## 1. INTRODUCTION

**1.1 Overview**

Osteoporosis is a chronic and often debilitating condition that affects the strength and density of bones, making them more fragile and susceptible to fractures. It is commonly diagnosed in older adults, especially postmenopausal women, though it can affect people of all ages. The ability to predict osteoporosis risk early can significantly reduce complications by enabling timely medical intervention and lifestyle adjustments.

This project aims to develop an easy-to-use web-based tool that predicts osteoporosis risk based on key health indicators such as age, gender, calcium intake, and bone density. By leveraging machine learning models, the system will analyze user-provided data to generate an accurate prediction of osteoporosis risk. In addition, the system will store these predictions and related data for future use, enabling users and healthcare professionals to track changes over time. This tool will be an invaluable resource for health-conscious individuals and medical practitioners alike, offering insights into one’s risk for osteoporosis and encouraging early preventative action.

**1.2 Objectives**

* **Predict Osteoporosis Risk**: The primary goal is to develop an accurate prediction model that uses machine learning techniques to analyze user-provided health information. This model will classify users as either high-risk or low-risk based on data such as age, gender, calcium intake, and bone density.
* **User-Friendly Interface**: The system will provide an intuitive, web-based interface for users to input their data. This interface will guide users through the process of entering information and will clearly display the prediction results in a user-friendly manner.
* **Data Storage and Retrieval**: A key objective is to design and implement a database that securely stores user data and prediction outcomes. This allows users to retrieve past predictions and review historical data trends for long-term monitoring.
* **Awareness and Prevention**: By providing an accessible tool, the system seeks to raise awareness about the risk factors associated with osteoporosis. Users will gain valuable insights into their own health, which could lead to early intervention and better health outcomes.

**1.3 Problem Formulation**

Osteoporosis is often diagnosed too late, leading to severe health issues such as fractures, immobility, and even premature death. The problem addressed by this project is the development of a system that can assess and predict the risk of osteoporosis before the disease progresses. Current healthcare systems often do not provide easy access to osteoporosis screening unless symptoms are already present. This project seeks to close that gap by offering a simple, cost-effective, and accessible solution that predicts the likelihood of osteoporosis based on individual health inputs.

The challenge lies in developing an accurate, easy-to-use system that can help users, especially those at higher risk (such as elderly women), assess their risk level and take preventive measures. By providing timely predictions, the system can reduce the number of undiagnosed osteoporosis cases and facilitate preventive healthcare.

**1.4 Scope**

The project covers the development of a predictive tool using machine learning to assess the risk of osteoporosis. The web application will:

* Accept user inputs (age, gender, calcium intake, bone density) through a clean, easy-to-use interface.
* Process this data through a trained machine learning model that outputs a prediction (high or low risk of osteoporosis).
* Display the result clearly on the user interface for immediate interpretation.
* Store the prediction and user data in a backend database for future analysis and tracking.

The scope also includes:

* Model training using real or synthetic data on osteoporosis risk factors.
* Database design for secure storage of user inputs and prediction results.
* A simple but secure user interface for data entry and result retrieval.

Future extensions of this project could include additional data points such as family history, physical activity, and other lifestyle factors that influence bone health.

**1.5 Feasibility**

* **Technical Feasibility**: The project is technically feasible as it uses well-established technologies, including Python for machine learning, Flask for web development, and SQLite for database management. These tools are open-source, well-documented, and widely used, ensuring that they can be implemented effectively.
* **Operational Feasibility**: The project is designed to be easy to use, with a simple and intuitive web interface that does not require any specialized knowledge from the user. The predictions provided by the machine learning model will be easily interpretable, making it practical for a wide audience, including non-technical users.
* **Economic Feasibility**: The use of open-source tools such as Flask, Scikit-learn, and SQLite ensures that the project incurs no licensing costs. Furthermore, minimal hardware resources are required, making the project economically viable for small-scale or large-scale deployment without significant investment in infrastructure.

**1.6 System Requirements**

**1.6.1 Software Requirements**

* **Operating System:** Windows, Linux, or MacOS
* **Backend Framework:** Flask (Python), a lightweight Python web framework, will handle HTTP requests, data processing, and the integration of the machine learning model with the web interface.
* **Frontend:** HTML, CSS, JavaScript. The front end will use HTML, CSS, and JavaScript to create an interactive and responsive user interface, compatible with modern browsers.
* **Database:** SQLite will serve as the database for storing user data and predictions. It is lightweight, serverless, and suitable for smaller datasets.
* **Machine Learning Libraries:** Scikit-learn, Pandas, NumPy
* **Scikit-learn**: This popular machine learning library will be used to develop and implement the predictive model for osteoporosis risk.
* **Pandas** and **NumPy** will be used for data manipulation and analysis during the model-building phase.
* **Browser:** The application will work on any modern web browser, such as Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge, making it accessible from any device.
* **IDE:** VS Code or PyCharm
  + Development will take place in VS Code or PyCharm, both of which are user-friendly integrated development environments (IDEs) with excellent support for Python and Flask.

**1.6.2 Hardware Requirements**

* **Processor:** Dual-core processor (2.5 GHz or higher)

A dual-core processor with a speed of 2.5 GHz or higher will suffice for running both the application and the machine learning model without significant delays.

* **RAM:** Minimum 4GB

RAM will be required to run the Flask application and the machine learning model concurrently without performance issues.

* **Storage:** 100MB free space for the project

Storage (approximately 100MB) for storing user data, the trained model, and application resources.

* **Internet Connection:**

An active internet connection will be necessary for accessing the web-based application.

**2. REQUIREMENTS**

**2.1 Requirements Specification**

* **User Inputs:** Age, gender, calcium intake, bone density.

The system will collect the following key inputs from users:

* + **Age**: This is a critical factor in determining osteoporosis risk, as bone density tends to decrease with age, particularly after 50.
  + **Gender**: Osteoporosis disproportionately affects women, particularly post-menopausal women. Gender input will help the model in determining the appropriate risk factor.
  + **Calcium Intake**: Calcium is essential for maintaining bone health. The system will accept daily calcium intake (in mg/day) from users to assess their dietary contribution to bone health.
  + **Bone Density**: Users will input their bone density score (g/cm²), which is a crucial measurement in assessing bone strength and osteoporosis risk.
* **Prediction:** A machine learning model will predict osteoporosis risk.

Once the user inputs their data, the system will run the information through a **pre-trained machine learning model** designed to predict osteoporosis risk. The model will use algorithms such as Random Forest or Logistic Regression to classify users into high-risk or low-risk categories based on the provided health parameters. The results will be numeric (risk score) and textual (either "High Risk" or "Low Risk"). This prediction will help users understand their susceptibility to osteoporosis and encourage timely action.

* **Database Storage:** Predictions will be stored in an SQLite database for future analysis.

The application will utilize an **SQLite** database to store user data and the corresponding prediction outcomes. This enables users to:

* + View past predictions.
  + Track changes in their health over time.
  + Allow for data-driven research or future improvements in model accuracy by analyzing stored records.
* **Frontend UI:** User-friendly web interface for input and result display.

The system will offer an intuitive, web-based user interface built using **HTML, CSS, and JavaScript**. The design will prioritize simplicity and ease of use to ensure that users of all ages and backgrounds can easily input data, understand the results, and navigate through the application

* **Backend:** Flask for handling requests and responses.

The application’s backend will be handled using **Flask**, a lightweight Python web framework. Flask will process incoming requests, handle user inputs, communicate with the machine learning model to produce predictions, and store/retrieve data from the SQLite database. Flask will also serve the frontend and manage the connection between the client (browser) and the server.

**2.2 User Interfaces**

The web application will feature several pages that ensure seamless navigation and functionality. Each page is designed to cater to different aspects of user interaction and data processing.

* **Home Page:** The home page will serve as the entry point to the application. It will:
  + Welcome users and introduce them to the system’s purpose.
  + Provide a brief overview of osteoporosis and the importance of early prediction.
  + Contain links or navigation buttons that direct users to the prediction page or the prediction history page.
  + Include a visually appealing layout with background images or graphics related to bone health and osteoporosis to engage users.
* **Prediction Page:** Provides fields for entering user data (age, gender, calcium intake, bone density) and a button for making predictions.

This is the main functional page where users will enter their health data. It will feature:

* + Input fields for **age**, **gender** (using radio buttons for male/female), **calcium intake** (drop-down or number input), and **bone density**.
  + A **"Predict"** button that sends the user’s data to the machine learning model for analysis.
  + Proper validation to ensure all required fields are filled in before submission.
  + Once the prediction is processed, the user will either be redirected to the results page or shown the result on the same page, depending on implementation.
* **Result Page:** Displays whether the user is at high or low risk of osteoporosis based on their inputs.

After submitting the data, users will be shown their osteoporosis risk level:

* + The result will be displayed as a clear textual output such as **"You are at high risk of osteoporosis"** or **"You are at low risk of osteoporosis"**.
  + The page will also provide users with additional information or suggestions, such as visiting a healthcare provider for a bone density scan if the result indicates a high risk.
  + Optionally, the page may include a summary of the user’s inputs alongside the prediction for reference.

**Prediction History Page:** Shows previously stored predictions and associated user data. Users will have access to a history page that displays all past predictions made by the system. This page will include:

* + A table that lists each prediction along with the associated data (age, gender, calcium intake, bone density, risk level).
  + Each row in the table will represent a prediction, allowing users to see how their osteoporosis risk has changed over time.
  + The data will be fetched from the **SQLite** database, ensuring it is persistent across sessions.
* The page will have features like sorting or filtering the table by date or risk level for easier navigation.

**2.3 Constraints and Prerequisites**

* **Machine Learning Model:** A pre-trained machine learning model is required to make predictions. The training process, including feature selection, model optimization, and validation, must be completed before deploying the web application. The model could be based on algorithms such as **Random Forest**, **Logistic Regression**, or other classification techniques. The accuracy of the predictions will depend on how well the model is trained, validated, and tested.
* **Database:** An **SQLite** database needs to be configured before the system can store predictions. The database schema must include tables for user data, predictions, and metadata (e.g., timestamps). This ensures that each prediction is properly stored and can be retrieved when needed. Database connection logic in Flask must also be established, and queries must be optimized for fast and reliable access to stored data.
* **User Data:**

Proper data validation is essential to ensure the inputs are correctly formatted for prediction. Age, calcium intake, and bone density must fall within realistic ranges for accurate predictions. For example:

* + Age should be between 18 and 100.
  + Calcium intake should be within a reasonable daily range (e.g., 500–1500 mg/day).
  + Bone density inputs should be clinically valid (e.g., 0.5–1.5 g/cm²). Inaccurate or outlier data can lead to incorrect predictions, so form validation and error handling must be in place.
* **Model Interpretability:**

The predictions should be easily interpretable for users. A **low-risk** or **high-risk** result should come with a brief explanation or recommendation. For example, a high-risk prediction might be followed by a suggestion to consult a healthcare provider for a bone density scan or to improve calcium intake and physical activity.

* **Performance Constraints**:

Since this is a web-based tool, performance needs to be optimized for fast processing times. The time between submitting data and receiving a prediction should be minimal to enhance user experience. This may require optimization of both the machine learning model and the database queries.

## 3. ANALYSIS

**3.1 Use Case Model**

**3.1.1 Use Case Diagram**

The **Use Case Diagram** outlines the interactions between users and the system. It defines the relationship between the user and the functions they can perform within the osteoporosis risk prediction system.

**User :** This can be anyone using the system, such as patients or healthcare providers, who input personal and medical data (age, gender, calcium intake, and bone density) for risk assessment.

**Use Cases:**

1. **Input User Data:** The user inputs the necessary health information.
2. **Predict Risk:** The system runs a machine learning model on the input data to determine osteoporosis risk.
3. **Display Prediction:** The system provides feedback (whether the user is at high or low risk).

**Store Prediction:** The system stores the user’s prediction data in the database for future reference.

**3.1.1.1 Flow of Events**

1. **User Inputs Data:**

The user visits the **Prediction Page** and is presented with a form where they can enter personal

details: age, gender, calcium intake, and bone density. The user submits the form by clicking the

"Predict Risk" button.

1. **System Predicts Risk:**

Once the user’s input is submitted, the system uses a machine learning model (e.g., Random Forest)

to predict the risk of osteoporosis. The system processes the input data and runs it through the

trained model.

1. **Result Displayed:**

After processing, the system provides a prediction of either "High Risk" or "Low Risk" based on

the input data. The system sends the prediction result back to the web interface, and the result is

displayed on the screen. The prediction (High Risk/Low Risk) is shown to the user along with

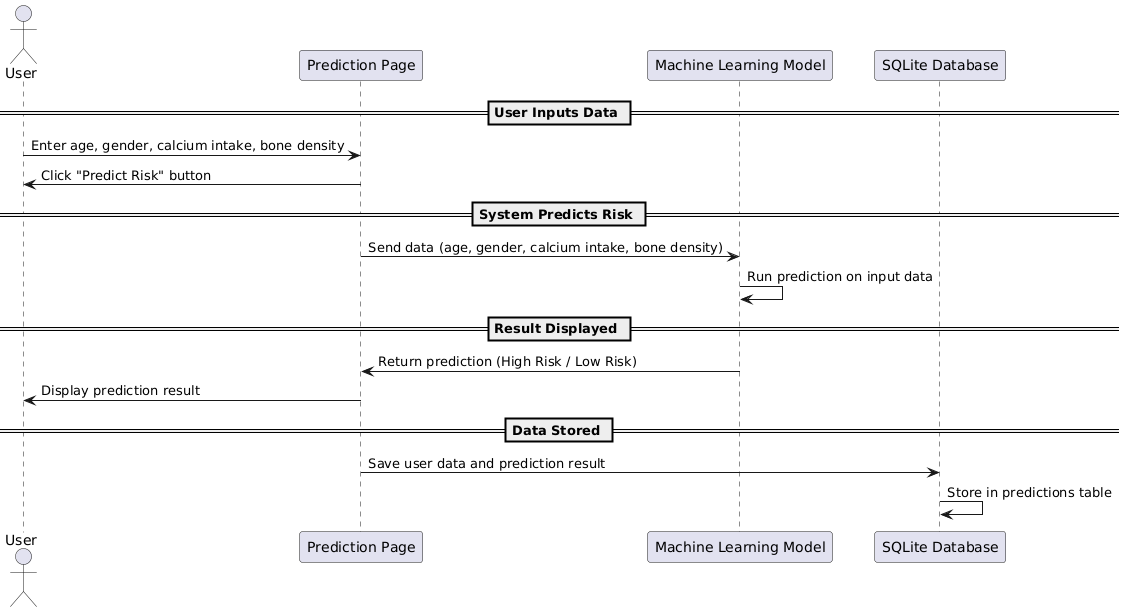
relevant details.

1. **Data Stored:**

The prediction result, along with the user’s input data (age, gender, calcium intake, and bone

density), is saved in the system’s SQLite database for future analysis. Once the prediction is

generated, the system automatically saves the data in the database.



* + 1. **Activity Diagram**

Activity diagrams provide a step-by-step visual flow of how specific tasks are carried out in the system. These diagrams help us understand the flow of events and decisions made during different phases of user interaction with the system.

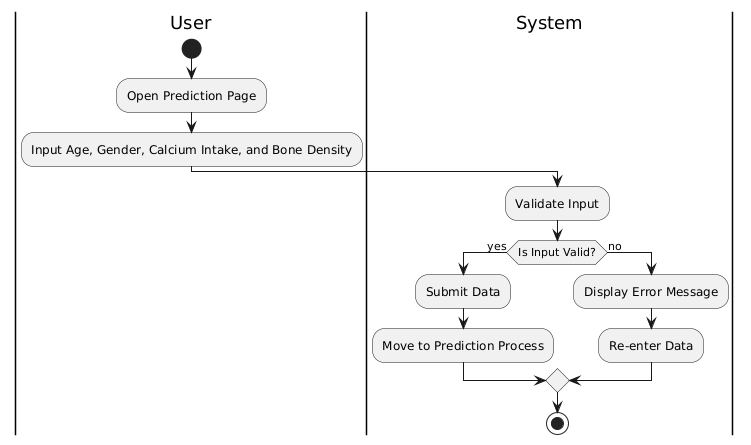
**3.1.2.1 Activity Diagram for Data Input:**

This diagram visualizes the process of entering user data into the system:

1. **Start:** User opens the prediction page.
2. **Input Age, Gender, Calcium Intake, and Bone Density:** User fills in all the required fields (age, gender, calcium intake, bone density) in the form.
3. **Validate Input:** The system checks if all required fields are filled and data is valid (e.g., non-empty, valid values).

* If valid, proceed to the next step.
* If invalid, an error message is displayed, and the user is asked to re-enter data.

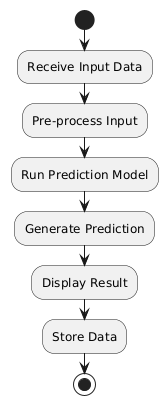
1. **Submit Data:** Once the data is valid, the user clicks the "Predict Risk" button to submit the form.
2. **End:** The system moves to the prediction process.



* + - 1. **Activity Diagram for Prediction:**

This diagram visualizes the system processing the input data and generating the osteoporosis risk prediction:

1. **Start:** System receives input data (age, gender, calcium intake, bone density) from the user.
2. **Pre-process Input:** The system may normalize or scale data, depending on the requirements of the machine learning model.
3. **Run Prediction Model:** The input data is passed through the pre-trained machine learning model (e.g., Random Forest classifier).
4. **Generate Prediction:** The system generates a prediction (e.g., "High Risk" or "Low Risk").
5. **Display Result:** The prediction result is displayed to the user.
6. **Store Data:** The user’s input data and prediction are saved in the database for future reference.
7. **End:** The user is presented with an option to either return to the home page or view stored predictions.



**4. Design**

**4.1 Database Design**

The database design is centered around storing user inputs and their respective osteoporosis risk predictions. The main table, **Predictions**, will store the following data:

* **id**: Primary key, an auto-incremented integer that uniquely identifies each record.
* **age**: Integer representing the age of the user.
* **gender**: Integer where 0 represents male and 1 represents female.
* **calcium\_intake**: Integer representing the user’s daily calcium intake in milligrams.
* **bone\_density**: Float representing the user’s bone density (g/cm²).
* **risk**: Integer where 0 indicates low risk and 1 indicates high risk of osteoporosis.

The database will be in **SQLite**, and this table will store multiple predictions per user. The structure is simple, efficient, and scalable for storing and querying prediction history.

#### **4.2 E-R Diagram**

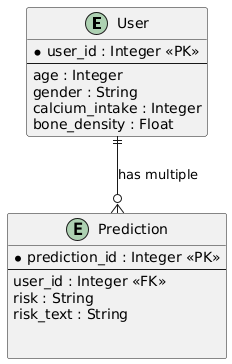
**Entities:** User, Prediction

In this project, there are two main entities:

* **User**: Represents the individual using the system. While the user inputs data, the system does not necessarily need to store user profiles. Therefore, user data is represented within the prediction table.
* **Prediction**: Represents the result of the osteoporosis risk prediction based on user inputs.

**Relationships:**

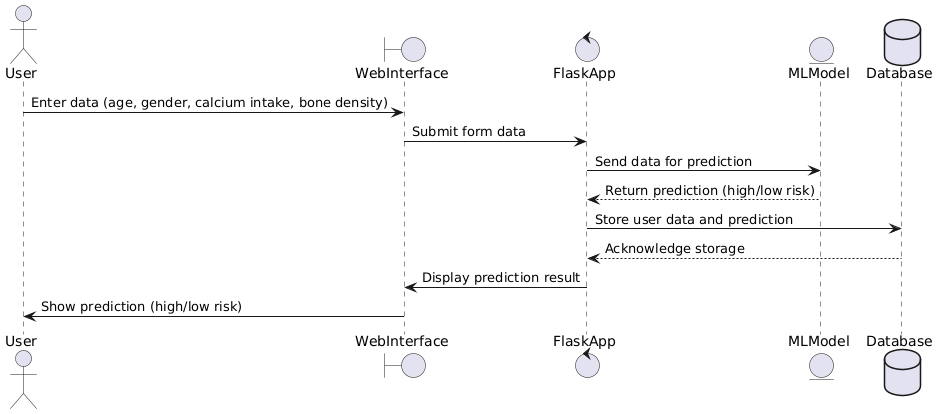
* **User to Prediction**: One-to-many relationship, where each user can have multiple predictions stored over time.



**4.3 Sequence Diagram**

The sequence diagram visualizes the flow of interactions between the user and the system during the prediction process:

1. **User Submits Data**: The user inputs their age, gender, calcium intake, and bone density, then submits the form.
2. **System Sends Data to the Machine Learning Model**: The system captures the user data and forwards it to the pre-trained machine learning model for osteoporosis risk prediction.
3. **Model Returns Prediction**: The model processes the input data and returns a prediction (high or low risk).
4. **System Stores Data and Prediction in the Database**: The system saves the input data and the prediction result in the **Predictions** table for future reference.



**4.4 Class Diagram**

The system architecture is organized into the following classes:

1. **Flask App Class**:

* **Responsibilities**: Handles the HTTP requests, routes, and rendering of web pages. It manages the user interactions like form submissions and the display of prediction results.
* **Methods**:
* predict(): Accepts user input and passes it to the model.
* show\_predictions(): Displays the history of stored predictions.

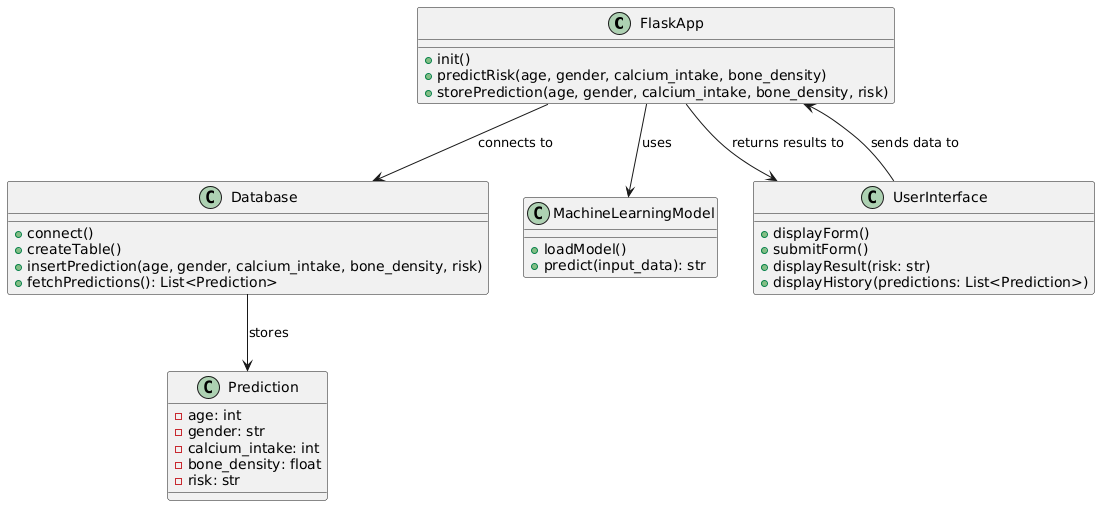
1. **Database Class**:

* **Responsibilities**: Manages the connection to the SQLite database and performs the necessary CRUD (Create, Read, Update, Delete) operations.
* **Methods**:
* save\_prediction(): Stores the user data and prediction into the **Predictions** table.
* fetch\_predictions(): Retrieves the stored predictions for display.

1. **Model Class**:

* **Responsibilities**: Loads the pre-trained machine learning model and processes the input data for osteoporosis risk prediction.
* **Methods**:
* load\_model(): Loads the pre-trained machine learning model.
* predict\_risk(): Takes user input and returns a prediction of either high risk or low risk.

These design elements ensure that the system is modular, maintainable, and capable of performing the required tasks efficiently.



**5.Implementation**

The project is implemented using Flask for the backend, SQLite for the database, and HTML/CSS/JavaScript for the frontend. The machine learning model is trained using Scikit-learn and is serialized into a .pkl file for use in the Flask app.

**5.1 Libraries Used**

1. **Flask**: A lightweight WSGI web application framework used to create the web interface.
   * **Version:** 2.x
   * **Purpose**: Flask is a lightweight web application framework that powers the backend of this application. It handles HTTP requests, defines routes, and renders dynamic HTML templates.
   * **Key Uses**:

**Routing**: Maps URLs (like /, /predict) to functions in the backend. Each route defines the behavior of the system for specific user actions.

**Form Handling**: Processes user-submitted data (age, gender, calcium intake, etc.) through HTTP POST requests and passes it to the machine learning model for risk prediction.

**Template Rendering**: Flask uses the Jinja2 templating engine to dynamically render HTML pages by injecting prediction results and other data into the frontend.

1. **SQLite**: A serverless, self-contained SQL database engine.
   * **Version:** 3.x
   * **Purpose**: SQLite is used to store and manage user inputs and predictions. It is a serverless SQL database that operates on a local file, making it easy to set up and manage without requiring external database servers.
   * **Key Uses**:

**Predictions Storage**: Stores user inputs (age, gender, calcium intake, and bone density) and the corresponding risk predictions.

**Database Operations**: The database is queried to retrieve past predictions for display on the history page, and to insert new predictions when a user submits their data.

1. **Pandas**: A powerful data analysis and manipulation library.
   * **Version:** 1.x
   * **Purpose**: Pandas is a powerful library for data manipulation and analysis, used here to process user inputs and organize them in a format suitable for machine learning models.
   * **Key Uses**:

**Data Structuring**: When the user submits data, it is loaded into a Pandas DataFrame, making it easy to manipulate and pass to the machine learning model for prediction.

**Data Processing**: Pandas allows for efficient data cleaning and formatting, ensuring that the inputs are correctly structured for the model.

1. **Numpy**: A library for numerical computation in Python.
   * **Version:** 1.x
   * **Purpose**: NumPy is a library for numerical computing, used to handle mathematical operations such as matrix and array manipulations.
   * **Key Uses**:

**Numerical Operations**: NumPy ensures efficient handling of numeric input data and model predictions, especially for matrix operations involved in machine learning models.

1. **Scikit-learn**: A library for machine learning in Python.
   * **Version:** 0.24.x
   * **Purpose**: Scikit-learn is a comprehensive library for machine learning. It was used to train the osteoporosis risk prediction model, which classifies the risk as "high" or "low."
   * **Key Uses**:

**Model Training**: Scikit-learn was used to build and train the machine learning model using a dataset of osteoporosis-related health data (age, gender, calcium intake, bone density, etc.).

**Model Serialization**: The trained model is serialized (saved) using Python's Pickle library for future use.

**Prediction**: When user inputs are submitted, Scikit-learn is used to load the pre-trained model and generate a prediction based on the input data.

1. **Pickle**: A Python library for serializing and deserializing objects.
   * **Version:** Standard Python module.
   * **Purpose**: Pickle is used for serializing and deserializing Python objects, making it possible to save the trained machine learning model as a file and load it later in the Flask app for predictions.
   * **Key Uses**:

**Model Saving**: After training the osteoporosis risk prediction model, it is saved to a .pkl file using Pickle.

**Model Loading**: When the Flask application is running, the Pickle library is used to load the saved model into memory for making predictions in real-time without retraining the model each time.

1. **Matplotlib/Seaborn**: For data visualization (if necessary for testing).
   * **Version:** 3.x / 0.11.x
   * **Purpose**: These libraries are used for data visualization. While they are not directly integrated into the core application, they are useful for visualizing data during the development and testing phases.
   * **Key Uses**:

**Dataset Visualization**: During the development phase, these libraries were used to explore and visualize the dataset (e.g., age vs. risk distribution).

**Model Evaluation**: Used to plot graphs such as confusion matrices or feature importance charts to evaluate the model's performance during training.

1. **Jinja2**: A templating engine for Python.
   * **Version:** 2.x
   * **Purpose**: Jinja2 is a templating engine used by Flask to render HTML pages. It allows dynamic content insertion into the frontend pages based on the backend data (e.g., prediction results).
   * **Key Uses**:

**Dynamic Page Rendering**: Jinja2 is used to inject data into the HTML templates. For instance, the results of predictions are dynamically inserted into the result page, and stored predictions are inserted into the prediction history page.

1. **Gunicorn**: A Python WSGI HTTP Server.
   * **Version:** 20.x
   * **Purpose**: Gunicorn is a Python WSGI HTTP server for serving the Flask application in production. It ensures that the web application can handle multiple requests concurrently, making it scalable.
   * **Key Uses**:

**Production Deployment**: Gunicorn is used to serve the Flask application on a production server. It handles incoming HTTP requests and interacts with the Flask app efficiently.

1. **OS**: Python’s standard utility library.
   * **Version:** Standard Python module.
   * **Purpose**: The OS library provides functions to interact with the operating system, such as handling file paths and directories.
   * **Key Uses**:

**File Management**: The OS library is used to locate and handle files like the .pkl model file during the loading process.

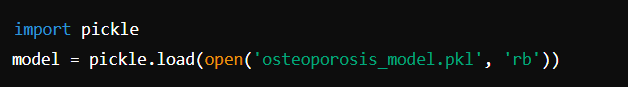
**Key Functions:**

1. **Model Loading(**pickle**)**:  
   The machine learning model responsible for predicting osteoporosis risk is stored in a serialized format using Python’s pickle library. This step allows the model, which was previously trained on historical health data, to be loaded into memory without retraining every time the Flask app runs.

**Detailed Explanation**:

* **File**: The model is saved as osteoporosis\_model.pkl.
* **Process**: When the Flask application starts, the Pickle library deserializes this file and loads the machine learning model into the application.
* **Usage**: After loading, the model is used to predict osteoporosis risk when the user submits new data.

**Code Example**:

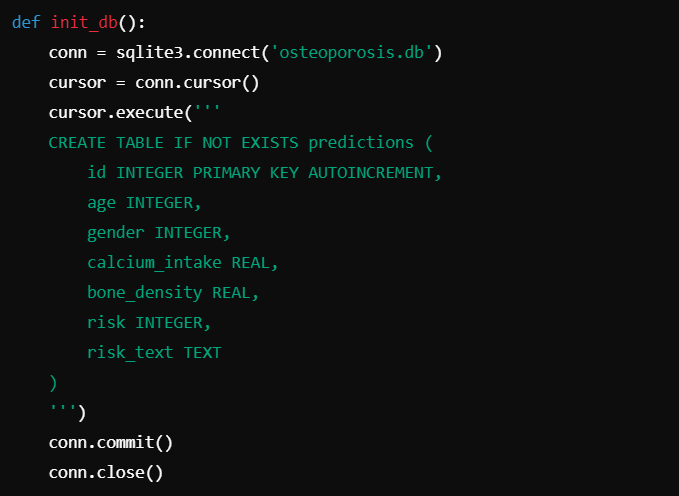


1. **Data Initialization(**init\_db**):**  
   The system uses SQLite as its database to store user inputs and prediction results. The init\_db function sets up the database by creating a table that captures user information and their corresponding osteoporosis risk prediction.

**Detailed Explanation**:

* **Database**: SQLite is a lightweight, file-based SQL database.
* **Table Structure**:
  + id: A unique identifier for each prediction record.
  + age: The user’s age.
  + gender: User’s gender, typically represented as 0 for male and 1 for female.
  + calcium\_intake: The daily calcium intake of the user in milligrams.
  + bone\_density: Bone density in g/cm², a critical factor in osteoporosis prediction.
  + risk: A numeric value indicating the predicted risk (e.g., 0 for low risk, 1 for high risk).
  + risk\_text: A textual representation of the risk (e.g., "Low Risk" or "High Risk").

**Initialization Example**:



This function is crucial for setting up the environment where user data and predictions are stored persistently.

1. **Prediction Logic**:

This part of the system handles the process of gathering user data, passing it to the model, and storing the prediction results in the database. The prediction is based on the machine learning model loaded via Pickle.

**Detailed Steps**:

1. **Data Input**: User provides their age, gender, calcium intake, and bone density through an HTML form.
2. **Model Prediction**:
   * The user data is structured into a format compatible with the model, typically a NumPy array or Pandas DataFrame.
   * This data is then fed into the pre-loaded machine learning model, which returns a numeric prediction (e.g., 0 for low risk, 1 for high risk).
3. **Result**: Based on the prediction, the result is converted into human-readable text like "Low Risk" or "High Risk."
4. **Storage**: Both the raw inputs and the prediction results (numeric and text) are stored in the SQLite database.

**Code Example**:

|  |
| --- |
| @app.route('/predict', methods=['POST'])  def predict():  age = int(request.form['age'])  gender = int(request.form['gender'])  calcium\_intake = float(request.form['calcium\_intake'])  bone\_density = float(request.form['bone\_density'])    # Prepare data for prediction  input\_data = np.array([[age, gender, calcium\_intake, bone\_density]])    # Make prediction  risk\_numeric = model.predict(input\_data)[0]  risk\_text = 'High Risk' if risk\_numeric == 1 else 'Low Risk'    # Store the prediction in the database  conn = sqlite3.connect('osteoporosis.db')  cursor = conn.cursor()  cursor.execute('INSERT INTO predictions (age, gender, calcium\_intake, bone\_density, risk, risk\_text) VALUES (?, ?, ?, ?, ?, ?)',  (age, gender, calcium\_intake, bone\_density, risk\_numeric, risk\_text))  conn.commit()  conn.close()    return render\_template('result.html', risk=risk\_text) |

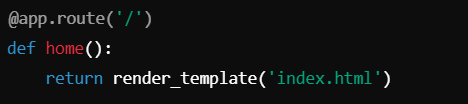
This ensures that predictions are made and stored seamlessly, providing users with immediate feedback and enabling historical data analysis.

1. **Routes**:

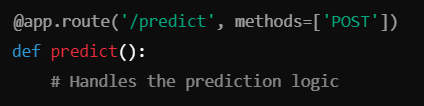
Flask routes define the various URL endpoints that the system supports. These routes handle requests, return the appropriate views (HTML pages), and manage backend logic such as predictions and database operations

**Key Routes**:

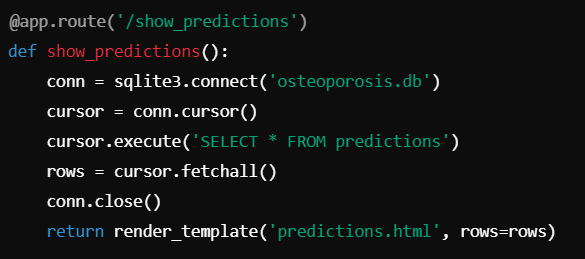
* **/:** Displays the **Home Page**.
  + **Functionality**: Introduces users to the tool and its purpose.
  + **Route Example**:



* **/predict:** Handles prediction requests.
  + **Functionality**: Receives user inputs, processes the prediction through the machine learning model, and displays the result.
  + **Route Example**:



* **/show\_predictions:** Displays the stored prediction records from the database.
  + **Functionality**: Retrieves past predictions from SQLite and presents them in a table on the web page.
  + **Route Example**:



These routes form the backbone of the application's user interaction and control flow. They handle user navigation, data submission, prediction processing, and data retrieval.

**5.1 App.py:**

from flask import Flask, render\_template, request, jsonify

import sqlite3

import pickle

import pandas as pd

app = Flask(\_\_name\_\_)

# Load the saved machine learning model

with open('osteoporosis\_model.pkl', 'rb') as f:

model = pickle.load(f)

# Initialize the SQLite database

def init\_db():

conn = sqlite3.connect('database.db')

conn.execute('''

CREATE TABLE IF NOT EXISTS predictions (

id INTEGER PRIMARY KEY AUTOINCREMENT,

age INTEGER,

gender INTEGER,

calcium\_intake REAL,

bone\_density REAL,

risk INTEGER,

risk\_text TEXT

)

''')

conn.close()

# Home route

@app.route('/')

def index():

return render\_template('index.html')

# Prediction route

@app.route('/predict', methods=['POST'])

def predict():

try:

# Capture form data

age = int(request.form['age'])

gender = int(request.form['gender'])

calcium\_intake = float(request.form['calcium\_intake'])

bone\_density = float(request.form['bone\_density'])

# Prepare the data for model prediction

input\_data = pd.DataFrame([[age, gender, calcium\_intake, bone\_density]],

columns=['age', 'gender', 'calcium\_intake', 'bone\_density'])

# Predict the risk using the loaded model

prediction = model.predict(input\_data)[0]

# Interpret the prediction

risk\_text = 'High Risk' if prediction == 1 else 'Low Risk'

# Store the result in the SQLite database

conn = sqlite3.connect('database.db')

cursor = conn.cursor()

cursor.execute(

"INSERT INTO predictions (age, gender, calcium\_intake, bone\_density, risk, risk\_text) "

"VALUES (?, ?, ?, ?, ?, ?)",

(age, gender, calcium\_intake, bone\_density, prediction, risk\_text)

)

conn.commit()

conn.close()

# Return the prediction as JSON response

return jsonify({'prediction': risk\_text})

except Exception as e:

return jsonify({'error': str(e)})

# Route to show stored predictions

@app.route('/show\_predictions')

def show\_predictions():

try:

conn = sqlite3.connect('database.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM predictions")

rows = cursor.fetchall()

conn.close()

return render\_template('show\_predictions.html', rows=rows)

except Exception as e:

return str(e)

# Initialize the database on startup

if \_\_name\_\_ == '\_\_main\_\_':

init\_db()

app.run(debug=True)

**6. User Screens**

1. **Home Screen:** Provides a welcome message and project information.

The Home Screen introduces the user to the application, explaining its purpose and providing navigation options to access various features such as making a prediction or viewing previous predictions.

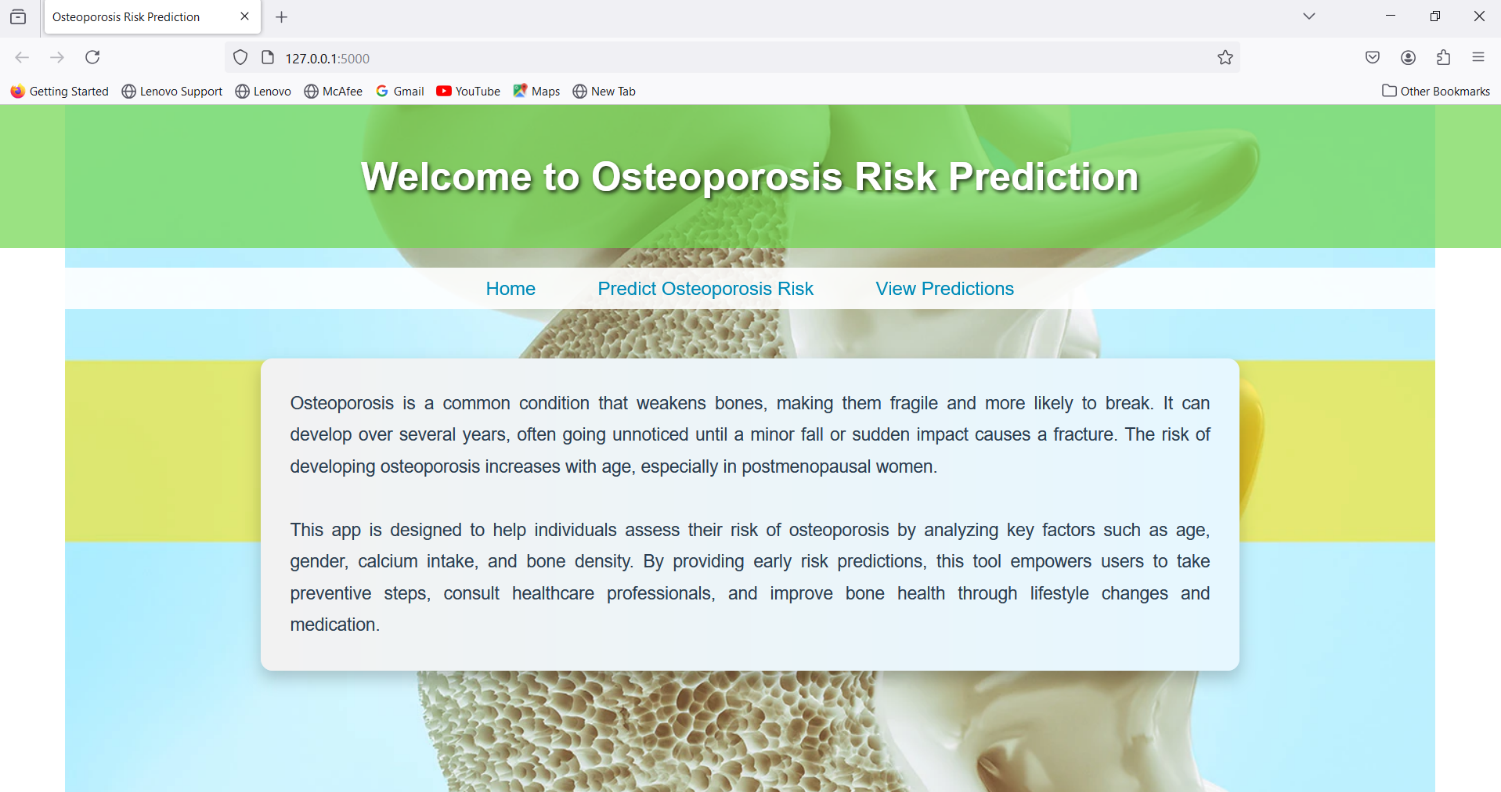
**Elements**:

* A welcoming **message** (e.g., "Welcome to the Osteoporosis Risk Prediction Tool").
* **Project Information**: A brief explanation of osteoporosis, its risks, and the purpose of the prediction tool.

**Navigation**:

1. A link or button to the **Prediction Screen** (e.g., "Predict Osteoporosis Risk").
2. A link or button to the **View Predictions Screen** (e.g., "View Previous Predictions").

**Visual Elements**: Attractive background images related to health or osteoporosis, with easy-to-read font and clear navigation buttons.



1. **Prediction Screen:**

This screen allows users to input personal data and receive a prediction of their osteoporosis risk.

**Form Fields**:

* **Age**: A numerical input field where the user enters their age.
* **Gender**: Radio buttons for selecting gender (Male or Female).
* **Calcium Intake**: A dropdown menu that allows the user to select their daily calcium intake from predefined ranges (e.g., "<500 mg/day", "500-1000 mg/day", ">1000 mg/day").
* **Bone Density**: A dropdown menu with various bone density ranges, allowing users to input their bone density in g/cm².

**Submit Button**:

* **Predict**: A button that submits the data for evaluation by the machine learning model. Once clicked, the system processes the data and provides a prediction.

**Result Display**:

After the prediction is complete, the screen dynamically shows the result. For example:

1. "You are at **High Risk** of developing osteoporosis."
2. "You are at **Low Risk** of developing osteoporosis."

This result is shown on the same screen after the user submits their data.

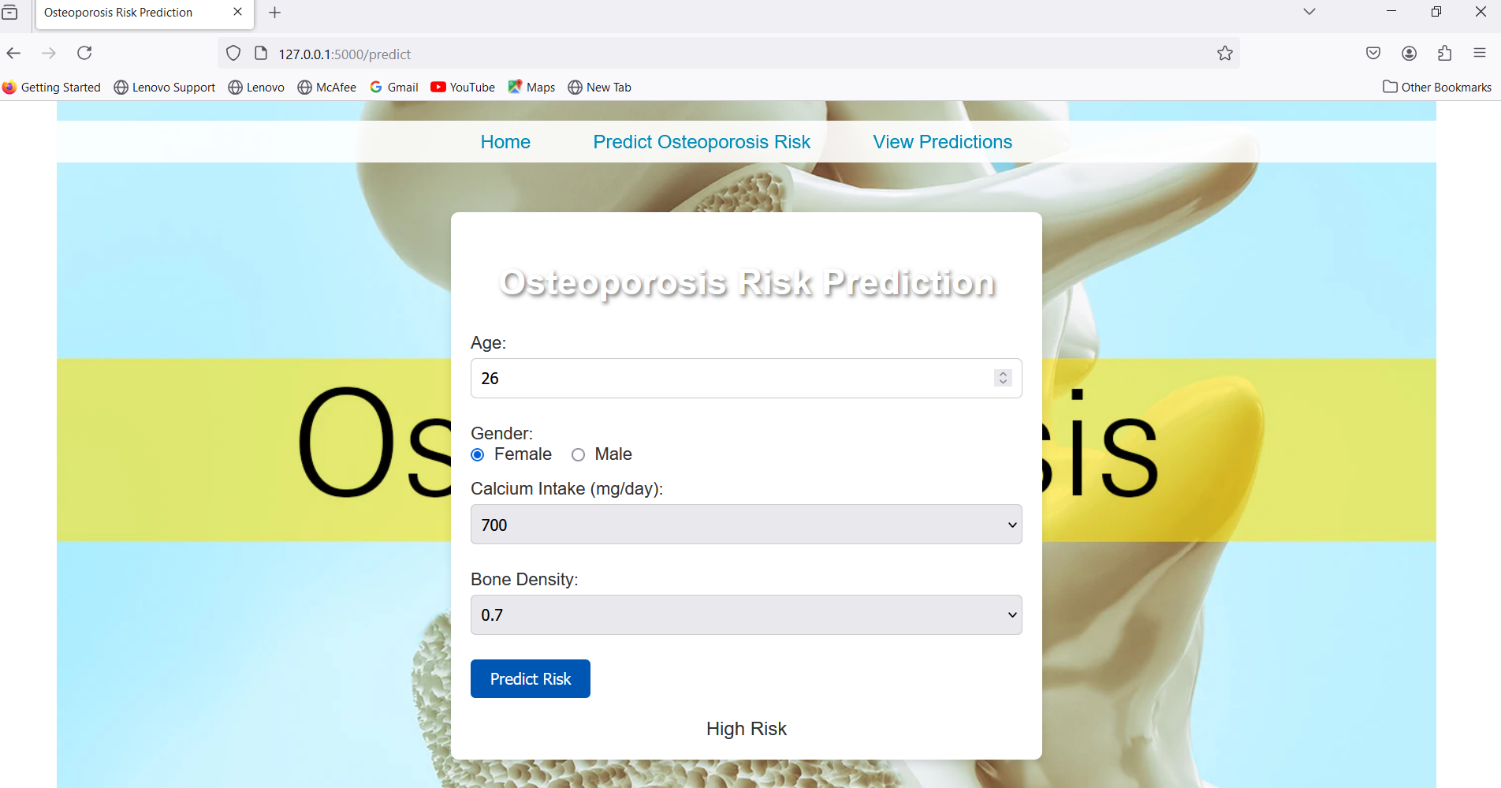
**Data Validation**:

**Form Validation**: Ensures that all fields are filled in before the user can submit the form. If any required fields are missing, the system will notify the user with an error message (e.g., "Please fill in all required fields").

**Input Constraints**: Age should be a valid numerical value, and gender, calcium intake, and bone density must be selected from their respective options.

**Example Workflow**:

1. User enters their **age**.
2. Selects their **gender**.
3. Chooses their **calcium intake** from the dropdown.
4. Chooses their **bone density** from the dropdown.
5. Clicks **Predict**.
6. The screen shows the **risk prediction**: "High Risk" or "Low Risk."



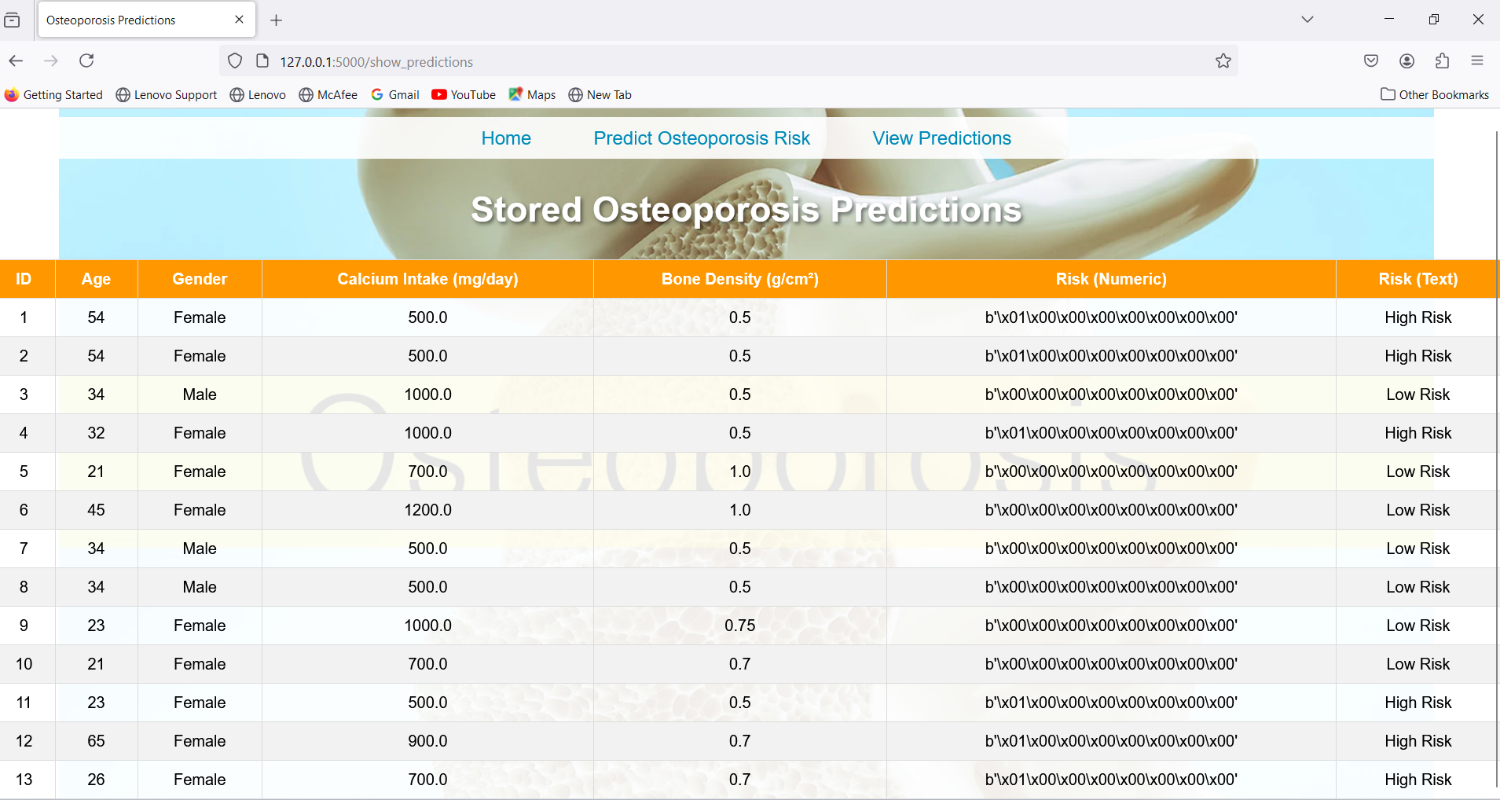
1. **View Prediction Screen:** Displays previous predictions stored in the database.

This screen allows users to view the history of their previous osteoporosis risk predictions.

A table that shows the user's previously entered data (age, gender, calcium intake, bone density) along with the corresponding risk prediction.

Each row represents a previous prediction, showing:

1. **ID** (automatically generated by the system).
2. **Age** (as entered by the user).
3. **Gender** (converted to Male/Female based on input).
4. **Calcium Intake** (the selected range of intake).
5. **Bone Density** (the selected bone density range).
6. **Risk (Numeric)**: A numerical representation of risk (0 for low risk, 1 for high risk).
7. **Risk (Text)**: A more user-friendly textual representation of risk ("Low Risk" or "High Risk").



**7. Testing Methodology**

1. **Unit Testing**

Unit testing is a fundamental approach used to validate individual functions or components of the application. Each function is tested in isolation to verify that it produces the expected outputs given specific inputs.

* **Flask Routes Testing**: Flask routes are responsible for handling HTTP requests and returning the correct responses. Unit tests ensure that each route (URL endpoint) in the Flask application behaves as intended.

*Example*: Testing the /predict route with different scenarios such as valid user input, missing or incomplete data, and invalid data types. The test ensures that: The correct status codes (200 for success, 400 for errors) are returned. The prediction response (high/low risk) is accurate. Proper error messages are provided when necessary.

* **Machine Learning Model Testing**: The machine learning model is a core component, so its predictions need to be consistent and accurate. Unit tests validate that the model produces correct risk predictions.

*Example*: Feeding known input data into the model, such as a user with high bone density and calcium intake, should consistently yield a low-risk prediction (0). Similarly, input data from a user with poor bone density should result in a high-risk prediction (1). The model’s behavior on edge cases, such as extremely high or low values, is also verified.

* **Input Validation Testing**: This ensures that user inputs are correctly validated before being passed to the prediction logic.

*Example*: Test cases are designed to confirm that numerical inputs like age (e.g., negative numbers or excessively high values) and calcium intake (e.g., strings instead of numbers) are properly validated, with appropriate error messages shown when inputs are invalid. For instance, the system should reject negative ages and provide a relevant error message, while only accepting positive numbers within a realistic range.

**II. Integration Testing**

Integration testing focuses on verifying that different parts of the application interact properly with one another. This type of testing ensures that when components such as the frontend, backend, and database are integrated, they work smoothly together.

* **Web Interface to Flask App**: The connection between the user interface (frontend) and the Flask backend is tested to ensure that form data is passed correctly.

*Example*: When a user submits their information (age, gender, calcium intake, and bone density) via the form on the prediction screen, the data should be properly received by the Flask backend. The test would simulate a user submitting data and verify that the backend processes the form without errors and generates a prediction that is correctly displayed.

* **Flask App to Machine Learning Model**: This test confirms that the Flask backend correctly sends the processed input data to the machine learning model for prediction.

*Example*: During testing, simulated inputs (like age and calcium intake) are passed to the backend. The test ensures that the inputs are correctly formatted and passed to the machine learning model, and that the predicted risk is returned without errors. This helps verify the correctness of the data flow between the backend and the model.

* **Flask App to Database**: Integration tests are also performed to ensure that the Flask application correctly interacts with the SQLite database, including storing and retrieving data.

*Example*: After a user submits a prediction, the system should store the user’s data (age, gender, calcium intake, bone density) along with the prediction in the database. The test would involve submitting data and then querying the database to ensure the data is correctly stored and can be retrieved and displayed in the prediction history section.

**III. Database Testing**

Database testing verifies the proper creation, manipulation, and retrieval of data within the SQLite database. Since user inputs and predictions are stored in the database, testing ensures that all data is stored accurately and remains consistent over time.

* **Table Creation Testing**: Tests are conducted to verify that the correct tables (e.g., predictions) are created in the database with all the necessary fields.

*Example*: The test checks whether the predictions table has fields such as age, gender, calcium\_intake, bone\_density, and risk, ensuring that the schema matches the expected structure. This is crucial to avoid errors during data insertion.

* **Data Insertion Testing**: This ensures that after a user submits their data, it is correctly inserted into the database. The test also checks whether the right data types are used for each field.

*Example*: After submitting data through the user interface, the test would query the predictions table to confirm that the input values (age, gender, calcium intake, bone density) and the corresponding risk prediction are correctly stored. For instance, testing that age is stored as an integer and risk as a binary value (0 or 1).

* **Data Retrieval Testing**: Tests are run to ensure that stored data can be accurately retrieved from the database for display in the user’s prediction history.

*Example*: After a prediction is stored, the test checks whether the data is retrieved properly for display in the "View Predictions" screen. This ensures the system can retrieve all records, display them accurately, and handle cases where there may be no data to display (empty database).

**8. Conclusion and Future Enhancements**

The **osteoporosis risk prediction system** has been successfully developed to provide a reliable and accessible tool for predicting osteoporosis risk. The project utilizes a machine learning model, trained on significant factors such as **age, gender, calcium intake, and bone density**, to predict whether a user is at high or low risk of developing osteoporosis. The system integrates a user-friendly interface where individuals can input their data and instantly receive predictions. Additionally, the use of an **SQLite database** ensures that each prediction is stored securely, allowing users to view and analyze their past health trends.

One of the key accomplishments of this project is its demonstration of how **data science and machine learning** can positively impact healthcare. By providing real-time predictions, users can take proactive measures in managing their bone health. This project underscores the importance of early detection in reducing the likelihood of fractures and other complications associated with osteoporosis. The straightforward design of the system makes it accessible to a wide range of users, offering them a practical tool for monitoring their health.

The use of a machine learning model has been effective in classifying risk into two categories: high and low. This project highlights the potential for leveraging data science and machine learning to improve healthcare, enabling early detection of health risks and empowering individuals to take preventive measures.

Future enhancements could include:

* Adding more data points like physical activity, genetics, and diet.
* Enhancing the machine learning model for better accuracy.
* Implementing a mobile version of the application for broader accessibility.

These enhancements would improve the usability, accuracy, and scalability of the application, making it a valuable tool for both individuals and healthcare professionals in the early detection and management of osteoporosis risk.

**9. References**

1. World Health Organization (WHO) – Osteoporosis Facts  
   *World Health Organization*. (2019). "Osteoporosis." Retrieved from [https://www.who.int](https://www.who.int/news-room/fact-sheets/detail/osteoporosis).
2. Journal of Machine Learning Research, 12, 2825-2830. Retrieved from <https://scikit-learn.org>, <https://pandas.pydata.org>. , <https://flask.palletsprojects.com> , <https://www.sqlite.org>. ,
3. **Osteoporosis Prediction Using Machine Learning Algorithms**  
   Singh, A., Gupta, S., & Kumar, R. (2020). "Predicting the Risk of Osteoporosis Using Data Mining Techniques." International Journal of Medical Engineering and Informatics. DOI: 10.1504/IJMEI.2020.10032672.

Top of Form

Bottom of Form