity.

Missing data in blood bank records may include inventory details, which can be corrected by validating with internal sources or cross-referencing with the most recent system changes. Similarly, blood request data is frequently missing or erroneous. The system detects missing fields, such as contact information or urgency level, and marks them for verification or completion.

3.2.1 Handling Missing Values

Missing values are a prevalent problem in data management, particularly when working with user-generated input or automated data gathering from several sources. In the context of the system, missing data in donor records (such as contact information or blood type) are a top priority to repair because they can impede communication and correct donor matching to requests. These gaps are filled using a combination of external references (e.g., contacting the donor directly or querying more data) and highlighting the missing fields for user action.

3.2.2 Identifying and Removing Duplicate Entries

Duplicates can have a substantial impact on the system's accuracy and performance, particularly when matching donors with blood requests. The system employs a variety of ways to detect and delete duplicate records. Unique identifiers, such as phone numbers, email addresses, or national ID numbers, are used to detect redundancy. In circumstances when exact matches are not detected, the system uses geolocation data or fuzzy matching algorithms to find near-duplicate entries, in which the same person registers with slightly different information (for example, various spellings of a name or address).By carefully identifying and deleting duplicates, the system eliminates redundancy in donor information, ensuring that each donor's contributions are only counted once and that blood requests are not handled numerous times for the same donor.

3.2.3 Detecting and Handling Outliers

Outlier detection is conducted using two methods: Z-score and Interquartile Range (IQR). The Z-score method identifies values that are more than 3 standard deviations away from the mean, while the IQR method detects values below Q1 - 1.5IQR or above Q3 + 1.5IQR. Although the code includes functions to replace outliers with threshold values, these are commented out in the final implementation. A new feature, ”donation-duration,” is created by subtracting the last donation month from the first donation month. The target variable ”is-donor” is separated from the feature set. To handle potential numerical instability, a small constant (1e-8) is added to all feature values. Finally, the features are transformed using PowerTransformer with the Yeo-Johnson method, which helps to make the data more Gaussian-like and mitigate the effect of outliers. The transformed features are then converted back to a pandas DataFrame, completing the preprocessing steps for subsequent model training and evaluation.

3.3 Analysis Techniques

The code employs several visualization techniques to analyze the blood donation dataset. Histograms and box plots are generated for each feature to understand their distributions and identify potential outliers. A scatter plot is created to visualize the relationship between donation frequency and total blood donated, with points color-coded by donor status. Another scatter plot shows the same relationship without color coding. A box plot compares the total blood donated between donors and non-donors. These visualizations help in understanding feature distributions, relationships between variables, and differences between donor groups. Additionally, the code includes functions for outlier detection using Z-score and Interquartile Range methods, which are applied to each feature and the results are printed. These visualization and outlier detection techniques provide a comprehensive exploratory data analysis of the blood donation dataset.

4 Results

The provided code implements a comprehensive machine learning pipeline for blood donation prediction using the ”transfusion.csv” dataset. The process begins with data preprocessing, including renaming columns and removing duplicates. Exploratory data analysis is conducted using visualizations such as histograms, box plots, and scatter plots to understand feature distributions and relationships. The code then performs outlier detection using Z-score and Interquartile Range (IQR) methods, and creates a new feature called ”donation-duration.” For model preparation, the target variable ”is-donor” is separated from the feature set, and feature transformation is applied using PowerTransformer. The pipeline includes model selection, implementing various classifiers such as Logistic Regression, Random Forest, and CatBoost, which are evaluated using cross-validation and ROC AUC score. Hyperparameter tuning is performed using Optuna for optimizing CatBoost parameters. Finally, the selected model (Logistic Regression in this case) is trained on the preprocessed data, and predictions are generated to calculate the final AUC score on the test set.

4.1 Visualization Plots

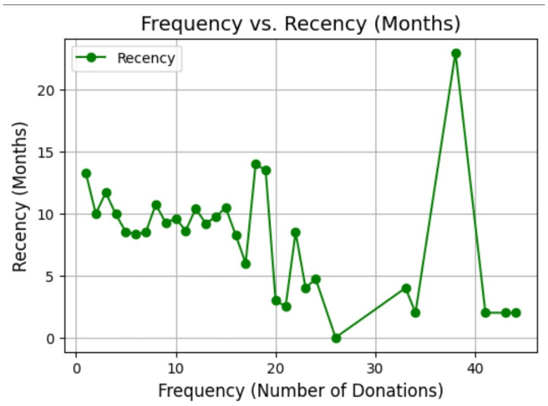


Figure 1: Line Graph

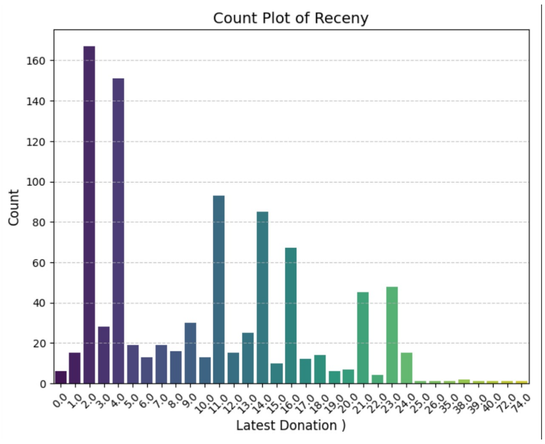


Figure 2: Count Plot

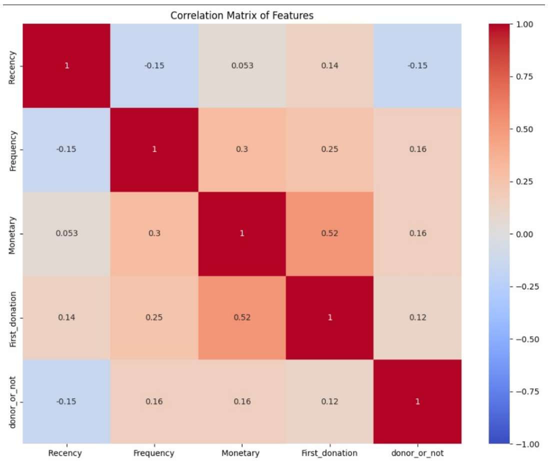


Figure 3: Correlation Matrix

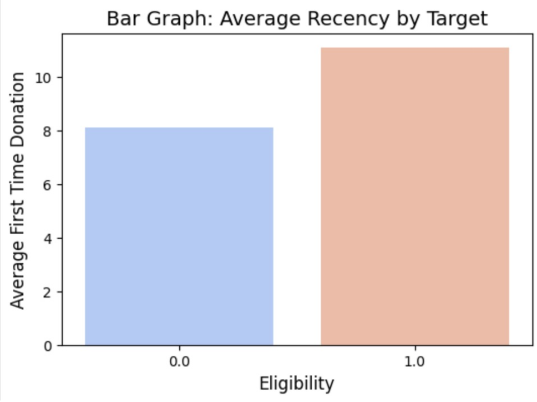
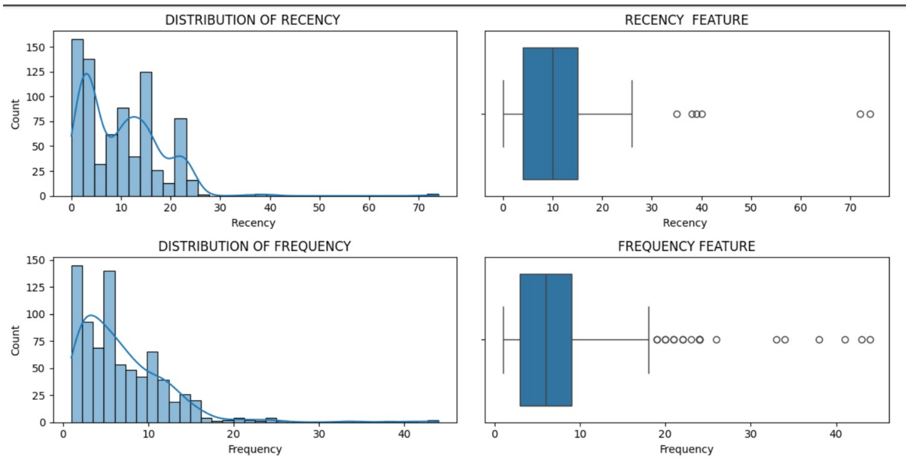


Figure 4: Box Plot



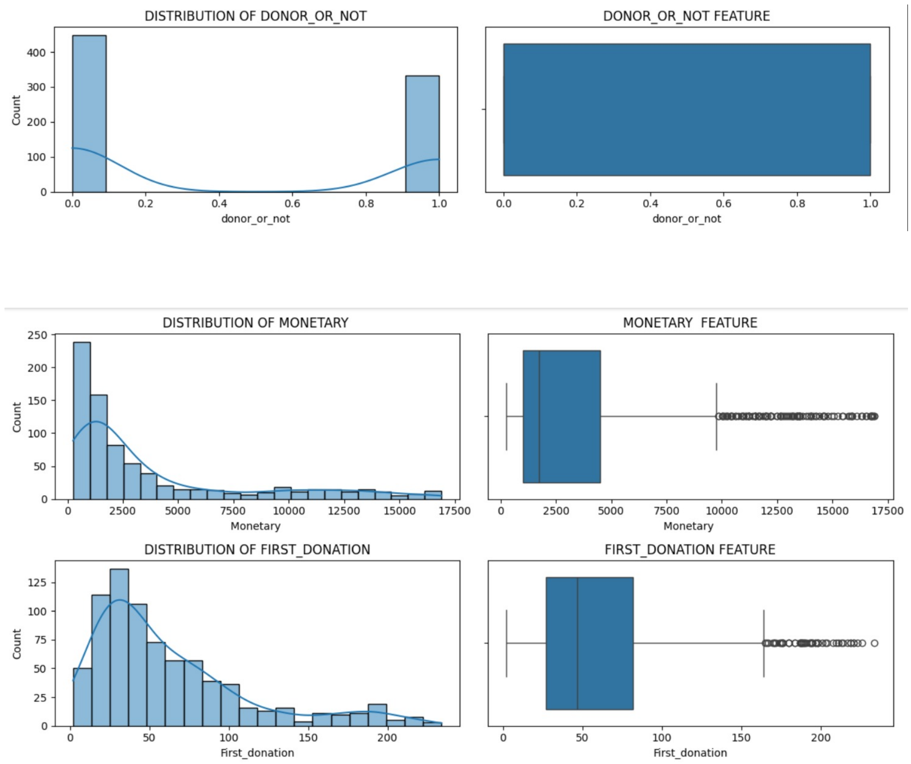


Figure 5: Histograms and Box Plots

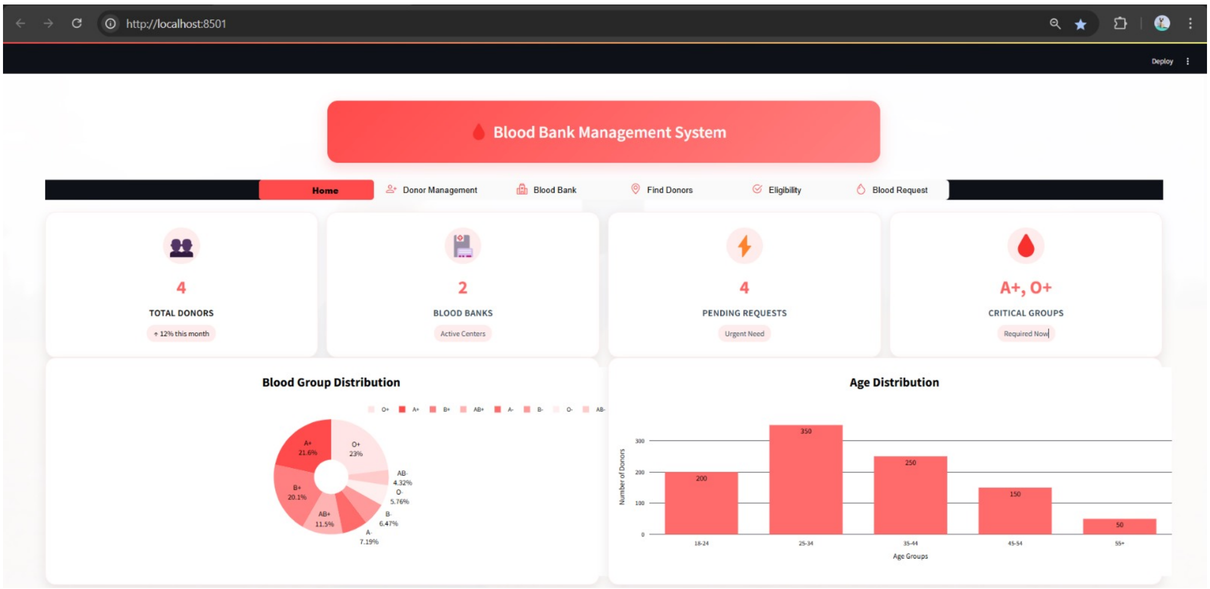
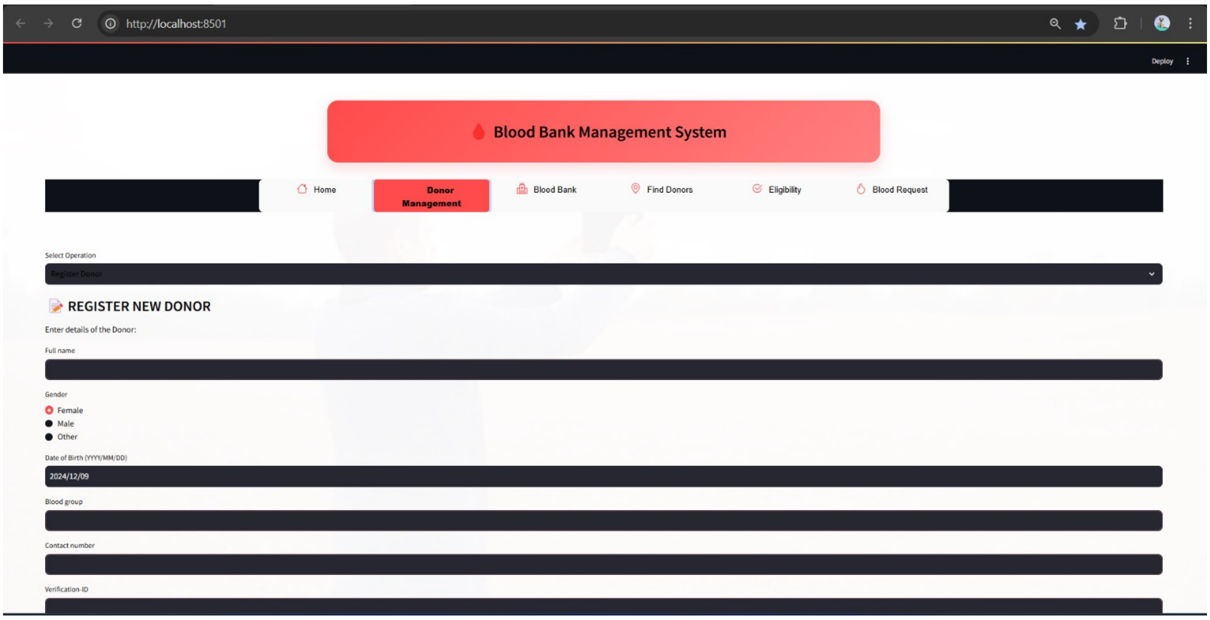


Figure 6: Main Page



Select Operation: Register Donor

REGISTER NEW DONOR

Enter details of the Donor:

Full name

Gender
- ○ Female
- ● Male
- ● Other

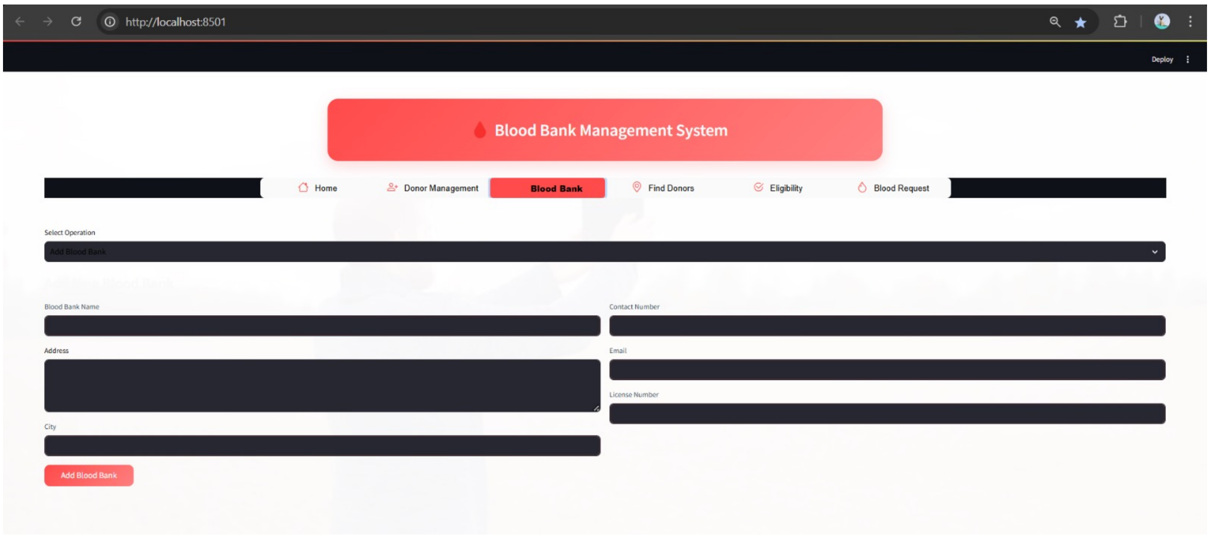
Date of Birth (YYYY/MM/DD): 2024/12/09

Blood group

Contact number

Verification-ID

Figure 7: Donor Management



Select Operation: Add Blood Bank

Add New Blood Bank

Blood Bank Name

Contact Number

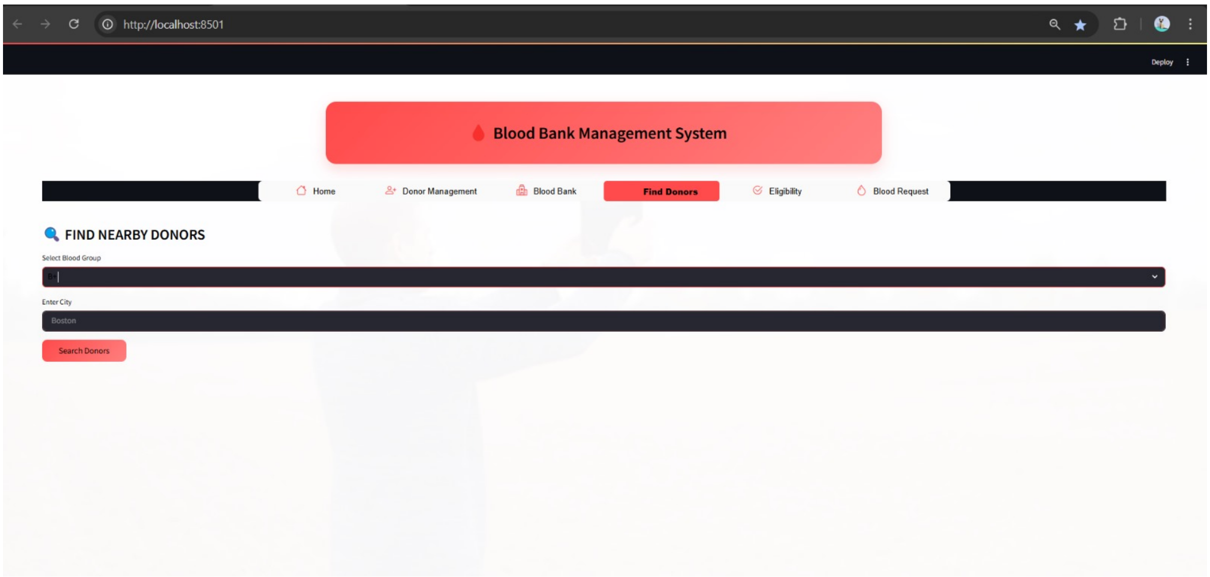
Address

Email

License Number

City

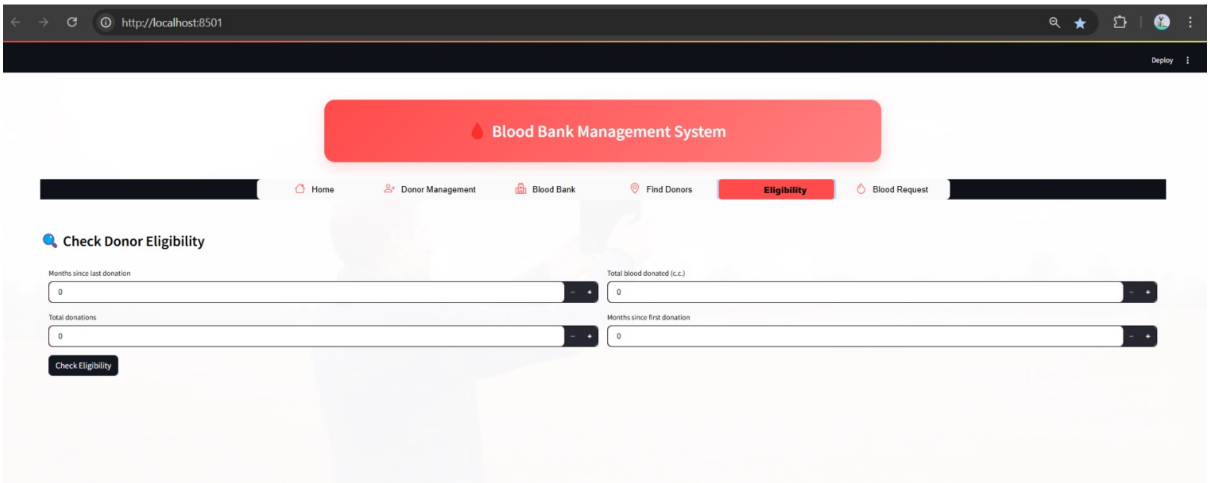
Figure 8: Blood Bank



Select Blood Group: B+

Enter City: Boston

Figure 9: Find Donors



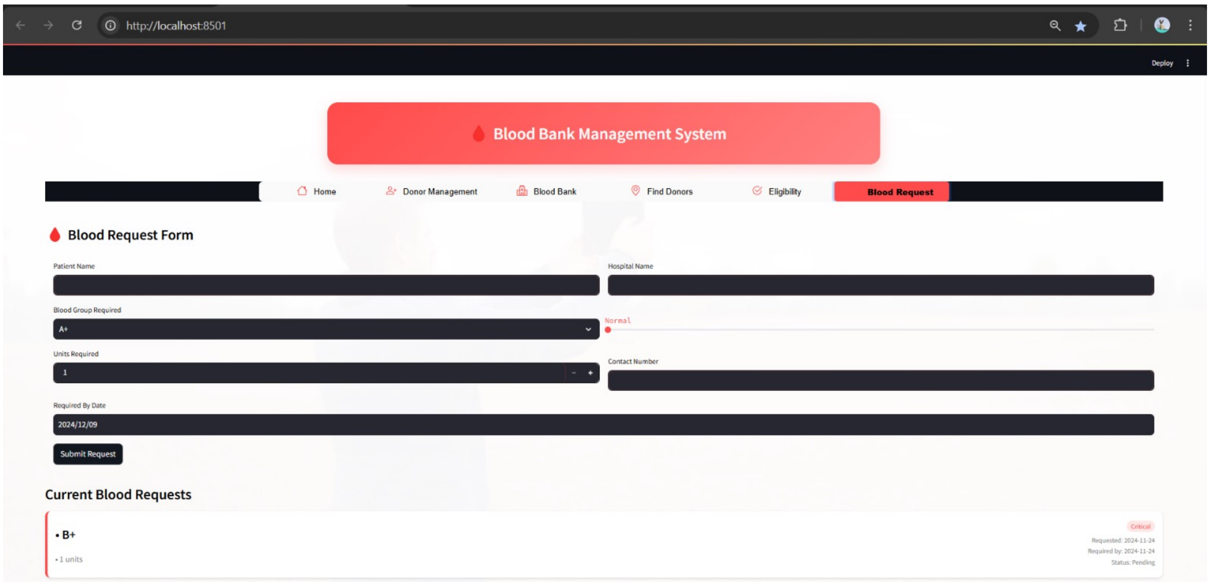
Months since last donation: 0

Total blood donated (c.c.): 0

Total donations: 0

Months since first donation: 0

Figure 10: Checking eligibility



Blood Bank Management System

Blood Request Form

Patient Name

Hospital Name

Blood Group Required: A+

Normal

Units Required: 1

Contact Number

Required By Date: 2024/12/09

Current Blood Requests

- B+
- 1 units

Critical
Requested: 2024-11-24
Required by: 2024-11-24
Status: Pending

Figure 11: Blood Request

5 Discussion

Data analysis from the healthcare management system revealed various valuable patterns on donor behavior, blood supply, and demand. First, donor demographics revealed that younger people, particularly those aged 18 to 35, donate more frequently than older people. This finding is consistent with previous research, which has found that younger persons are more likely to participate in donation programs due to fewer health restrictions and higher levels of interest. However, older contributors, particularly those aged 50 and higher, demonstrated a consistent interest in donating, implying that tailored outreach initiatives could better involve this group.Furthermore, the blood type distribution revealed that rare blood types, such as O-negative, were in higher need but less commonly donated, emphasizing the need for efforts to increase donations of these rare types.

Another major discovery concerned geographical changes in blood demand. Certain places, particularly rural areas, have faced ongoing shortages of specific blood types. This is consistent with the research, which shows that rural areas frequently experience greater difficulty in maintaining enough blood supply due to limited access to donation facilities. According to the statistics, localized blood donation drives and mobile units could help solve these shortages, resulting in more timely emergency responses. Furthermore, the study found that blood banks with well-defined donation interval regulations kept better balanced stocks, which is critical for assuring blood availability without overburdening donors.

The correlation study revealed vital information on the relationships between many characteristics, such as donor age and donation frequency, as well as the effect of donation intervals on blood availability. These findings are consistent with previous research highlighting the need of maintaining an ideal donation frequency to guarantee a continuous blood supply. The research also suggested that blood bank inventory management systems should be improved so that supply and demand are more accurately balanced. For example, blood banks should launch focused recruiting initiatives for rarer blood types to ensure that these important supplies are available when they are most required.

While the findings were generally consistent with current research, significant differences were discovered. For example, despite previous research indicating that donations among older people are declining owing to health concerns, this analysis discovered that many older donors continue to donate on a regular basis. This could be related to specific engagement techniques used by some blood banks to keep older donors involved. Furthermore, whereas urban locations are frequently identified as experiencing greater blood shortages due to increased demand, this study discovered that rural areas were more affected, which could be attributed to disparities in blood bank infrastructure and mobile donation availability. Despite these inconsistencies, the findings underline the importance of data-driven efforts to optimize blood donation processes and assure a consistent and adequate blood supply.

6 Conclusion

The provided code implements a comprehensive machine learning pipeline for blood donation prediction using the ”transfusion.csv” dataset. The process begins with data preprocessing, including renaming columns and removing duplicates. Exploratory data analysis is conducted using visualizations such as histograms, box plots, and scatter plots to understand feature distributions and relationships. The code then performs outlier detection using Z-score and Interquartile Range (IQR) methods, and creates a new feature called ”donation-duration.” For model preparation, the target variable ”is-donor” is separated from the feature set, and feature transformation is applied using PowerTransformer. The pipeline includes model selection, implementing various classifiers such as Logistic Regression, Random Forest, and CatBoost, which are evaluated using cross-validation and ROC AUC score. Hyperparameter tuning is performed using Optuna for optimizing CatBoost parameters. Finally, the selected model (Logistic Regression in this case) is trained on the preprocessed data, and predictions are generated to calculate the final AUC score on the test set.

7 References

- T. Jaiswal, S. Singhal, J. N. Singh and S. Singh Yadav, ”Blood Donation System,” 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N).

- W. B. Zulfikar, Y. A. Gerhana and A. F. Rahmania, ”An Approach to Classify Eligibility Blood Donors Using Decision Tree and Naive Bayes Classifier,” 2018 6th International Conference on Cyber and IT Service Management (CITSM).

- M. Kaur et al., ”A Web-based Blood Bank System for Managing Records of Donors and Receipts,” 2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES).

- P. A. J. Sandaruwan, U. D. L. Dolapihilla, D. W. N. R. Karunathilaka, W. H. Rankothge and N. D. U. Gamage, ”Towards an Efficient and Secure Blood Bank Management System,” 2020 IEEE 8th R10 Humanitarian Technology Conference (R10-HTC).

- B. Qiao, Z. Zhou, H. Yang, and J. Cao, “Bird species recognition based on SVM classifier and decision tree,” in 2017 First International Conference on Electronics Instrumentation Information Systems (EIIS), 2017, pp. 1–4

A Appendix A: Code

```python
import streamlit as st
from streamlit_option_menu import option_menu
from streamlit_lottie import st_lottie
import requests
import database as db
from donor import Donor
from doctor import Doctor
from blood_bank import BloodBank
from blood_request import BloodRequest
import sqlite3 as sql
import pandas as pd
from sklearn.preprocessing import StandardScaler
import joblib
import os
import plotly.express as px
import plotly.graph_objects as go
from datetime import datetime, timedelta
import calendar

def create_button_with_description(icon, title, description, key, color="#ff4b4b"):
    st.markdown(f"""
        <style>
        .button-{key} {{
            background-color: white;
            border: 2px solid {color};
            padding: 20px;
            border-radius: 15px;
            margin-bottom: 20px;
            cursor: pointer;
            transition: all 0.3s ease;
            position: relative;
            overflow: hidden;
        }}
        .button-{key}:hover {{
            transform: translateY(-5px);
            box-shadow: 0 8px 15px rgba(0, 0, 0, 0.1);
            background: linear-gradient(45deg, {color}15, white);
        }}
        .button-{key}:hover .button-icon-{key} {{
            transform: scale(1.1);
        }}
        .button-icon-{key} {{
            font-size: 2.5em;
            margin-bottom: 10px;
            color: {color};
            transition: transform 0.3s ease;
        }}
        .button-title-{key} {{
            font-size: 1.3em;
            font-weight: 600;
            color: #000000;
            margin-bottom: 8px;
        }}
        .button-desc-{key} {{
            font-size: 0.9em;
            color: #000000;
            line-height: 1.4;
        }}
        .button-stats-{key} {{
            margin-top: 10px;
            padding-top: 10px;
            border-top: 1px solid #eee;
            font-size: 0.8em;
            color: #666;
        }}
        </style>
        <div class="button-{key}" onclick="document.getElementById('{key}').click()">
            <div class="button-icon-{key}">{icon}</div>
            <div class="button-title-{key}">{title}</div>
            <div class="button-desc-{key}">{description}</div>
            <div class="button-stats-{key}">
                {get_button_stats(key)}
            </div>
        </div>
    """, unsafe_allow_html=True)
    return st.button("", key=key, help=description)

def get_button_stats(key):
    """Return relevant statistics based on button type"""
    try:
        conn = sql.connect('database_1A.db')
        cursor = conn.cursor()

        if key == "donor_btn":
            cursor.execute("SELECT COUNT(*) FROM donors")
            total_donors = cursor.fetchone()[0]
            cursor.execute("SELECT MAX(last_donation_date) FROM donors")
            last_date = cursor.fetchone()[0] or "No donations yet"
            return f"Active Donors: {total_donors} | Last Registration: {last_date}"

        elif key == "bank_btn":
            cursor.execute("SELECT COUNT(*) FROM blood_banks")
            total_banks = cursor.fetchone()[0]
            cursor.execute("SELECT SUM(units_required) FROM blood_requests WHERE status='Pending'")
            total_required = cursor.fetchone()[0] or 0
            return f"Active Banks: {total_banks} | Required Units: {total_required}"

        elif key == "request_btn":
            cursor.execute("SELECT COUNT(*) FROM blood_requests WHERE status='Pending'")
            pending = cursor.fetchone()[0]
            return f"Pending Requests: {pending}"

        elif key == "nearby_btn":
            cursor.execute("SELECT COUNT(DISTINCT city) FROM blood_banks")
            locations = cursor.fetchone()[0]
            return f"Available Locations: {locations}"

        elif key == "eligibility_btn":
            return f"check eligibility"


    except Exception as e:
        return "Error fetching stats"
    finally:
        conn.close()

    return ""  # Default return for unhandled keys

def handle_donor_management(p):
    st.markdown("""
        <style>
        /* Style for donor management section */
        .donor-section {
            color: black !important;
        }

        /* Style for selectbox label */
        .stSelectbox label {
            color: black !important;
            font-weight: 500;
        }

        /* Style for selectbox selected value */
        .stSelectbox div[data-baseweb="select"] span {
            color: white !important;
        }

        /* Style for dropdown options */
        .stSelectbox div[role="listbox"] div {
            color: black !important;
        }

        /* Style for form labels */
        .stTextInput label, .stNumberInput label, .stDateInput label, .stTextArea label {
            color: black !important;
            font-weight: 500;
        }

        /* Style for form inputs */
        .stTextInput input, .stNumberInput input, .stDateInput input, .stTextArea textarea {
            color: white !important;
        }
        </style>
    """, unsafe_allow_html=True)

    st.markdown('<div class="donor-section">', unsafe_allow_html=True)
```

```python

    donor_option = st.selectbox(
        'Select Operation',
        ['Register Donor', 'Update Donor Info', 'Delete Donor', 'Show All Donors', 'Search Donor']
    )

    if donor_option == 'Register Donor':
        st.markdown('<h3 style="color: black;">       REGISTER NEW DONOR</h3>', unsafe_allow_html=True)
        p.add_donor()
    elif donor_option == 'Update Donor Info':
        st.markdown('<h3 style="color: black;">    UPDATE DONOR INFO</h3>', unsafe_allow_html=True)
        p.update_donor()
    elif donor_option == 'Delete Donor':
        st.markdown('<h3 style="color: black;">       DELETE DONOR</h3>', unsafe_allow_html=True)
        try:
            p.delete_donor()
        except sql.IntegrityError:
            st.error('This entry cannot be deleted as other records are using it.')
    elif donor_option == 'Show All Donors':
        st.markdown('<h3 style="color: black;">       COMPLETE DONOR RECORD</h3>', unsafe_allow_html=True)
        p.show_all_donors()
    elif donor_option == 'Search Donor':
        st.markdown('<h3 style="color: black;">       SEARCH DONOR</h3>', unsafe_allow_html=True)
        p.search_donor()

    st.markdown('</div>', unsafe_allow_html=True)

def handle_nearby_donors(p):
    st.markdown("<h3 style='color: black;'>       FIND NEARBY DONORS</h3>", unsafe_allow_html=True)

    blood_group = st.selectbox(
        "Select Blood Group",
        ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'],
        key="blood_group_select"
    )

    # City input
    city = st.text_input(
        "Enter City",
        placeholder="Boston",
        key="donor_city"
    )

    if st.button("Search Donors"):
        if city and blood_group:
            p.find_nearby_donors(city, blood_group)  # Update your Donor class method accordingly
        else:
            st.warning("Please enter both city and blood group to search")
```

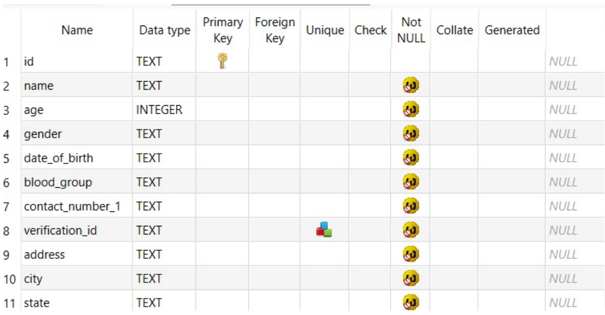
```python

def handle_blood_bank_management(p):
    st.markdown("""
        <style>
        /* Style for blood bank section */
        .blood-bank-section {
            color: black !important;
        }

        /* Style for selectbox label */
        .stSelectbox label {
            color: black !important;
            font-weight: 500;
        }

        /* Style for selectbox selected value */
        .stSelectbox div[data-baseweb="select"] span {
            color: white !important;
        }

        /* Style for dropdown options */
        .stSelectbox div[role="listbox"] div {
            color: black !important;
        }

        /* Style for text area (address) */
        .stTextArea textarea {
            color: white !important;
        }

        /* Style for text area label */
        .stTextArea label {
            color: black !important;
        }

        /* Style for text input */
        .stTextInput input {
            color: white !important;
        }

        /* Style for text input label */
        .stTextInput label {
            color: black !important;
            font-weight: 500;
        }

        /* Style for date input */
        .stDateInput input {
            color: white !important;
        }

        /* Style for date picker selected value */
        [data-baseweb="input"] input {
            color: white !important;
        }
        </style>
    """, unsafe_allow_html=True)

    st.markdown('<div class="blood-bank-section">', unsafe_allow_html=True)

    bank_option = st.selectbox(
        'Select Operation',
        ['Add Blood Bank', 'View Blood Banks', 'Update Blood Bank']
    )

    if bank_option == 'Add Blood Bank':
        p.add_blood_bank()
    elif bank_option == 'View Blood Banks':
        st.markdown("<h3 style='color: black;'>       View Blood Banks</h3>", unsafe_allow_html=True)
        p.view_blood_banks()
    elif bank_option == 'Update Blood Bank':
        p.update_blood_bank()

    st.markdown('</div>', unsafe_allow_html=True)

def handle_blood_request():
    st.markdown("""
        <style>
        /* Style for all form inputs */
        .stTextInput input, .stNumberInput input, .stSelectbox select {
            color: white !important;
        }

        /* Style for all labels */
        .stTextInput label, .stNumberInput label, .stSelectbox label {
            color: black !important;
            font-weight: 500;
        }

        /* Style for selectbox text and options */
        .stSelectbox div[data-baseweb="select"] span {
            color: white !important;
        }

        .stSelectbox div[role="listbox"] div {
            color: black !important;
        }

        /* Style for date input */
        .stDateInput input {
            color: white !important;
        }
        .stDateInput label {
            color: black !important;
            font-weight: 500;
        }
        /* Style for date picker calendar */
        .stDateInput div[data-baseweb="calendar"] {
            color: black !important;
        }
        </style>
    """, unsafe_allow_html=True)

    st.markdown("<h3 style='color: black;'>       Blood Request Form</
   h3>", unsafe_allow_html=True)

    with st.form("blood_request_form"):
        col1, col2 = st.columns(2)

        with col1:
            patient_name = st.text_input("Patient Name", key="patient_name")
            blood_group = st.selectbox(
                "Blood Group Required",
                ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'],
                key="blood_group_req"
            )
            units_required = st.number_input(
                "Units Required",
                min_value=1,
                max_value=10,
                value=1,
                key="units_required"
            )

        with col2:
            hospital_name = st.text_input("Hospital Name", key="hospital_name")
            urgency = st.select_slider(
                "Urgency Level",
                options=['Normal', 'Urgent', 'Critical'],
                value='Normal',
                key="urgency_level"
            )
            contact_number = st.text_input("Contact Number", key="contact_number")

        # Add required_by date
        required_by = st.date_input(
            "Required By Date",
            min_value=datetime.now().date(),
            key="required_by_date"
        )

        submitted = st.form_submit_button("Submit Request")

        if submitted:
            if patient_name and hospital_name and contact_number:
                try:
                    # Use the existing database function instead of direct connection
                    query = '''
                        INSERT INTO blood_requests (
                            patient_name, blood_group, units_required,
                            urgency, hospital_name, contact_number,
                            request_date, required_by, status
                        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
                    '''

                    params = (
                        patient_name, blood_group, units_required,
                        urgency, hospital_name, contact_number,
                        datetime.now().date(), required_by, 'Pending'
                    )

                    db.execute_query(query, params)
                    st.success("Blood request submitted successfully!")

                    # Show emergency contact information
                    st.info("""
                        Emergency Contacts:
                        - Blood Bank Hotline: 1-800-BLOOD-HELP
                        - Emergency Services: 911
                        - Red Cross: 1-800-RED-CROSS
                    """)

                except Exception as e:
                    st.error(f"Error submitting request: {str(e)}")
            else:
                st.warning("Please fill in all required fields.")

    # Show existing requests
    st.markdown("<h3 style='color: black;'>Current Blood Requests</h3>", unsafe_allow_html=True)
    try:
        query = """
            SELECT
                patient_name, blood_group, units_required,
                urgency, hospital_name, request_date,
                required_by, status
            FROM blood_requests
            ORDER BY
                CASE urgency
                    WHEN 'Critical' THEN 1
                    WHEN 'Urgent' THEN 2
                    ELSE 3
                END,
                required_by ASC
        """
        results = db.execute_query(query, fetch=True)

        if results:
            for request in results:
                urgency_color = {
                    'Critical': '#ff4b4b',
                    'Urgent': '#ffa500',
                    'Normal': '#2ecc71'
                }.get(request['urgency'], '#000000')

                st.markdown(f"""
                    <div style="
                        padding: 15px;
                        border-radius: 10px;
                        margin-bottom: 10px;
                        background-color: white;
                        border-left: 5px solid {urgency_color};
                        box-shadow: 0 2px 5px rgba(0,0,0,0.1);">
                        <div style="display: flex; justify-content: space-between; align-items: center;">
                            <div>
```

```
                                <h4 style="margin: 0; color: black;">
                                    {request[’patient_name’]}     {request[’blood_group’]}
                                </h4>
                                <p style="margin: 5px 0; color: gray;">
                                    {request[’hospital_name’]}     {request[’units_required’]} units
                                </p>
                            </div>
                            <div style="text-align: right;">
                                <span style="
                                    background-color: {urgency_color}22;
                                    color: {urgency_color};
                                    padding: 3px 8px;
                                    border-radius: 15px;
                                    font-size: 0.8em;">
                                    {request[’urgency’]}
                                </span>
                                <p style="margin: 5px 0; color: gray; font-size: 0.8em;">
                                    Requested: {request[’request_date’]}<br>
                                    Required by: {request[’required_by’]}<br>
                                    Status: {request[’status’]}
                                </p>
                            </div>
                        </div>
                    </div>
                """, unsafe_allow_html=True)
        else:
            st.info("No blood requests at the moment.")

    except Exception as e:
        st.error(f"Error loading blood requests: {str(e)}")

def get_total_donors():
    try:
        conn = sql.connect(’database_1A.db’)
        cursor = conn.cursor()
        cursor.execute("SELECT COUNT(*) FROM donor_record")
        total = cursor.fetchone()[0]
        return total
    except Exception as e:
        st.error(f"Error fetching donor count: {e}")
        return 0
    finally:
        conn.close()

def get_active_blood_banks():
    try:
        conn = sql.connect(’database_1A.db’)
        cursor = conn.cursor()
        cursor.execute("SELECT COUNT(*) FROM blood_banks")
        total = cursor.fetchone()[0]
        return total
    except Exception as e:
        st.error(f"Error fetching blood bank count: {e}")
        return 0
    finally:
        conn.close()

def get_active_blood_requests():
    try:
        conn = sql.connect('database_1A.db')
        cursor = conn.cursor()
        cursor.execute("SELECT COUNT(*) FROM blood_requests WHERE status = 'Pending'")
        total = cursor.fetchone()[0]
        return total
    except Exception as e:
        st.error(f"Error fetching blood requests count: {e}")
        return 0
    finally:
        conn.close()

def get_critical_blood_groups():
    try:
        conn = sql.connect('database_1A.db')
        cursor = conn.cursor()

        # Get blood groups with pending requests and their required units
        cursor.execute("""
            SELECT blood_group, SUM(units_required) as needed
            FROM blood_requests
            WHERE status = 'Pending'
            GROUP BY blood_group
            HAVING needed > 0
            ORDER BY needed DESC
            LIMIT 2
        """)

        critical_groups = cursor.fetchall()
        if critical_groups:
            # Format blood groups for display
            critical_text = ", ".join([group[0] for group in critical_groups])
            return critical_text
        return "None"

    except Exception as e:
        return "Error"
    finally:
        conn.close()

def predict_donor_eligibility(recency, frequency, monetary, time):
    try:
        features = pd.DataFrame([[recency, frequency, monetary, time]],
                             columns=['Recency (months)', 'Frequency (times)',
                                    'Monetary (c.c. blood)', 'Time (months)'])

        if not os.path.exists('donation_model.joblib'):
            from donation_model import load_data, preprocess_data, train_model
            data = load_data('transfusion.csv')

            train_data = pd.DataFrame(data, columns=['Recency (months)', 'Frequency (times)',
                                                  'Monetary (c.c. blood)', 'Time (months)'])

            # Fit scaler on all training data first
            scaler = StandardScaler()
            scaler.fit(train_data)

            X, y = preprocess_data(data)
            X_scaled = scaler.transform(X)
            model = train_model(X_scaled, y)

            joblib.dump(model, 'donation_model.joblib')
            joblib.dump(scaler, 'scaler.joblib')

        # Load model and scaler
        model = joblib.load('donation_model.joblib')
        scaler = joblib.load('scaler.joblib')

        features_scaled = scaler.transform(features)
        prediction = model.predict(features_scaled)[0]
        probability = model.predict_proba(features_scaled)[0][1]

        return prediction, probability

    except Exception as e:
        st.error(f"Error predicting eligibility: {str(e)}")
        return None, None

def handle_eligibility_check():
    st.markdown("""
        <style>
        /* Style for eligibility section */
        .eligibility-section {
            color: black !important;
        }

        /* Style for form labels */
        .stTextInput label, .stNumberInput label {
            color: black !important;
            font-weight: 500;
        }

        /* Style for form inputs */
        .stTextInput input, .stNumberInput input {
            color: black !important;
            background-color: white !important;
            border: 1px solid rgba(255, 75, 75, 0.2) !important;
        }

        /* Style for form button */
        .stButton button {
            color: white !important;
            background-color: #ff4b4b !important;
            font-weight: 500 !important;
        }

        /* Style for form button text specifically */
        .stButton button p {
            color: white !important;
        }

        /* Rest of your existing styles... */
        </style>
    """, unsafe_allow_html=True)

    st.markdown('<div class="eligibility-section">', unsafe_allow_html=True)
    st.markdown("<h3 style='color: black;'>       Check Donor Eligibility</h3>", unsafe_allow_html=True)

    with st.form("eligibility_form"):
        col1, col2 = st.columns(2)

        with col1:
            recency = st.number_input(
                "Months since last donation",
                min_value=0,
                max_value=100,
                help="Number of months since the last donation"
            )

            frequency = st.number_input(
                "Total donations",
                min_value=0,
                max_value=100,
                help="Total number of donations made"
            )

        with col2:
            monetary = st.number_input(
                "Total blood donated (c.c.)",
                min_value=0,
                max_value=25000,
                step=250,
                help="Total volume of blood donated in c.c."
            )

            time = st.number_input(
                "Months since first donation",
                min_value=0,
                max_value=200,
                help="Number of months since first donation"
            )

        submitted = st.form_submit_button("Check Eligibility")

    if submitted:
        prediction, probability = predict_donor_eligibility(recency, frequency, monetary, time)

        if prediction is not None:
            col1, col2 = st.columns(2)

            with col1:
                if prediction == 1:
                    st.success("    Donor is likely eligible to donate")
                else:
                    # Change the warning message color to black and increase size
                    st.markdown("<h3 style='color: black;'>    Donor may need more time before next donation</h3>", unsafe_allow_html=True)

            with col2:
                probability_percentage = probability * 100
                # Change the metric display to black
                st.markdown(f"<h3 style='color: black;'>Donation Probability</h3>", unsafe_allow_html=True)
                st.markdown(f"<h2 style='color: black;'>{probability_percentage:.1f}%</h2>", unsafe_allow_html=True)

        else:
            st.error("Unable to make prediction. Please try again.")

    st.markdown('</div>', unsafe_allow_html=True)

def load_lottie_url(url):
    r = requests.get(url)
    if r.status_code != 200:
        return None
    return r.json()

def create_dashboard_metrics():
    col1, col2, col3, col4 = st.columns(4)

    with col1:
        st.markdown("""
            <div class="metric-card">
                <div class="metric-icon">      </div>
                <div class="metric-value" style="color: black;">{}</div>
                <div class="metric-label" style="color: black;">Total Donors</div>
                <div class="metric-trend" style="color: black;">    12% this month</div>
            </div>
        """.format(get_total_donors()), unsafe_allow_html=True)

    with col2:
        st.markdown("""
            <div class="metric-card">
                <div class="metric-icon">      </div>
                <div class="metric-value">{}</div>
                <div class="metric-label">Blood Banks</div>
                <div class="metric-trend">Active Centers</div>
            </div>
        """.format(get_active_blood_banks()), unsafe_allow_html=True)

    with col3:
        st.markdown("""
            <div class="metric-card">
                <div class="metric-icon">   </div>
                <div class="metric-value">{}</div>
                <div class="metric-label">Pending Requests</div>
                <div class="metric-trend urgent">Urgent Need</div>
            </div>
        """.format(get_active_blood_requests()), unsafe_allow_html=True)

    with col4:
        st.markdown("""
            <div class="metric-card">
                <div class="metric-icon">      </div>
                <div class="metric-value">{}</div>
                <div class="metric-label">Critical Groups</div>
                <div class="metric-trend">Required Now</div>
            </div>
        """.format(get_critical_blood_groups()), unsafe_allow_html=True)

def create_blood_group_distribution():
    # Get blood group counts from your database
    try:
        conn = sql.connect('database_1A.db')
        cursor = conn.cursor()
        cursor.execute("""
            SELECT blood_group, COUNT(*) as count
            FROM donors
            GROUP BY blood_group
            ORDER BY blood_group
        """)
        data = cursor.fetchall()
        blood_groups = [row[0] for row in data]
        counts = [row[1] for row in data]
    except Exception as e:
        # Fallback data if database query fails
        blood_groups = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']
        counts = [150, 50, 140, 45, 80, 30, 160, 40]
    finally:
        conn.close()

    fig = go.Figure(data=[
        go.Pie(
            labels=blood_groups,
            values=counts,
            hole=0.3,
            textinfo='label+percent',
            marker=dict(colors=['#ff4b4b', '#ff6b6b', '#ff8080', '#ff9999',
                                '#ffb3b3', '#ffcccc', '#ffe6e6', '#fff0f0']),
            textfont=dict(color='black', size=14),
            insidetextorientation='horizontal'
        )
    ])

    fig.update_layout(
        title={
            'text': "Blood Group Distribution",
            'font': {'color': 'black', 'size': 24},
            'x': 0.5,
            'xanchor': 'center',
            'y': 0.95
        },
        plot_bgcolor='rgba(0,0,0,0)',
        height=400,
        font={'color': 'black'},
        showlegend=True,
        legend=dict(
            font=dict(color='black'),
            orientation="h",
            yanchor="bottom",
            y=1.02,
            xanchor="right",
            x=1
        ),
        paper_bgcolor='white'
    )
    return fig

def create_age_distribution():
    try:
        conn = sql.connect('database_1A.db')
        cursor = conn.cursor()
        cursor.execute("""
            SELECT
                CASE
                    WHEN age < 25 THEN '18-24'
                    WHEN age BETWEEN 25 AND 34 THEN '25-34'
                    WHEN age BETWEEN 35 AND 44 THEN '35-44'
                    WHEN age BETWEEN 45 AND 54 THEN '45-54'
                    ELSE '55+'
                END as age_group,
                COUNT(*) as count
            FROM donors
            GROUP BY age_group
            ORDER BY age_group
        """)
        data = cursor.fetchall()
        age_groups = [row[0] for row in data]
        counts = [row[1] for row in data]
    except Exception as e:
        # Fallback data if database query fails
        age_groups = ['18-24', '25-34', '35-44', '45-54', '55+']
        counts = [200, 350, 250, 150, 50]
    finally:
        conn.close()

    fig = go.Figure(data=[
        go.Bar(
            x=age_groups,
            y=counts,
            marker_color='#ff6b6b',
            text=counts,
            textposition='auto',
            textfont=dict(color='black', size=14),
            hovertemplate='Age: %{x}<br>Count: %{y}<extra></extra>'
        )
    ])

    fig.update_layout(
        title={
            'text': "Age Distribution",
            'font': {'color': 'black', 'size': 24},
            'x': 0.5,
            'xanchor': 'center',
            'y': 0.95
        },
        plot_bgcolor='rgba(0,0,0,0)',
        height=400,
        font={'color': 'black'},
        xaxis={
            'title': {'text': 'Age Groups', 'font': {'color': 'black'}},
            'tickfont': {'color': 'black', 'size': 12},
            'ticktext': age_groups,
            'tickvals': age_groups
        },
        yaxis={
            'title': {'text': 'Number of Donors', 'font': {'color': 'black'}},
            'tickfont': {'color': 'black', 'size': 12}
        },
        paper_bgcolor='white',
        bargap=0.3
    )
    return fig

def home():
    # Existing page config
    st.set_page_config(
        page_title="Blood Bank Management System",
        page_icon="      ",
        layout="wide",
        initial_sidebar_state="expanded"
    )

    # Enhanced CSS
    st.markdown("""
        <style>
        .stApp {
            background-image: linear-gradient(rgba(255, 255, 255, 0.97), rgba(255, 255, 255, 0.97)),
                url("https://img.freepik.com/free-photo/close-up-doctor-holding-blood-sample_23-2149140414.jpg");
            background-size: cover;
            background-position: center;
            background-repeat: no-repeat;
            background-attachment: fixed;
        }
```

```css

        .main-header {
            background: linear-gradient(135deg, #ff4b4b, #ff8080);
            padding: 1.5rem;
            border-radius: 20px;
            box-shadow: 0 10px 25px rgba(255, 75, 75, 0.2);
            margin-bottom: 2rem;
            text-align: center;
            white-space: nowrap;
            overflow: hidden;
        }

        .metric-card {
            background: white;
            padding: 1.8rem;
            border-radius: 20px;
            box-shadow: 0 8px 20px rgba(0,0,0,0.08);
            text-align: center;
            transition: all 0.3s ease;
            border: 1px solid rgba(255, 75, 75, 0.1);
        }

        .metric-card:hover {
            transform: translateY(-5px);
            box-shadow: 0 12px 30px rgba(0,0,0,0.12);
            border-color: rgba(255, 75, 75, 0.3);
        }

        .metric-icon {
            font-size: 2.8em;
            margin-bottom: 0.8rem;
            color: #ff4b4b;
            background: rgba(255, 75, 75, 0.1);
            width: 70px;
            height: 70px;
            display: flex;
            align-items: center;
            justify-content: center;
            border-radius: 50%;
            margin: 0 auto 1rem auto;
        }

        .metric-value {
            font-size: 2.2em;
            font-weight: 700;
            color: #2c3e50;
            margin: 0.5rem 0;
            background: linear-gradient(45deg, #ff4b4b, #ff8080);
            -webkit-background-clip: text;
            -webkit-text-fill-color: transparent;
        }

        .metric-label {
            color: #2c3e50;
            font-size: 1.1em;
            font-weight: 600;
            margin-bottom: 0.5rem;
            text-transform: uppercase;
            letter-spacing: 0.5px;
        }

        .metric-trend {
            color: #2c3e50;
            font-size: 0.9em;
            padding: 0.3rem 0.8rem;
            border-radius: 15px;
            background: rgba(255, 75, 75, 0.1);
            display: inline-block;
        }

        .nav-link {
            background: white !important;
            border-radius: 15px !important;
            margin: 0.3rem !important;
            transition: all 0.3s ease !important;
            border: 1px solid rgba(255, 75, 75, 0.1) !important;
            padding: 0.8rem 1.5rem !important;
        }

        .nav-link:hover {
            transform: translateY(-2px) !important;
            background: rgba(255, 75, 75, 0.05) !important;
            border-color: rgba(255, 75, 75, 0.3) !important;
        }

        .nav-link.active {
            background: linear-gradient(135deg, #ff4b4b, #ff8080) !important;
            color: white !important;
            border: none !important;
            box-shadow: 0 5px 15px rgba(255, 75, 75, 0.3) !important;
        }

        /* Chart styling */
        .js-plotly-plot {
            border-radius: 20px;
            box-shadow: 0 8px 20px rgba(0,0,0,0.08);
            padding: 1rem;
            background: white;
            border: 1px solid rgba(255, 75, 75, 0.1);
            margin-bottom: 2rem;
        }

        /* Make all text inputs and labels consistent */
        .stTextInput label, .stNumberInput label, .stSelectbox label {
            color: #2c3e50 !important;
            font-weight: 600;
            font-size: 1rem;
        }

        .stTextInput input, .stNumberInput input, .stSelectbox select {
            border-radius: 10px;
            border: 1px solid rgba(255, 75, 75, 0.2);
            padding: 0.5rem 1rem;
        }

        .stButton>button {
            background: linear-gradient(135deg, #ff4b4b, #ff8080);
            color: white;
            border: none;
            padding: 0.5rem 2rem;
            border-radius: 10px;
            font-weight: 600;
            transition: all 0.3s ease;
        }

        .stButton>button:hover {
            transform: translateY(-2px);
            box-shadow: 0 5px 15px rgba(255, 75, 75, 0.3);
        }
        </style>
    """, unsafe_allow_html=True)

    # Header without Lottie Animation
    col1, col2, col3 = st.columns([1,2,1])
    with col2:
        st.markdown("""
            <div class="main-header">
                <h2 style="white-space: nowrap; font-size: 32px;"> Blood Bank Management System</h2>
            </div>
        """, unsafe_allow_html=True)

    # Navigation Menu
    selected = option_menu(
        menu_title=None,
        options=["Home", "Donor Management", "Blood Bank", "Find Donors", "Eligibility", "Blood Request"],
        icons=['house', 'person-plus', 'hospital', 'geo-alt', 'check2-circle', 'droplet'],
        menu_icon="cast",
        default_index=0,
        orientation="horizontal",
        styles={
            "container": {"padding": "0!important", "background-color": "#fafafa"},
            "icon": {"color": "#ff4b4b", "font-size": "20px"},
            "nav-link": {
                "font-size": "16px",
                "text-align": "center",
                "margin": "0px",
                "--hover-color": "#ff4b4b15",
                "color": "black",
            },
            "nav-link-selected": {"background-color": "#ff4b4b"},
        }
    )

    # Content based on selection
    if selected == "Home":
        # Show dashboard content
        create_dashboard_metrics()
        col1, col2 = st.columns(2)
        with col1:
            st.plotly_chart(create_blood_group_distribution(), use_container_width=True)
        with col2:
            st.plotly_chart(create_age_distribution(), use_container_width=True)
    elif selected == "Donor Management":
        handle_donor_management(Donor())
    elif selected == "Blood Bank":
        handle_blood_bank_management(BloodBank())
    elif selected == "Find Donors":
        handle_nearby_donors(Donor())
    elif selected == "Eligibility":
        handle_eligibility_check()
    elif selected == "Blood Request":
        handle_blood_request()

if _name_ == "_main_":
    home    ()
```

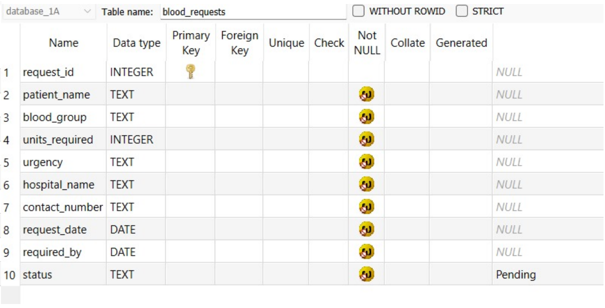
Listing 1: Code

B Appendix B: Additional Figures



	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	id	TEXT	✓							NULL
2	name	TEXT					✓			NULL
3	age	INTEGER					✓			NULL
4	gender	TEXT					✓			NULL
5	date_of_birth	TEXT					✓			NULL
6	blood_group	TEXT					✓			NULL
7	contact_number_1	TEXT					✓			NULL
8	verification_id	TEXT			✓		✓			NULL
9	address	TEXT					✓			NULL
10	city	TEXT					✓			NULL
11	state	TEXT					✓			NULL

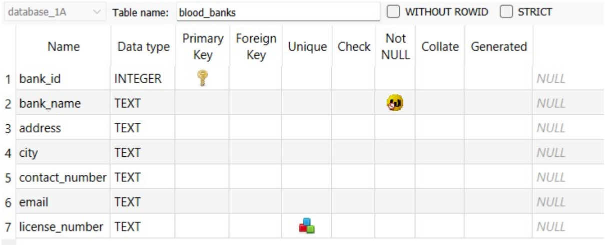
Figure 12: Donor Table



database_1A Table name: blood_requests □ WITHOUT ROWID □ STRICT

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	request_id	INTEGER	✓							NULL
2	patient_name	TEXT					✓			NULL
3	blood_group	TEXT					✓			NULL
4	units_required	INTEGER					✓			NULL
5	urgency	TEXT					✓			NULL
6	hospital_name	TEXT					✓			NULL
7	contact_number	TEXT					✓			NULL
8	request_date	DATE					✓			NULL
9	required_by	DATE					✓			NULL
10	status	TEXT					✓			Pending

Figure 13: Blood Request Table



database_1A Table name: blood_banks □ WITHOUT ROWID □ STRICT

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	bank_id	INTEGER	✓							NULL
2	bank_name	TEXT					✓			NULL
3	address	TEXT								NULL
4	city	TEXT								NULL
5	contact_number	TEXT								NULL
6	email	TEXT								NULL
7	license_number	TEXT			✓					NULL

Figure 14: Blood Bank Table