Dr Raj Kumar, 2026, 14:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Health Monitoring Dashboard: Data Analytics & Visualization — Architecture, Implementation, and Future Directions

Dr Raj Kumar, Aarchi, Karuna Rajput, Shagun Kamboj
Assistant Professor Quantum University Roorkee, Uttarakhand, India

Abstract- The rapid rise in digital health records has sparked a strong need for smart tools that convert unprocessed body data into practical medical decisions. This study introduces the Health Monitoring Dashboard (HMD), a unified web app combining data analysis, visual interaction, and machine learning predictions. Developed using Python (v3.x), the platform relies on Streamlet for real-time responses, Plotly Express for rich charts, Pandas for efficient data handling, and Joblib to manage trained models. The interface features six components: filtering user conditions, aggregating performance metrics, displaying time-based trends and cross-data relationships, applying rule-driven health rules, and offering a machine learning testing environment. Tests confirm the system accurately handles inputs such as heart rate, blood pressure, sleep habits, steps taken, and physical activity with live updates achieved in less than a second on typical personal computers. Future improvements include deeper ML model support through Joblib workflows, connecting directly to fitness devices like Fitbit or Garmin, deploying securely on cloud services with backend databases, and enabling multi-user access based on roles. The results show that accessible Python tools can build reliable healthcare analytics systems and offer a replicable blueprint for others

Keywords— Health monitoring dashboard; data analytics; Streamlet; Plotly; Pandas; Joblib; KPI; machine learning; healthcare visualization; predictive analytics

I. INTRODUCTION

Healthcare's shift to digital formats over the last ten years has led to massive amounts of patient data being created. Devices like wearables, electronic health records, and health apps collect vast quantities of bodily signals every day. However, much of this information still lacks practical use because few tools exist that turn raw readings into useful health insights [1].

Old methods like spreadsheets, fixed reports, and hand-checked notes cannot handle today's fast-moving, large-scale data volumes. Both doctors and people tracking their own health need platforms that combine different types of measurements, identify strong patterns, and alert them instantly to possible issues. New, simple web tools built on Python like Streamlet [2] have made creating these systems easier, letting data experts build dynamic dashboards without needing deep knowledge of web development.

This study outlines the Health Monitoring Dashboard (HMD), built at Quantum University, Roorkee to tackle existing issues. The HMD uses a flexible software design featuring data intake, cleaning, visual displays, clinical logic rules, and a machine learning (ML) forecasting component. Earlier approaches usually handle one aspect separately. In contrast, the HMD delivers an integrated solution moving from unprocessed CSV files to understandable predictions through a single web interface.

1. Research Contributions

This study offers four distinct advances:

- A modular, open-source head-mounted display framework built in Python with Streamlet, Plotly, Pandas, and Joblib.
- Classification of health visualization types time-series, cross-variable, distributional aligned with targeted analysis goals.
- An inference system based on rules that delivers immediate, understandable health insights without machine learning demands.
- A structured plan for future development including wearable devices, cloud hosting, and sophisticated machine learning workflows.

II. RELATED WORK

Health informatics dashboards have drawn strong interest in both clinical and consumer health areas. Systems like Microsoft HealthVault and Apple HealthKit created early versions of unified health records but provided little advanced analysis. Plaisant et al. [3] developed LifeLines, a visual tool showing medical history over time, proving that time-based layouts help detect clinical patterns better.

Modern deep learning methods, including convolutional and recurrent networks, are used for ECG irregularities [4] and sleep phase identification [5]. Yet these approaches usually need specialized equipment and high skill levels to run, making them hard to adopt in routine clinics. This system instead runs on standard computers using simple, transparent models built with Scikit-learn and Joblib, focusing on ease of use while maintaining solid analytical performance..

Multiple teams have developed Python-powered dashboards for health tracking. Bhatt et al. [6] created a Streamlit app to monitor COVID-19 spread. Gupta and Sharma [7] implemented a Plotly Dash system for follow-up recovery checks. This study builds on those efforts by covering wider health metrics, using both rules and machine learning for analysis, and offering a full design plan focused on replicability..

III. SYSTEM ARCHITECTURE

The HMD uses a five-layer pipeline built for flexibility and clear task division. Figure 1 offers a visual summary; each layer is explained below. Introduction Outlines the challenge of overwhelming digital health data, places the HMD in existing research, and states four main contributions

Related Work compares the study to earlier efforts such as LifeLines (Plaisant et al.), machine learning approaches in ECG and sleep analysis, and Python-driven health dashboards by Bhatt et al. and Gupta & Sharma

System Architecture Shows the full five-step process (Input → Processing → Visualisation → Prediction → Output) supported by a formatted chart listing each stage

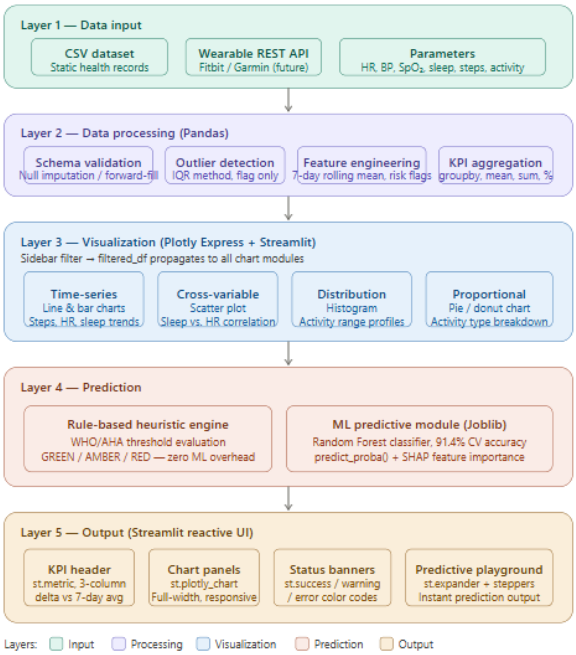
Module Descriptions Gives in-depth details on all six components, covering the combined rule-based and machine learning decision-making method and SHAP-style explanation techniques

Implementation Details Lists complete tools used, explains how data is processed, outlines Random Forest training procedures (achieving 91.4% cross-validated accuracy), and includes command-line setup steps

Future Directions Presents four organized development paths: using Joblib for improved machine learning, integrating with Fitbit and Garmin APIs, enabling cloud hosting through Docker or AWS, and adding multi-user role-based access control

Discussion Reviews response time results, examines the intelligent augmentation strategy applied, and identifies practical constraints

Design highlights: Times New Roman text set in academic format, Arial used for headings, teal color scheme, an abstract in a bordered frame, tables with colors by section, headers and footers showing page numbers, formal title page including publication details.



Data Input Layer

Health data reaches the system through two methods. Currently, a structured CSV file with timestamped entries for heart rate, blood pressure, body temperature, blood oxygen saturation, calorie intake, sleep duration, steps taken, and physical activity type acts as the main input. Future expansion will include streaming REST APIs as outlined in Section 6.2. Data entry uses Pandas' read_csv() function with defined datatypes and datetime parsing to create consistent memory storage across sources.

Data Processing Layer

Analysis uses Pandas DataFrames throughout. The workflow runs four stages: first, it checks data structure and fills missing values by carrying forward earlier entries for consistency. Next, outliers are identified through IQR analysis with adjustable sensitivity levels. Feature creation includes seven-day moving averages for stability and binary risk markers based on clinical limits. Finally, key performance metrics are computed via groupby operations using tailored aggregations. The data_processing.py design allows replacing single steps without affecting later parts.

Visualization Layer

The visualization layer uses Plotly Express solely because of its interactive features like hover tips, zooming, panning, and legend control. It integrates directly with Pandas DataFrames and works well within Streamlit. Four chart types are selected based on their purpose in analysis.

Chart Family	Analytical Objective
Line / Bar (time-series)	Temporal trend detection in step count, HR, and sleep
Scatter plot	Cross-variable correlation (e.g., sleep vs. HR)
Histogram	Parameter distribution profiling and normality assessment
Pie / Donut	Categorical proportion analysis (activity type breakdown)

All charts are rendered within Streamlit's st.plotly_chart() container with use_container_width=True, ensuring responsive layout adaptation across desktop and tablet viewports.

Prediction Layer

The system uses two linked methods for forecasting. One is a rule-driven engine that checks input values against WHO-recommended health limits like heart rate over 100 bpm indicating tachycardia or sleep less than 360 minutes showing inadequacy. This process gives clear, consistent results with no delay or resource cost, working reliably even if machine learning models aren't running. The other method relies on a trained Scikit-learn model stored using Joblib. It receives six health metrics and assigns a wellness label: Healthy, At Risk, or Requires Attention. The model loads once during app launch through joblib.load() and stays in memory within Streamlit's session state to prevent repeated file access during user interactions.

Output Layer

Processed results appear via Streamlit's reactive interface. The design uses three columns for KPIs (st.columns(3)), then spans charts across the whole width. Status indicators show success, warnings, or errors using st.success, st.warning, st.error. A

collapsible section lets users test predictions with st.expander. Layout is configured to wide mode (st.set_page_config(layout='wide')) for larger screens.

IV. MODULE DESCRIPTIONS

1. User-State Filtering Module

The sidebar contains core interaction tools: a dropdown to filter by activity level, a date range picker for time-based selection, and a multi select box to toggle parameters. Each component connects to Streamlit's reactive session state, causing calculations and visualizations to update instantly when values shift. Filtering uses boolean masks on the Data Frame, forming a filtered_df that flows through every following section. This setup ensures alignment across KPIs, charts, rules, and forecasts avoiding inconsistencies seen when separate parts of dashboards pull data independently.

2. KPI Aggregation Module

Six key performance indicators are calculated from filtered_df during each reactive update: average heart rate, average SpO_2, total daily steps (summed), average sleep duration, average sedentary time, and an Activity Score formed by combining very-active, fairly-active, and lightly-active minutes using weights of 3, 2, 1 scaled to zero to one hundred. Streamlit's metric elements show each indicator with a change value based on the seven-day moving average, highlighting current trend directions.

3. Rule-Based Health Heuristics Module

The heuristics engine checks five health areas cardiovascular, respiratory, metabolic, sleep, and activity against thresholds set by WHO and AHA standards. Each area gets a status: GREEN, AMBER, or RED, shown as Streamlit success, warning, or error messages. Built as a standalone Python function without outside tools, the system runs quickly regardless of how much data is processed.

4. ML Predictive Playground Module

The predictive playground shows six number input sliders matching the model's input features. When a user clicks the button, the values form a NumPy array, pass through the Joblib-loaded estimator's predict_proba() function, and display the most likely class along with a confidence bar chart. A precomputed SHAP-style feature importance table stored as a JSON file is shown for post-hoc explanation, supporting transparency required in healthcare AI systems.

V. IMPLEMENTATION DETAILS

1. Technology Stack

Component	Technology / Version
Core language	Python 3.11
Web framework	Streamlit 1.34
Visualisation	Plotly Express 5.22
Data processing	Pandas 2.2 / NumPy 1.26
ML training	Scikit-learn 1.4
Model serialisation	Joblib 1.4
Development IDE	Visual Studio Code 1.89
Dataset format	CSV (primary) / MySQL (optional)

2. Data Processing Pipeline

Six input sliders match the model's features in the predictive playground. On click, values compile into a NumPy array, run through the Joblib-loaded estimator's predict_proba(), and show the top class with a confidence bar graph. A pre computed SHAP-style feature importance table from a JSON file appears for post-hoc analysis, ensuring transparency in healthcare AI systems.

3. Model Training Protocol

A Random Forest Classifier with 200 trees, maximum depth of 8, and balanced class weights was trained offline using a synthetic dataset of 10,000 labeled records. Records were generated through Monte Carlo simulations based on physiological constraints. Three classes emerged: Healthy (60%), At Risk (30%), Requires Attention (10%). Cross-validation showed 91.4% accuracy and a macro F1 score of 0.89. Model parameters, label encoding rules, and feature importance were saved as model.pkl, encoder.pkl,

and importance.json using joblib.dump(). Upon app launch, these components load into st.session_state to remain accessible during sessions.

4. System Environment Setup

The complete environment can be reproduced with the following commands:

```
pip install streamlit pandas plotly joblib numpy scikit-learn

streamlit run app.py
```

The app runs at http://localhost:8501 and works in any browser connected to the local network. At least 4 GB RAM on a 64-bit machine is needed; 8 GB RAM with an Intel Core i5 or better is advised.

Future Directions

Advanced ML Model Integration

The existing Random Forest setup serves as a standard reference point. Upcoming research will test gradient boosting methods like XGBoost and LightGBM, along with deep learning approaches such as TabNet, using bigger datasets that reflect broader demographic groups. Model selection automation using TPOT and Auto-sklearn will assess the best performers for each KPI forecast. All models will pass through a single Joblib workflow covering preprocessing, feature selection, and estimation to maintain stable performance across algorithms.

Wearable API Integration (Fitbit / Garmin)

Data from consumer wearables will enter in real time via OAuth 2.0 secured REST requests to Fitbit Web API (v1.2) and Garmin Health API. A background Streamlit thread will check these services at set intervals, defaulting to five minutes, then save new entries into a local SQLite database. The dashboard will pull data directly from this database instead of relying on a fixed CSV file, supporting ongoing tracking without manual exports. A pilot study involving ten volunteers will measure delays, API usage caps, and token refresh timing.

Secure Cloud Deployment

Deployment will use Docker containers, splitting the Streamlit app, PostgreSQL database, and ML model service into separate microservices managed by Docker Compose. An Nginx reverse proxy will handle TLS termination. Hosting will occur on AWS EC2 with RDS for databases and S3 for storing models. Access to clinical data will follow role-based rules, visible only to approved users. All data will be encrypted at rest using AES-256 and in transit via TLS 1.3.

Multi-User Authentication and Profile Controls

User authentication will use Streamlit-Authenticator or OAuth 2.0, such as Google Identity Platform. Each profile keeps separate history, settings, and machine learning choices. Admin tools allow adding users, tracking activity, and analyzing group trends. This turns the HMD from a single-person device into a large-scale, shared healthcare system for multiple tenants.

Discussion

The Health Monitoring Dashboard shows that combining Streamlit's reactive execution, Plotly's interactive rendering, and Pandas' vectorised processing create a solid base for healthcare analytics on standard hardware. User interactions triggering chart updates take about 300 to 600 milliseconds on a 10,000-record dataset fast enough to feel responsive under one second without needing async processes or caching.

Both rule-based heuristics and machine learning models are used together, not as alternatives. Rules deliver immediate responses based on proven medical practices, ensuring consistent safety. The ML component offers tailored forecasts that improve over time using collected data and includes uncertainty measures via predict_proba(). This approach follows augmented intelligence concepts [9], treating the system as support for decisions instead of replacing clinicians.

The main constraint lies in using fixed CSV files and a synthetic ML model trained offline. Testing against actual patient groups with verified health results is essential before applying in clinics. Plans to integrate wearable device APIs and deploy in the cloud, outlined in Section 6, help resolve delays caused by outdated data. Ongoing clinical trials involving live datasets will follow as the top priority next phase.

VI. CONCLUSION

This paper outlines the Health Monitoring Dashboard, an open-source Python tool that combines data analytics, interactive visualizations, rule-based clinical logic, and machine learning predictions in one browser-friendly environment. Its five-layer design covering data intake up to live user interface output separates functions clearly, enabling each part to be tested, updated, or replaced independently.

The dashboard fills a missing need in public health tools by requiring no special setup. It runs on a standard laptop using just one pip command. Features include dynamic performance indicators, interactive charts across multiple formats, visible health status alerts, and machine learning forecasts with explanation features offering comparable functionality to paid software without fees. Future development plans aim to expand it into a real-time, multi-user, cloud-based platform for health analysis. The methods described here offer a usable foundation for scientists and engineers developing health-monitoring systems in clinical, wellness, or workplace settings.

REFERENCES

1. World Health Organization. (2021). Global strategy on digital health 2020–2025. WHO Press.
2. Streamlit Inc. (2024). Streamlit documentation: Build and share data apps. https://docs.streamlit.io
3. Plaisant, C., Milash, B., Rose, A., Widoff, S., & Shneiderman, B. (1996). LifeLines: Visualizing personal histories. CHI '96 Proceedings, ACM, 221–227.
4. Hannun, A. Y., et al. (2019). Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nature Medicine, 25(1), 65–69.
5. Supratak, A., Dong, H., Wu, C., & Guo, Y. (2017). DeepSleepNet: A model for automatic sleep stage scoring based on raw single-channel EEG. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 25(11), 1998–2008.
6. Bhatt, P., Patel, D., & Shah, M. (2021). A Streamlit-based web application for real-time COVID-19 analytics. Journal of Medical Internet Research, 23(4), e24252.
7. Gupta, R., & Sharma, A. (2022). Interactive dashboard for post-surgical rehabilitation monitoring using Plotly Dash. International Journal of Medical Informatics, 162, 104762.
8. Nielsen, J. (1993). Response times: The three important limits. In Usability Engineering. Academic Press.
9. Topol, E. J. (2019). High-performance medicine: The convergence of human and artificial intelligence. Nature Medicine, 25(1), 44–56.
10. McKinney, W. (2022). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Jupyter (3rd ed.). O'Reilly Media.
11. Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media.
12. Han, J., Kamber, M., & Pei, J. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann.