Project Report — Health Monitoring Dataset

1. Cover Page

NAME:PRATEEK KUMAR PANDEY(25BSA10084)

Project Title: Health Monitoring System
Course: — Introduction to problem solving
Date: November 24, 2025
Dataset: Health Monitor Dataset (`/mnt/data/dataset_extract/Health Monitor Dataset.xlsx`)

2. Introduction

Continuous monitoring of physiological vital signs (heart rate, respiratory rate, body temperature, blood pressure, SpO_2, etc.) enables early detection of clinical deterioration. This project builds a pipeline for data ingestion, cleaning, exploratory analysis, and models to (a) classify a patient's current state (normal vs. abnormal) and/or (b) predict near-future deterioration. The final deliverable is a prototype system capable of real-time alerting and dashboarding.

---

3. Problem Statement

Design and implement a system that:

- Processes streaming or batch vital-signs data,
- Detects abnormal physiological states,
- Issues early alerts for possible clinical deterioration,
- Provides explainable model outputs and supports clinicians with interpretable features.

Primary ML task(s):

- **Classification**: normal vs abnormal / multi-class diagnosis.
- **Time-series forecasting** (optional): predict next-hour risk score.

Success metrics:

- Classification: Accuracy, Precision, Recall, F1-score, AUROC.

- Forecasting: RMSE, MAE, and lead-time for correct early alerts.

---

4. Functional Requirements

1. **Data ingestion:** Read Excel/CSV files or stream data from devices.
2. **Preprocessing:** Handle missing values, timestamps, unit normalization.
3. **Feature engineering:** Rolling stats (mean, std), HRV features, SpO_2 trends.
4. **Modeling:** Train classification and optional forecasting models.
5. **Alert generation:** Threshold / model-based alerting with severity levels.
6. **Visualization:** Dashboard with time-series plots and patient summary.
7. **Explainability:** SHAP/LIME explanations for alerts.
8. **Export/Reporting:** Save model, logs, and export patient reports as PDF/Word.
9. **User roles:** Clinician (view/acknowledge alerts), Admin (manage models).

---

5. Non-functional Requirements

- **Performance:** Real-time ingestion latency < 5 seconds for streaming data path.
- **Scalability:** Able to handle thousands of patient streams (horizontal scaling).
- **Reliability:** Fault tolerant ingestion and persistent logs.
- **Security & Privacy:** Data encryption at rest/in transit; follow HIPAA/GDPR-type best practices.
- **Maintainability:** Modular codebase with unit tests and CI pipeline.
- **Usability:** Clean dashboard, mobile friendly, clear alert messages.

---

6. System Architecture

High-level components:

1. **Data Sources:** Wearables, bedside monitors, uploaded files (Excel/CSV).
2. **Ingestion Layer:** API endpoints / message queue (e.g., Kafka) to accept data.
3. **Preprocessing Service:** Normalization, cleaning, outlier handling.
4. **Feature Store:** Time-windowed features, cached for quick access.
5. **Model Serving:** REST API serving classification and forecasting models.
6. **Alert Engine:** Receives model scores, applies rules, sends notifications (email/SMS/UI).
7. **Dashboard/UI:** Visualize vitals, trends, alerts, and explanations.
8. **Storage:** Relational DB for metadata, time-series DB for vitals (e.g., InfluxDB), object storage for logs.
9. **Monitoring & Logging:** System health and model drift monitoring.

(If building locally: ingestion → pandas pipeline → scikit-learn model → Flask API → simple web UI.)

---

7. Design Diagrams

Below are descriptions you can use to create diagrams (tools: draw.io, Lucidchart, or any UML tool).

Use Case Diagram

Actors: Clinician, System Admin, Sensor Device, Patient
Use cases: Upload data, Monitor patient, Acknowledge alert, Configure thresholds, View reports.

Workflow Diagram

1. Sensor -> Ingestion -> Preprocessing -> Feature Extraction -> Model -> Alert Engine -> Clinician Dashboard.
2. Periodic retraining workflow: Data store -> Data labeling -> Train -> Validate -> Deploy.

Sequence Diagram (for alert generation)

1. Device sends reading -> Ingestion service -> Preprocessing -> Model scoring -> Alert Engine -> Push notification -> Dashboard update -> Clinician acknowledges.

Class / Component Diagram

- `IngestionService`, `Preprocessor`, `FeatureExtractor`, `ModelServer`, `AlertManager`, `DashboardUI`, `DBConnector`.

ER Diagram (storage)

- `Patient(patient_id, name, dob, demographics)`
- `Reading(reading_id, patient_id, timestamp, heart_rate, spo2, temp, sbp, dbp, resp_rate, ... )`
- `Alert(alert_id, patient_id, timestamp, severity, model_score, status)`
- `ModelVersion(version_id, name, metrics, created_at)`

---

8. Design Decisions & Rationale

- **Model choice:** Start with interpretable models (Logistic Regression, Random Forest). Use XGBoost/LightGBM if more performance needed. LSTM/Temporal CNN for sequence forecasting if temporal patterns dominate.
- **Feature windows:** Rolling windows (5m, 15m, 1h) capture immediate vs. trend features.
- **Alerting strategy:** Combine rule-based (clinically validated thresholds) with model-based risk scores to reduce false positives.

- **Storage:** Time-series DB for efficient retrieval; RDBMS for metadata.
- **Explainability:** Use SHAP values so clinicians can see feature contributions.

---

9. Implementation Details

Environment & tools

- Python 3.10+, pandas, NumPy, scikit-learn, XGBoost, tsfresh (optional), statsmodels, Flask/FastAPI, Plotly/Dash or Streamlit for dashboard.

Data loading (example)

Dataset path: `/mnt/data/dataset_extract/Health Monitor Dataset.xlsx`

```
import pandas as pd
df = pd.read_excel("/mnt/data/dataset_extract/Health Monitor Dataset.xlsx")
```

Preprocessing steps

1. Parse timestamps, set timezone if needed.
2. Sort by patient_id & timestamp.
3. Handle missing values:
    - Short gaps: forward/backward fill within a limit.
    - Long gaps: flag and ignore for model training.
4. Outlier removal: clamp physiologically impossible values (HR < 20 or > 300).
5. Unit normalization: ensure SpO_2 in %; temperature in °C.
6. Resample to uniform interval (e.g., 1-min) with aggregation.

Feature engineering

- Instant features: current HR, SpO_2, temp.
- Trend features: rolling mean/std over 5/15/60 minutes.
- Derived: pulse pressure (SBP-DBP), HRV approximations (if RR intervals available).
- Time features: hour-of-day, day-of-week.

Modeling

- **Baseline classifier:** Logistic Regression (features standardized).
- **Tree-based:** Random Forest / XGBoost for performance.
- **Temporal model (optional):** LSTM or Temporal Convolution for forecasting.
- **Evaluation:** 5-fold cross validation grouped by patient_id to avoid leakage.

Example training snippet

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
```

```
X, y = features, labels
clf = Pipeline([
    ("scaler", StandardScaler()),
    ("rf", RandomForestClassifier(n_estimators=200, random_state=42))
])
clf.fit(X, y)
```

Model explainability

- Compute SHAP values for tree models to show feature importance per-prediction.

---

10. Testing Approach

- **Unit tests:** For preprocessing functions (missing fill, resampling).
- **Integration tests:** End-to-end pipeline on a small sample.
- **Model validation:** Grouped cross-validation to avoid patient-level leakage.
- **Stress testing:** Simulate high-volume ingestion and ensure latency constraints.
- **User acceptance testing:** Clinician review of alert quality and UX testing on dashboard.

---

11. Challenges Faced

- **Data quality:** Missing timestamps, inconsistent units, noisy sensor readings.
- **Labeling:** Ground-truth labels for deterioration are often sparse and require clinical review.
- **Class imbalance:** Critical events are rare — need sampling strategies or cost-sensitive learning.
- **Real-time constraints:** Ensuring low latency while computing rolling features.
- **Explainability:** Balancing complex models with clinicians' need for understandable reasons.

---

12. Learnings & Key Takeaways

- Preprocessing & correct temporal handling are more important than fancy models.
- Combining rule-based thresholds with model predictions reduces false alarms.
- Patient-level splitting is essential to get realistic performance estimates.
- Transparent explanations (SHAP) drastically improve clinician trust.

---

13. Future Enhancements

1. **Deploy streaming pipeline** with Kafka + Flink (or Cloud Pub/Sub) + model serving (KFServing / Seldon).
2. **Multi-modal inputs:** Add EHR data and medication info to improve predictions.
3. **Active learning:** Let clinicians label uncertain cases to improve the model.
4. **Personalized baselines:** Model per-patient baselines to capture unique physiology.
5. **Edge inference:** Lightweight models running on devices for ultra-low latency alerts.
6. **Explainability UI:** Interactive SHAP explanations for bedside use.

---

14. References

- E. R. et al., "Early warning systems for patient deterioration," *Journal of Clinical Monitoring*, 2019.
- Lundberg & Lee, "A Unified Approach to Interpreting Model Predictions," *NIPS*, 2017.
- Scikit-learn documentation — model training and evaluation.
- tsfresh documentation — time-series feature extraction.
