CSE 6220: Big Data Systems and Analytics

Project Report: Group 8

Enhancing Patient Care: A Real-Time Healthcare Monitoring, Alerting, And Predictive System

Anshit Verma, Manvi Goel, Navdha Agarwal, Padma Priya Mukkamala, Reshma Ramachandra, Sindhu Raghuram Panyam

Contents

Introduction

In the dynamic healthcare environment of hospitals and Intensive Care Units, there is a pressing need for an **advanced system capable of continuously monitoring crucial medical parameters**. This system should not only handle escalating data volumes seamlessly but also integrate harmoniously with diverse healthcare devices and existing systems. Furthermore, it should extend its capabilities to encompass at-home care and telemedicine scenarios, ensuring comprehensive patient care. One key feature of this system is the ability to **tailor alert thresholds to individual patient conditions** and preferences.

This project designs a novel and end-to-end solution for patient health monitoring. This solution includes a two-part design implementation as shown in Fig. 1:

1. **Ensemble Predictive Model:** We implement a novel solution for time-series analysis by fine-tuning popular machine learning algorithms, Arima (Gilbert, 2005), Prophet[1,] and XGBoost (Chen & Guestrin, 2016) on our custom patient health dataset. Our model **forecasts future heart rate and respiratory rate** values for individual patients, leveraging their historical data. Additionally, it identifies the time steps at which the predicted values surpass the normal human range, **facilitating pro-active timely intervention** in healthcare scenarios.

2. Real-time alert generation: We also design and develop a **real-time alert generation system** based on **Apache Flink and Kafka**. The model is designed to handle large amounts of medical data and issues alerts when the heart rate and respiratory rate values of a patient deviate from the acceptable human range, pinpointing specific time steps where such deviations occur. Since this is a real-time system, the alerts generated are accurate and are used to validate the performance of the ensemble predictive model.

While machine learning helps in predicting values of patient health metrics like heart rate and respiratory rate, there is a good chance that these models generate false positives, i.e. cases where the model predicts concerning values when the patient is actually in a stable condition. The implications of such predictions are steep in the healthcare domain. Hence, we also have a real-time monitoring system that can validate the alerts generated by the predictive model.

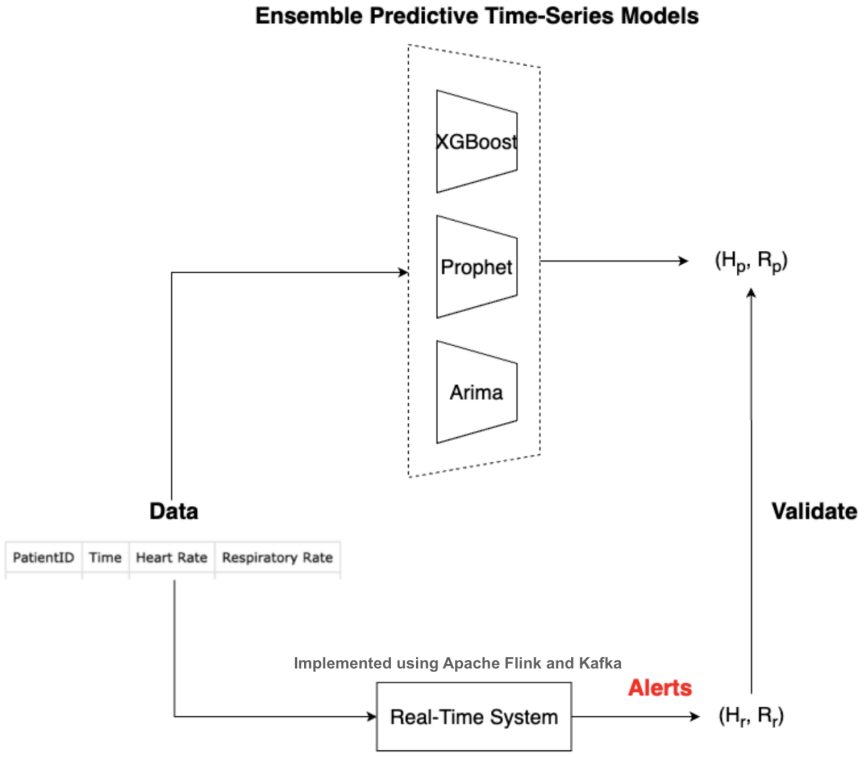


Figure 1: An overview of the proposed solution for patient health monitoring. The Ensemble model predicts heart rate and respiratory value (Hp, Rp) for a given patient at a future time step, and the real-time system validates if these values are accurate.

Existing solutions for vital sign monitoring cannot often predict critical conditions in advance. Traditional monitoring systems are reactive, generating alerts only when vital signs cross predefined thresholds. This approach may lead to delayed intervention and compromised patient outcomes. Our goal is to address this limitation by developing a system that goes beyond threshold-based alerts. We aim to leverage historical data and advanced machine learning techniques to forecast abnormal vital sign patterns and provide timely alerts, thus bridging the gap in current healthcare monitoring practices.

The current landscape of healthcare monitoring is characterized by several pressing **challenges** that necessitate innovative solutions:

1. **Data Volume and Real-time Monitoring**: With the advent of wearable devices and IoT in healthcare, there's a surge in the volume of data generated. Managing and processing this large volume of data can be a significant challenge. In critical care scenarios, real-time monitoring is vital. Delays in data transmission or processing can be detrimental to patient outcomes.

2. **Reactive Healthcare Monitoring**: Traditional healthcare monitoring systems predominantly rely on reactive approaches, triggering alerts only when vital sign values breach predefined thresholds. This reactive approach often

results in delayed interventions, escalating healthcare costs, and, in some unfortunate cases, preventable patient morbidity and mortality.

3. **Lack of personalization**: Healthcare monitoring systems typically lack the personalization required to provide alerts tailored to individual patient profiles. Existing systems employ static thresholds for generating alerts, treating all patients uniformly. This approach overlooks the inherent variations in patient vital sign baselines, medical history, and overall health conditions. Consequently, it may lead to a high rate of false alarms and alert fatigue among healthcare professionals.

4. **Resource-Intensive Emergency Interventions**: Critical incidents often necessitate resource-intensive emergency interventions, including hospital readmissions, intensive care unit admissions, and invasive medical procedures. These interventions not only strain healthcare resources but also expose patients to increased healthcare costs and potential complications.

i. Goal and Scope

Our project's overarching goal is to revolutionize the field of healthcare monitoring by introducing a holistic, predictive, and personalized system for real-time medical vitals monitoring. We aim to introduce a **novel approach** to **real-time monitoring** of medical vitals, such as blood pressure, body temperature, heart rate, oxygen level, etc., while simultaneously **incorporating predictive analytics** and creating an alerting mechanism system. While existing solutions primarily focus on static monitoring, our unique perspective is to enhance healthcare outcomes by identifying critical conditions before they escalate. This project will cater to a diverse user community, including healthcare professionals, patients, and caregivers, by providing personalized and timely alerts, ultimately improving patient care and reducing healthcare costs.

Our project aims to:

- **Create Personalized Alerts**: The idea is to tailor alerts to individual patient profiles, accounting for variations in vital sign baselines and medical history. This approach ensures that alerts are relevant and **minimizes false alarms**. By leveraging **real-time analysis of big data**, we will design a system that not only considers real-time vital sign data but also factors in the patient's historical health data. We envision that this personalized approach will greatly benefit healthcare professionals, who will receive alerts that are highly specific to each patient's unique health profile.
- **Predict Critical Events**: We will develop **predictive models** that **forecast deteriorating health conditions** based on historical data patterns. Our machine learning algorithms will analyze large datasets, identify trends, and provide **early warnings** about potential health crises. This proactive approach enables healthcare providers to intervene before conditions escalate, reducing the risk of adverse events and improving patient outcomes. Additionally, we will conduct a comprehensive comparison and analysis of the behavior predicted by the models with respect to real-time data. This analysis

will serve as a critical component of our project, ensuring the accuracy and effectiveness of the predictive models.

- **Create a cost-effective healthcare system**: Ultimately, our project aims to **reduce the cost burden** on healthcare systems by minimizing critical incidents, hospital readmissions, and the need for emergency interventions. By preventing medical emergencies through early intervention and personalized alerts, we can significantly reduce the financial strain on healthcare institutions and improve the overall quality of care.

ii. Novelty

Our novel contributions to this project include:

- Developed an Apache Flink and Kafka-based real-time alert generation system that takes the patient health data as input data streams and generates alerts for when the data points exceed the normal human range.
- Designed a Prophet-based, Arima-based, and XGBoost-based model that works for time-series analysis i.e. for our unique use-case. These models are trained on our custom data in a supervised learning manner and fine-tuned using hyperparameter tuning.
- Built an ensemble predictive model that combines the workings of the three machine learning models to ensure that the system provides accurate results for all patients and makes the entire solution more robust.

iii. Related Works

Recent advancements in big data and real-time analytics have significantly impacted healthcare monitoring systems. For example, studies have shown the effectiveness of real-time data processing in healthcare, leveraging frameworks like Apache Flink for handling streaming patient data [Rhaed Khiati; Muhammed Hanif; Choonhwa Lee et al]. The importance of timely and accurate data ingestion, often through systems like Apache Kafka, is highlighted as a critical component for ensuring the efficacy of real-time monitoring systems[Paul Le Noac'h; Alexandru Costan; Luc Bougé et al]. Moreover, the implementation of anomaly detection algorithms plays a pivotal role in identifying potential health risks, providing a foundational basis for developing advanced alerting mechanisms in healthcare settings [Rhaed Khiati; Muhammed Hanif; Choonhwa Lee et al]. These studies collectively underline the potential of big data technologies in transforming healthcare monitoring, emphasizing the need for scalable, reliable, and efficient systems to enhance patient care and outcomes.

Several Machine Learning (ML) methods have been employed to analyze survival data in medical contexts. The most commonly used algorithms include Support Vector Machines (SVM), Random Forests, etc. (Singh & Mukhopadhyay, Ishwaran et al, Chen et al). However, they often rely on predefined features, limiting their ability to capture intricate patterns in complex medical data. Additionally, these traditional ML techniques may struggle with modeling nonlinear relationships and interactions, hindering their predictive performance, particularly when complex dependencies among variables exist.

Deep Learning (DL) methods have garnered attention for survival analysis in medical data because of their capacity to automatically extract intricate features from vast and unstructured datasets. Models such as DeepSurv [Katzman et al] and DeepHit [Lee et al] leverage deep neural networks to capture nuanced relationships within patient data. However, DL models are often perceived as "black boxes," posing challenges in interpreting their predictions, which is vital for clinical decision-making. Furthermore, DL techniques typically demand substantial labeled data, which can be a limitation in medical settings where obtaining labeled survival data can be resource-intensive. Additionally, DL models may not perform optimally with small datasets or in cases of significant class imbalance, which are common challenges encountered in medical survival analysis tasks.

Time-series analysis of medical data has also been explored concerning ML and DL models. Bloch et al. present an ML approach that utilizes SVM with a radial basis function to predict the onset of sepsis in ICU patients using real-time bedside monitor data. Alghatani et al. also propose a similar approach that uses the random forest algorithm to predict the length of ICU stay and mortality based on the patient's vital signs data. However, a limitation lies in the risk of over-relying on data-driven predictions, which may not fully capture the complexity of real-world clinical scenarios and may require further validation in diverse healthcare settings.

Lehman et al. present a novel Deep Learning approach that combines switching state space models (SVAR) with convolutional neural networks (CNN) for the classification of time series data, effectively tackling the complexities inherent in the analysis of physiological data. SVAR is employed to capture the underlying temporal dynamics of the time series, while CNN is tasked with identifying higher-level transition patterns. Nonetheless, a potential drawback lies in the interpretability of SVAR-derived features, which might need domain expertise to comprehend their physiological implications.

Data

Our dataset is built on top of the vast PhysioNet dataset, which mainly looks at mortality prediction within intensive care unit (ICU) contexts. In contrast, our dataset has a different aim. Instead of guessing overall survival, our **custom dataset** focuses on **predicting single vital parameters**, one at a time. Instead of giving a big picture of whether someone might survive or not, our dataset digs deep into predicting specific factors. This helps us understand these important indicators better, giving us detailed insights into each one's ability to predict outcomes.

	Physionet Dataset	**Our custom dataset**
Focus	Mortality prediction, Disease progression modeling, Risk assessment	Predictive analysis of individual vital parameters
Models implemented	Logistic regression, Neural networks, SVM	Time-series model for predictive analysis - Prophet,

		ARIMA, XGBoost
Number of patients	4000	266
Metrics available	37 variables may be observed once, more than once, or not at all in some cases (Heart Rate, Respiratory rate, Blood pressure, Glucose etc)	Heart rate, Respiratory rate

i. Preprocessing

The available dataset contains varying measurements for patient vitals. To prepare this data for our predictive model, we undergo three important steps of processing. These steps help us sort and organize the information, creating a uniform dataset that our predictive model can easily work with. This careful preparation ensures that the data is consistent and well-suited for accurate predictions, allowing us to effectively use it for our analysis.

1. **Identification of patients:**

The preprocessing step includes identifying patients possessing both heart rate and respiratory rate data. Among the patients, 3933 have heart rate readings, while 791 have respiratory rate readings. Notably, nearly all patients with respiratory rate data also have corresponding heart rate information available.

Table: Statistics of patients with heart rate and respiratory rate readings

Number of patients with Heart Rate readings	**Number of patients with Respiratory Rate readings**	**Number of patients with both readings**
3933	1104	1100

2. **Uniform timestamps:**

To maintain consistent timestamps across all patients, we either eliminate specific timestamps or apply interpolation methods to standardize the data. Patient data is recorded at different intervals—some every 15 seconds, others at 30 seconds or 1-minute intervals. Utilizing interpolation, we ensure uniformity in this data. Following the initial processing stage, while the maximum recorded duration for a patient is 48 minutes, the majority have roughly 30 minutes of data. We select patients with 29 minutes of data or more and extract information from the initial 29 minutes. Subsequently, after this processing stage, we have a total of 266 patients in our dataset.

3. **Data interpolation:**

We use interpolation to increase the data points to 2900 per patient, ensuring a more robust dataset. These extra data points help improve statistical reliability. They're then used by time-series models to make accurate predictions.

Table: Data Statistics

Number of patients	**Total Number of Data Points for each patient**	**Number of data points for Training**	**Number of data points for Testing**
266	2900	2800	100

Real-Time Monitoring System

i. System Overview

We developed a system that utilizes the power of Apache Flink and Kafka framework to deliver a dynamic and responsive solution. By processing vital signs such as heart rate and respiratory rate in real time, the system ensures timely anomaly detection, allowing healthcare professionals to intervene promptly. The seamless integration of data ingestion, dynamic alert definition handling, and advanced analytics positions this system as a robust and adaptive healthcare monitoring solution.

The figures below represent the complete workflow highlighting the system architecture:

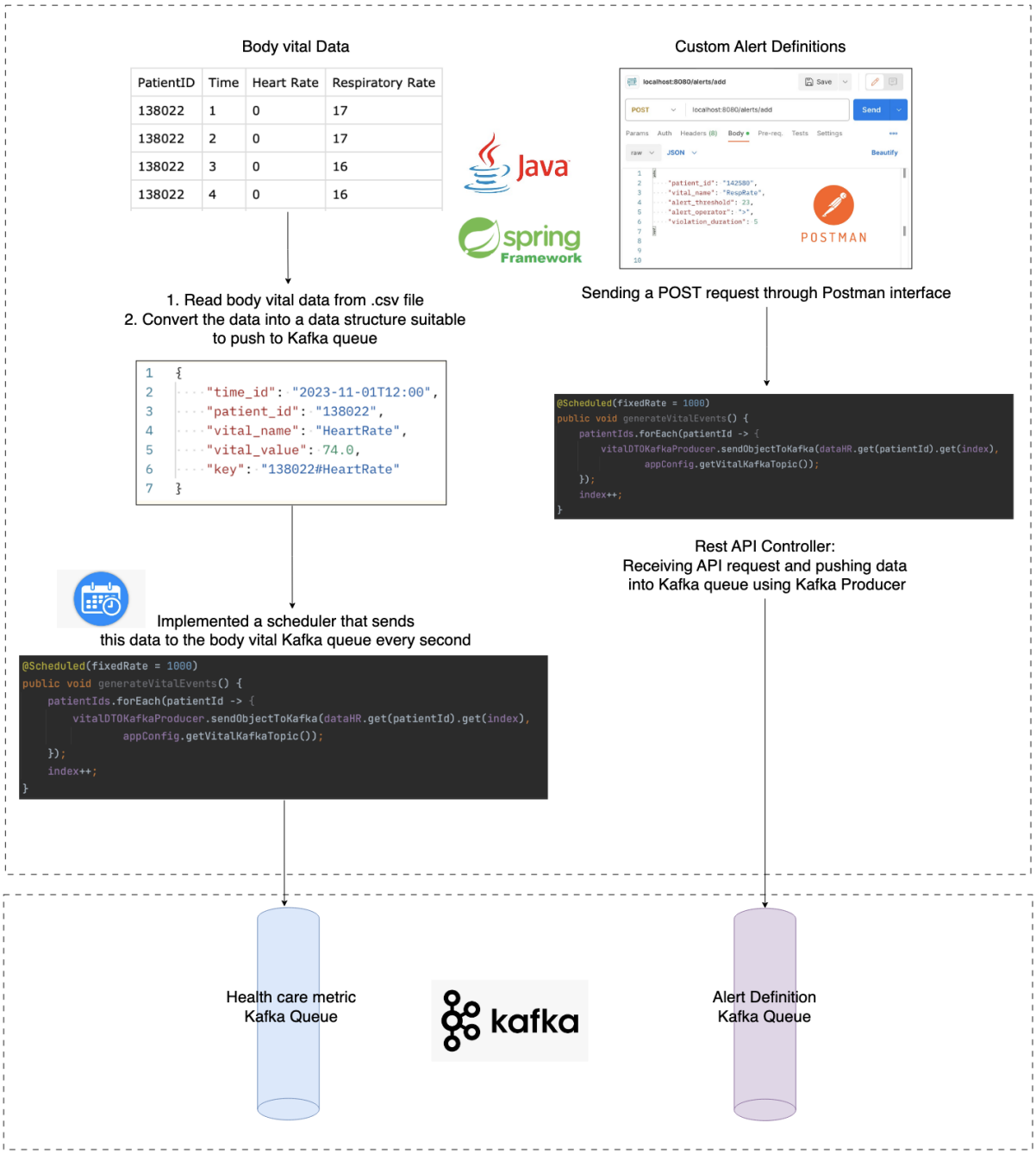


Figure 2: Reading Data and pushing to the Kafka queue

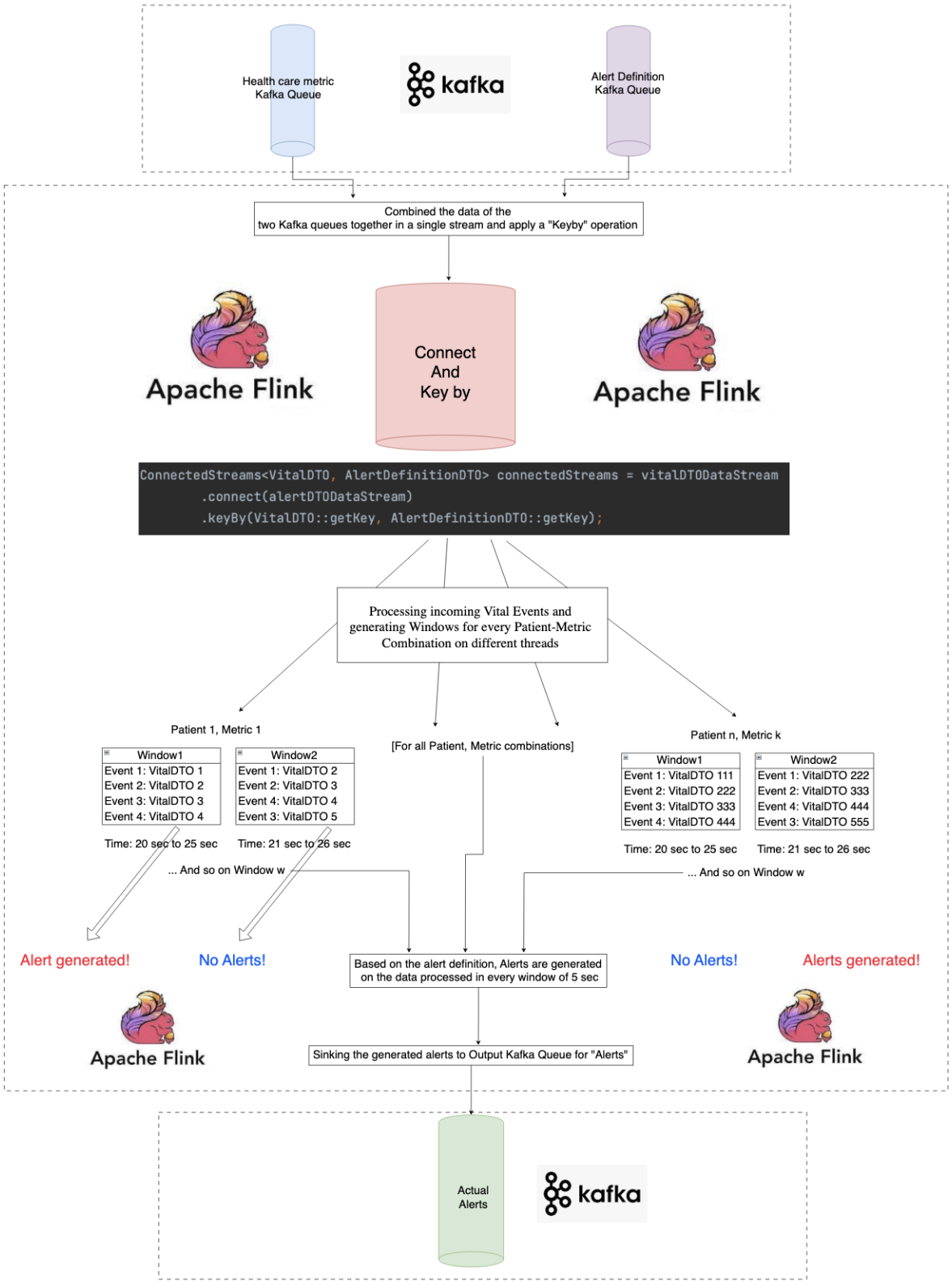


Figure 3: Processing incoming Events, triggering Windows, and generating final Alerts in Apache Flink Application

ii. Motivation and Novelty

Why Apache Kafka and Flink?

1. **Scalable and Fault-Proof Stream Processing**

   Kafka serves as the backbone for its ability to handle large-scale, real-time data ingestion. Its distributed and fault-tolerant architecture ensures that vital signs, such as heart rate and respiratory rate metrics, are reliably transmitted to downstream processing components.

   Apache Flink complements Kafka by providing a powerful stream processing framework. Its ability to process data in real-time, handle complex event processing logic, and scale horizontally makes it an ideal choice for continuously analyzing and monitoring streaming healthcare data.

2. **Seamless Integration**

   The seamless integration of Kafka and Flink streamlines the entire data processing pipeline. Kafka acts as a durable and efficient buffer, decoupling the data producers (monitoring devices) from the consumers (Flink applications). This architecture allows for flexible and independent scaling of data ingestion and processing components.

Compatibility with the ML system

1. **Advanced Analytics with Machine Learning Models:** Incorporating ARIMA, Prophet, and XGBoost time series models into the system enhances its predictive capabilities. The compatibility between Flink and machine learning systems allows for the seamless integration of these models into the real-time data processing pipeline.

2. **Leveraging Predictive Analytics for Proactive Alerts:** The predictive models generate future data points for heart rate and respiratory rates, enabling proactive monitoring. By applying these predicted values to the existing threshold definitions, the system can generate alerts before actual anomalies occur, facilitating early intervention and improving patient outcomes.

Novelties introduced

1. Traditionally, healthcare monitoring systems often rely on periodic data sampling and batch processing. The real-time nature of the developed system enables continuous monitoring of vital signs, allowing for immediate anomaly detection. This novelty ensures that healthcare professionals receive alerts in real time, enabling prompt intervention and potentially preventing adverse health events.

2. One of the key innovations is the integration of predictive analytics using time series models (ARIMA, Prophet, XGBoost). This addition allows the system to generate predictions for heart rate and respiratory rates, enabling proactive alerting. By comparing predicted values with predefined thresholds, the system can anticipate potential anomalies before they occur. This shift from reactive to proactive monitoring represents a significant advancement in patient care.

3. The system introduces a dynamic approach to alert definitions using a Postman service. Healthcare professionals can dynamically define and update alert criteria in real time through a user-friendly interface. This customization capability allows for tailored monitoring, accommodating the unique health conditions and requirements of individual patients. The ability to adapt alert definitions on-the-fly is a novel feature that our system introduces.

4. Unlike traditional systems with static alerting rules, the real-time monitoring system embraces continuous learning. The integration of machine learning models allows the system to adapt and refine predictions as it receives new data. This continuous learning aspect ensures that the monitoring system stays current and aligns with the evolving health conditions of patients. It represents a departure from static, one-size-fits-all monitoring strategies.

iii. Implementation

Data and Control Flow of Real-Time Monitoring System

The following describes the complete architecture and functional components of the application we developed using Apache Flink and Kafka frameworks.

1. **Data Ingestion from Monitoring Devices:**
    - The system begins with the ingestion of vital data from healthcare monitoring devices. Since this is a simulated environment, the vital data (e.g., heart rate, blood pressure) is assumed to be stored in CSV files.

- We implemented a script (or service) that reads this data from the CSV files and publishes it to the first Kafka queue, designated as the **Vital Data Queue**. This simulates the real-time data feed from monitoring devices in a healthcare setting.

2. **Ingestion of Alert Definitions:**
   - Concurrently, the system handles the ingestion of alert definitions through a RESTful service that we built using the Spring framework.
   - The motivation behind this is that healthcare professionals can define or update alert criteria in real time via this service, which then pushes these definitions into the second Kafka queue, known as the **Alert Definition Queue**. This system is implemented using the PostMan Interface.

3. **Data Processing in Apache Flink:**
   - Both data streams from the Kafka queues are consumed by the Spring-based **Apache Flink application**. The Vital Data Queue provides a continuous stream of patient vitals, while the Alert Definition Queue delivers the latest alert criteria.
   - The first operation that our Flink application performs is to connect these two streams. This allows the system to process vital data in the context of the most current alert definitions.

4. **Key-Based Stream Partitioning:**
   - After connecting the streams, the application applies a **keyBy()** operation. This function partitions the data based on Patient ID and vital signs (e.g., heart rate, respiratory rate), ensuring that all data related to a specific patient-vital combination is handled by the same thread. This step is crucial for maintaining data consistency and enables efficient processing.

5. **Sliding Window Analysis:**
   - The application then uses a sliding window mechanism, with each window spanning 5 seconds. Within each window, it evaluates all the vital sign events against the current alert criteria.
   - This sliding window approach allows the system to continuously analyze large amounts of data in real-time, providing an effective method for timely anomaly detection.

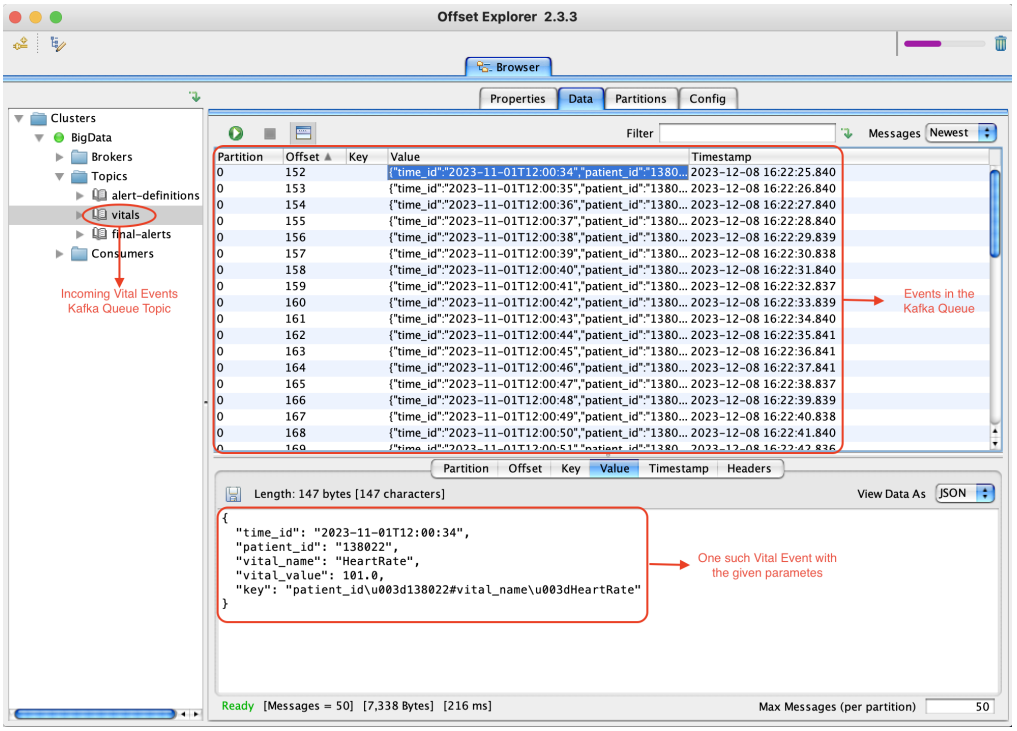
6. **Alert Generation and Notification:**
   - If the data within a window meets the criteria set in an alert definition (e.g., a heart rate falling outside a specified range), the system generates an alert.

- This alert is then published to a Sink Kafka Queue. From here, an Alert Notification System can consume these alerts and promptly notify the respective healthcare professionals with detailed information about the detected anomaly.

This data flow ensures a seamless and efficient process for real-time monitoring of patient vitals, enabling timely detection and notification of potential health risks. The system's architecture leverages modern data processing frameworks and methods, making it robust, scalable, and suitable for the dynamic needs of healthcare monitoring.

iv. Results and Screenshots of the Real-Time System

Step 1: Data Ingestion from CSV into Kafka Queue



Offset Explorer can be used to visually see the elements in the Kafka Queues.

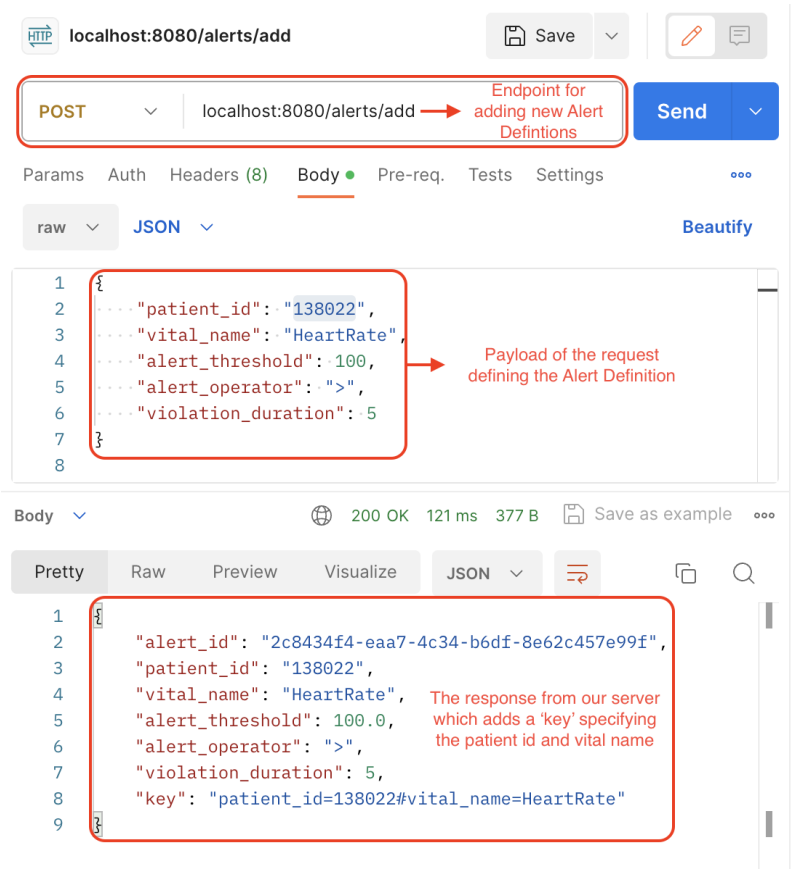
Step 2: Starting our Real-Time System, and ingestion of the above events from Kafka queue into the system by Apache Flink

```
2023-12-08 16:22:13,876 INFO : Vital:: (time='2023-11-01T12:00:22', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:14,914 INFO : Vital:: (time='2023-11-01T12:00:23', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:15,948 INFO : Vital:: (time='2023-11-01T12:00:24', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:16,873 INFO : Vital:: (time='2023-11-01T12:00:25', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:17,900 INFO : Vital:: (time='2023-11-01T12:00:26', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:18,923 INFO : Vital:: (time='2023-11-01T12:00:27', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:19,853 INFO : Vital:: (time='2023-11-01T12:00:28', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:20,881 INFO : Vital:: (time='2023-11-01T12:00:29', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:21,910 INFO : Vital:: (time='2023-11-01T12:00:30', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:22,939 INFO : Vital:: (time='2023-11-01T12:00:31', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:23,869 INFO : Vital:: (time='2023-11-01T12:00:32', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:24,905 INFO : Vital:: (time='2023-11-01T12:00:33', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:25,939 INFO : Vital:: (time='2023-11-01T12:00:34', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:26,869 INFO : Vital:: (time='2023-11-01T12:00:35', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:27,907 INFO : Vital:: (time='2023-11-01T12:00:36', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:28,941 INFO : Vital:: (time='2023-11-01T12:00:37', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:29,873 INFO : Vital:: (time='2023-11-01T12:00:38', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
```

Above is the screenshot of the **vital events** being read by our Flink Application from the "vital" Kafka Queue. The screenshot shows:

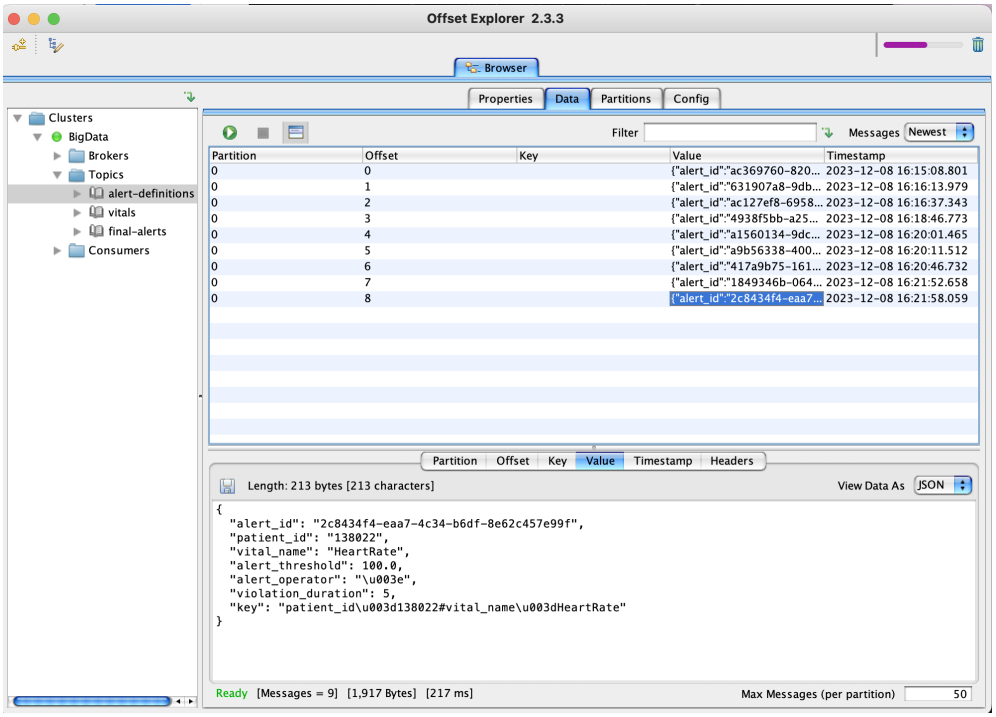
- the time the event was read in our application in green
- the time the event was generated
- the patient id
- the vital corresponding the event, here Heartrate
- the value of the vital.

Step 3: Sending Alert Definition to the system in the form of a REST API.



In the above screenshot, we are sending the alert definition which states that "An alert should be raised for **Patient with ID 138022**, when his/her **Heart Rate is >=100** for consecutive **duration of 5 sec**." Note the parameters - **patient id, vital name, alert threshold, alert operator and violation duration.**

Step 4: Pushing the Alert Definition in Kafka Queue.



After our system receives the alert definition, we push it to another kafka topic “alert-definition”.

Step 5: Processing the Alerts

Our Flink application then processes the alerts by creating a 5 sec window using windowing strategy of Flink, and keeps processing the above sent alert definitions on the windows. If all the events in the windows satisfy the definition then an alert is triggered.

Let’s understand this with examples. We have the incoming metric values from 12:00:34 - 12:00:44 (11 events). Below is the screenshot of all these events.

```
2023-12-08 16:22:25,939 INFO : Vital:: (time='2023-11-01T12:00:34', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:26,869 INFO : Vital:: (time='2023-11-01T12:00:35', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:27,907 INFO : Vital:: (time='2023-11-01T12:00:36', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:28,941 INFO : Vital:: (time='2023-11-01T12:00:37', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:29,873 INFO : Vital:: (time='2023-11-01T12:00:38', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:30,906 INFO : Vital:: (time='2023-11-01T12:00:39', patientId='138022', vitalName='HeartRate', vitalValue=99.0)
2023-12-08 16:22:31,946 INFO : Vital:: (time='2023-11-01T12:00:40', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
2023-12-08 16:22:32,881 INFO : Vital:: (time='2023-11-01T12:00:41', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
2023-12-08 16:22:33,918 INFO : Vital:: (time='2023-11-01T12:00:42', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
2023-12-08 16:22:34,951 INFO : Vital:: (time='2023-11-01T12:00:43', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
2023-12-08 16:22:35,880 INFO : Vital:: (time='2023-11-01T12:00:44', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
```

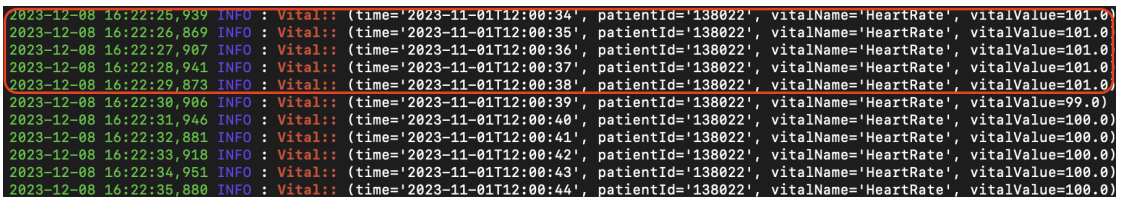
These are the alerts generated for the alert definition

```
2023-12-08 16:22:31,637 INFO : ALERT for window 2023-11-01T12:00:34 - 2023-11-01T12:00:38.999
2023-12-08 16:22:36,706 INFO : ALERT for window 2023-11-01T12:00:39 - 2023-11-01T12:00:43.999
2023-12-08 16:22:37,696 INFO : ALERT for window 2023-11-01T12:00:40 - 2023-11-01T12:00:44.999
```

Now for the above 11-sec time frame, the possible 5 sec windows are:

1. 12:00:34 - 12:00:39: We see that all the events in this window are above the threshold value of 100. And therefore we get the first alert.

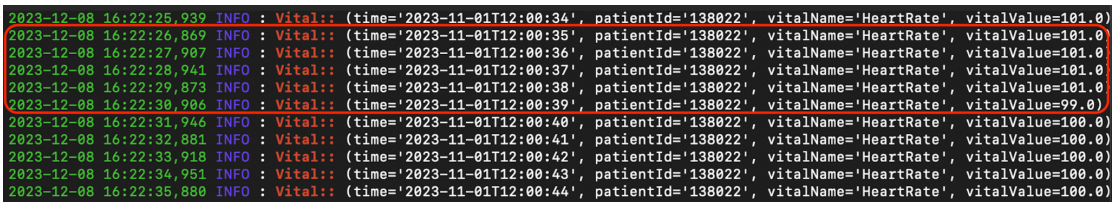
   Window:



   Alert:

2. 12:00:35 - 12:00:40: For this window, there is one event (the last one) which has the value less than a threshold.

   Window:



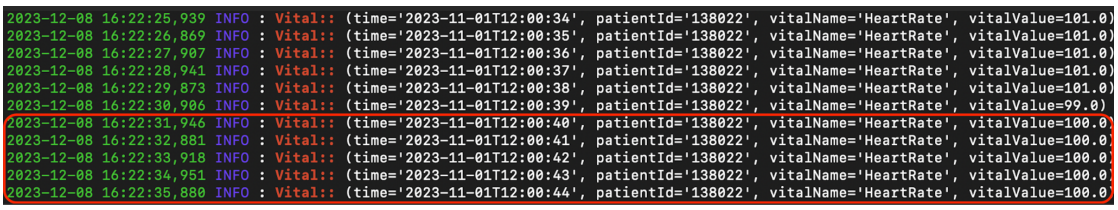
   Alert: None

3. 12:00:36 - 12:00:41: Same happens with this window. No alert is generated since event with value 99 lies in it.

4. 12:00:37 - 12:00:42: Same - No Alert
5. 12:00:38 - 12:00:43: Same - No Alert

6. 12:00:39 - 12:00:44: We get an alert here because our window starts just after the event with value 99.

   Window:



   Alert:

7. 12:00:40 - 12:00:45: We see that all the events in this window are above or equal the threshold value of 100. And therefore we get another alert.

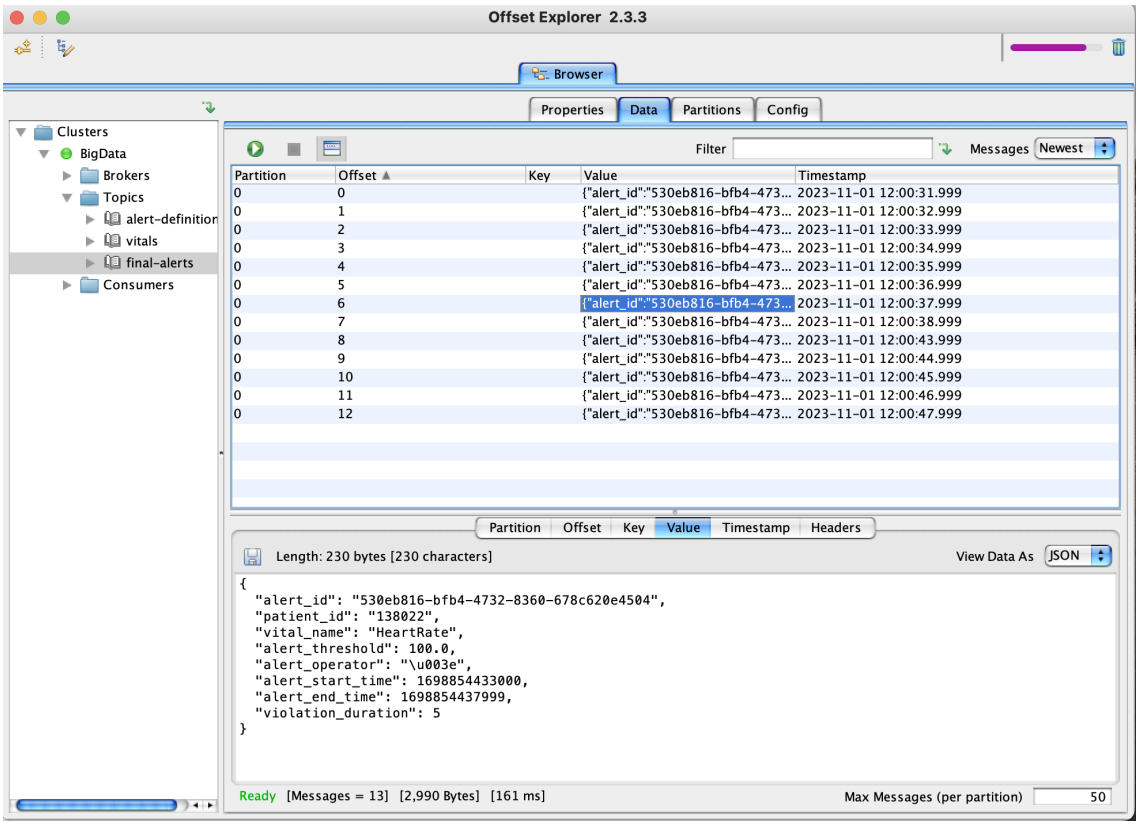
Window:

```
2023-12-08 16:22:25,939 INFO : Vital:: (time='2023-11-01T12:00:34', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:26,869 INFO : Vital:: (time='2023-11-01T12:00:35', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:27,907 INFO : Vital:: (time='2023-11-01T12:00:36', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:28,941 INFO : Vital:: (time='2023-11-01T12:00:37', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:29,873 INFO : Vital:: (time='2023-11-01T12:00:38', patientId='138022', vitalName='HeartRate', vitalValue=101.0)
2023-12-08 16:22:30,906 INFO : Vital:: (time='2023-11-01T12:00:39', patientId='138022', vitalName='HeartRate', vitalValue=99.0)
2023-12-08 16:22:31,946 INFO : Vital:: (time='2023-11-01T12:00:40', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
2023-12-08 16:22:32,881 INFO : Vital:: (time='2023-11-01T12:00:41', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
2023-12-08 16:22:33,918 INFO : Vital:: (time='2023-11-01T12:00:42', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
2023-12-08 16:22:34,951 INFO : Vital:: (time='2023-11-01T12:00:43', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
2023-12-08 16:22:35,880 INFO : Vital:: (time='2023-11-01T12:00:44', patientId='138022', vitalName='HeartRate', vitalValue=100.0)
```

Alert:

```
2023-12-08 16:22:31,637 INFO : ALERT for window 2023-11-01T12:00:34 - 2023-11-01T12:00:38.999
2023-12-08 16:22:36,706 INFO : ALERT for window 2023-11-01T12:00:39 - 2023-11-01T12:00:43.999
2023-12-08 16:22:37,696 INFO : ALERT for window 2023-11-01T12:00:40 - 2023-11-01T12:00:44.999
```

Step 6: Pushing the Generated Alerts to Kakfa - Final Step



Once the alerts are triggered, they are then pushed to the sink kafka queue “final-alerts” explaining all the alert parameters like when was the window triggered, and explaining the nature of the alert-definition that triggered it.

Ensemble Predictive Model

This section aims to delve deeper into the historical data of patients to predict future risks. In the realm of healthcare, the surge in data volume, variety, and velocity has necessitated an advanced approach towards monitoring and predicting patient health outcomes. This component of our project plays a pivotal role in this,

leveraging vast datasets to not only understand the current health status of patients but also to predict future risks. This is **starkly different from the conventional use of pre-trained deep learning models**, as it involves the creation of custom models tailored specifically to the unique dynamics of healthcare data. Specifically, it seeks to **determine the likelihood of a patient's health metrics falling outside of the healthy range in the future**.

Features:

Historical Data Analysis: Utilizes the vast dataset of patient metrics to understand patterns, trends, and correlations.

Survival Analysis: Employs survival analysis techniques to estimate the time until a significant event, such as a metric falling outside its healthy range. This provides a probabilistic understanding of future risks.

Predictive Modeling: Builds on the insights from survival analysis to develop predictive models. These models can forecast potential future deviations from healthy metric ranges based on a patient's current and past data.

Risk Stratification: Patients can be categorized based on their risk levels, allowing healthcare providers to tailor interventions and monitoring strategies accordingly.

Potential Impact: By predicting future risks, healthcare providers can proactively manage and monitor high-risk patients, potentially preventing adverse events. This analytical approach complements the real-time monitoring system, ensuring both immediate and long-term patient safety and well-being.

i. Engineering Choices

For our machine learning tasks, we are using multiple models to consume the data and use it in a variety of settings to give comprehensive results. Since for the first stage we are using predictive modeling, our models of choice would be Time-series-based models such as ARIMA, Prophet, and XGBoost. For the second stage, to determine the best outcome, we are using an ensemble of all three models to give the best prediction.

Pre-existing Models in Healthcare Monitoring

Traditional healthcare monitoring systems primarily relied on threshold-based alerts and simple statistical models. Here are some of the commonly used methods before the introduction of advanced analytics:

Simple Regression Models: Linear regression and its variants were used for forecasting, based on the assumption that relationships between variables are linear and constant over time.

Support Vector Machines (SVM): SVM was employed for classification and regression problems, focusing on finding the best margin that separates the classes.

Random Forests: An ensemble learning method used for classification and regression that operates by constructing multiple decision trees at training time.

Shortcomings of Previous Approaches

Reactive Nature: Traditional systems were reactive, alerting only after an adverse event was imminent or already occurring. Adding an element of predictive analytics would mitigate the risks associated with this concern.

Limited Complexity Handling: Simple statistical models often failed to capture complex, non-linear relationships in physiological data.

Static Predictive Capacity: Prior models did not adapt well over time to new data, which is essential in the dynamic environment of healthcare where patient conditions can change rapidly.

Data Utilization: Many models did not make full use of the time-series nature of healthcare data, losing valuable information embedded in the sequence of observations.

Time Series Model:

Time Series Analysis is a way of studying the characteristics of the response variable concerning time as the independent variable. To estimate the target variable in predicting or forecasting, use the time variable as the reference point. It is an observation from the sequence of discrete time of successive intervals. Since TSA involves producing the set of information in a particular sequence, this makes it distinct from spatial and other analyses.

Contrary to popular belief, time-series models in healthcare are not "plug-and-play" solutions like some pre-trained language models (e.g., GPT). Here's why:

1. **Customized Model Training***:* Unlike pre-trained models that come with a pre-existing understanding of language, time-series models for healthcare data require bespoke training. They must learn the intricacies and nuances of medical data, which are highly specific and varied across different patient demographics and conditions.
2. **Complex Data Structures***:* Healthcare data involves complex time-series information with varying intervals, unlike the more uniform structure of text data in language models.

3. **High Stakes for Accuracy***:* The margin for error is minimal, given that incorrect predictions can have serious consequences in a medical context.

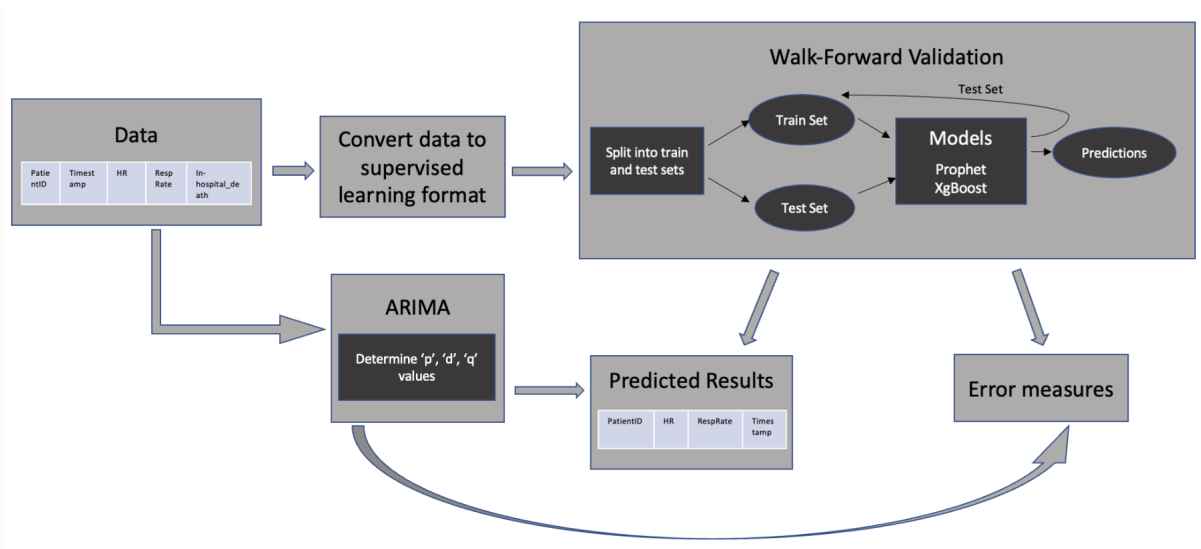
Currently, there exist significant amounts of digital healthcare data that can provide helpful insights into healthcare expenditures, and these data could bring about positive changes in healthcare policymaking (Farley et al., 2006). Although prior

research has performed time-series forecasting in healthcare data, a major challenge is selection of the appropriate predictive model to use for performing analyses (there are very few suggestive forecasting algorithms for healthcare data due to the newness of this domain and its datasets). A way of overcoming the model selection challenge is by evaluating the predictions from existing forecasting methods with other recent methods in literature (Makridakis et al., 2018).

For our use case, we are going to use the real-time data streams to train the model and then the time series model will be used in conjunction with the real-time alerting system. This provides us with a predicted future value of the metric we are observing and helps alert the medical professionals before the metric goes over threshold, for them to take preventive measures. Time series analysis will also help us tackle seasonality and cyclic variations that might occur for a patient, which can help identify periods where the patient might observe values which are not usually normal. Trend identification is also one of the major goals of our predictive analysis with time series modeling.

ii. Architectural Overview

We are mainly looking at 3 models - **ARIMA, XGBoost and Prophet**. These are renowned models with a well-established track record of effectively handling time-series data across various domains. Their prowess in forecasting and data analysis is not derived from pre-trained knowledge but rather from their underlying mathematical equations and algorithms, which have been rigorously vetted through extensive academic research and validated in diverse real-world applications. These models bring distinct advantages to time-series analysis, each addressing specific complexities inherent in the data to extract meaningful insights. It's crucial to understand that while libraries for these models provide the algorithmic frameworks, they do not include any "trained information." When we employ these models, we are utilizing their untrained, foundational structures. The actual predictive power will be developed as we train them on our specific healthcare data, ensuring that they are finely tuned to the nuances and patterns within our unique dataset.



The provided diagram encapsulates the process flow of predictive analytics in healthcare data monitoring.

1. **Data Source and Preparation:** The starting point is raw patient data, which includes identifiers such as timestamps and PatientIDs, and outcome variables such as heart rate (HR), and respiration rate (RespRate). This data is representative of continuous monitoring, akin to what would be obtained from real-world healthcare settings.

2. **Supervised Learning Conversion:** The data is then converted into a format suitable for supervised learning, implying the preparation of the dataset where the input variables (PatientID, time, etc.) are used to predict the target variable (HR or RespRate).

3. **Time-Series Specific Modeling (ARIMA):** The AutoRegressive Integrated Moving Average (ARIMA) model, a staple in time-series analysis, is utilized to assess and forecast based on past values of the vitals. It requires the determination of parameters 'p' (past value influence), 'd' (degree of differencing), and 'q' (size of moving average window).

4. **Data Splitting for Walk-Forward Validation:** The dataset is split into training and testing sets to evaluate the model's predictive performance. This step ensures that the model is not tested on the data it has been trained on, thereby providing a fair assessment of its predictive capabilities.

5. **Modeling with Advanced Algorithms:** Advanced algorithms like Prophet and XGBoost are applied. Prophet is tailored for time-series data with trends and seasonal effects, while XGBoost is a powerful gradient boosting machine learning algorithm that can handle non-linear relationships in data.

6. **Predictions:** These models generate predictions of future vital sign values, which are critical in preempting potential health issues.

7. **Error Measurement:** The final step involves calculating error measures to evaluate the accuracy of the models. This feedback loop is essential for refining the models to improve predictive performance.

The process illustrated above starkly contrasts with the application of pre-trained models for several reasons:

1. **Customization:** Each step in our workflow is tailored to the specifics of healthcare data, which varies significantly from patient to patient and is not as uniform as the data pre-trained models are typically trained on.

2. **Domain-Specific Tuning:** The ARIMA model, for example, requires manual tuning of its parameters, which is a process guided by domain expertise in both statistics and the specific healthcare context.

3. **Dynamic Validation:** Walk-forward validation is an iterative process that mimics the real-time prediction scenario and is not typically part of the pre-trained model's workflow.

4. **Ground-Up Training:** We are training models from the ground up using the provided healthcare data, which ensures that they learn the nuances and specifics of the dataset at hand, unlike pre-trained models that come with knowledge transferred from other domains.

iii. Model Training and Utilization:

Training Process

- **Selection of Algorithms:** Algorithms are chosen based on their suitability to handle the characteristics of medical time-series data.

- **Parameter Optimization:** For ARIMA, parameters are selected through a process of model fitting and testing on historical data. Prophet and XGBoost are similarly tuned to capture the patterns within the dataset.

- **Validation:** Models undergo a rigorous validation process using walk-forward validation to simulate real-world application and refine the models based on their performance.

Novelty: Predictive Modeling

Learning from Data: The models learn from historical data, identifying patterns, trends, and anomalies that can signify impending health risks.

Risk Prediction: Using the learned patterns, the models can forecast the likelihood of future vital sign anomalies, which can be indicative of a health risk.

Continuous Improvement: Error measures guide the continuous refinement of models, enhancing their predictive accuracy over time

The three models implemented are:

1. ARIMA

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a classic statistical approach for modeling and forecasting time series data. It is especially suited for series that show evidence of non-stationarity, where the mean and variance change over time.

Functionality

This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

***AR*:** Autoregression. This part of the model captures the relationship between an observation and a certain number of lagged observations.

The AR part is defined by the parameter *p*, which represents the number of lagged terms. Mathematically, it's written as:

$$AR(p) : Y_t = \alpha_1 Y_{t-1} + \alpha_2 Y_{t-2} + \ldots + \alpha_p Y_{t-p} + \epsilon_t$$

where Y_t is the value at time *t*, α_i are the coefficients of the lags, and ϵ_t is white noise.

***I*:** Integrated. This component is about differencing the time series to make it stationary, which means the series will have constant mean and variance over time. The *I* part is defined by the parameter *d*, indicating the number of times the data need to be differenced to become stationary.

$$I(d) : Y_t' = (1 - B)^d Y_t$$

where *B* is the backshift operator, *d* is the order of differencing, and Y_t' is the differenced series.

***MA*:** Moving Average. This part models the relationship between an observation and a residual error from a moving average model applied to lagged observations. The MA part is defined by the parameter *q*, which represents the size of the moving average window.

$$MA(q) : Y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q}$$

where Θ_i are the coefficients of the lagged forecast errors, and ϵ_t is white noise.

Combining these three components gives us the ARIMA model, which can be represented as:

$$ARIMA(p, d, q) : (1 - \sum_{i=1}^{p} \alpha_i B^i)(1 - B)^d Y_t = (1 + \sum_{i=1}^{q} \theta_i B^i)\epsilon_t$$

To forecast with an ARIMA model, you:

- Identify *p*, *d*, and *q* by looking at autocorrelation plots, partial autocorrelation plots, and performing unit root tests to assess stationarity.
- Estimate the coefficients of the AR and MA parts through methods like Maximum Likelihood Estimation.
- Diagnose the model by checking the residuals to ensure they resemble white noise.

This process, while it may sound complex, essentially involves using the past data to predict future values while adjusting for trends and patterns recognized over time.

Implementation

In our project, ARIMA is not just a forecasting tool; it is part of a larger predictive analytics pipeline designed to anticipate future health events and outcomes. Here's how we incorporate ARIMA into our project:

1. **Data Preprocessing:** Before feeding data into the ARIMA model, we process the raw patient vital signs data to ensure it's in a suitable format. This includes handling missing values, normalizing the data, and transforming it to make the time series stationary if needed.

2. **Parameter Selection:** The ARIMA model requires the selection of three parameters: p (autoregressive terms), d (order of differencing), and q (moving average terms). We determine these parameters through a combination of grid search and domain knowledge. The 'd' parameter is particularly important as it pertains to the number of differencing required to make the series stationary, a prerequisite for ARIMA modeling.

3. **Model Training:** We train the ARIMA model on historical data, which entails the model learning the values of its parameters that best fit the past data. This historical data comprises time-stamped records of patient vitals, which the ARIMA model uses to understand how past values of the series are related to future values.

4. **Diagnostics and Validation:** After the initial training, diagnostic checks are performed to ensure that the model's residuals are white noise—meaning they do not exhibit patterns that could be modeled further. This step is crucial as the presence of patterns in residuals can indicate model inadequacy.

5. **Walk-Forward Validation:** As shown in the provided architecture diagram, we implement walk-forward validation to assess the model's predictive power. This method involves using the model to predict the next time step, then updating the model with the actual observed value, and repeating this process moving forward one step at a time through the test set. This mimics real-world application and provides a robust measure of the model's performance.

6. **Forecasting:** With a validated model, we use ARIMA to forecast future patient vitals. This could mean predicting the next day's average heart rate or the trend in respiration rate over the next week.

Training ARIMA on Healthcare Data

Selection of Training Set: The dataset is divided into a training set and a testing set. The training set includes a sequence of vital sign measurements recorded over time from various patients.

Model Fitting: Using the ARIMA model, we fit the historical data from the training set. The process involves configuring the model's parameters to best capture the underlying patterns in the data:

- *Non-Seasonal Component:* The parameters 'p', 'd', and 'q' are identified, which represent the model's autoregressive terms, degree of differencing, and moving average window, respectively.
- *Seasonal Component:* Additionally, the parameters 'P', 'D', 'Q', and 's' are set to capture seasonality, where 'P', 'D', and 'Q' represent the seasonal equivalents of 'p', 'd', and 'q', and 's' denotes the periodicity of the seasonal effects.
- *Hyperparameter Tuning:* The 'p', 'd', and 'q' values are tuned based on the model's performance on the training data, using statistical criteria such as the Akaike Information Criterion (AIC) to guide the selection process.

The code applies a grid search over a range of parameter combinations, creating a comprehensive list of potential configurations. Each ARIMA model variation is then assessed using the Akaike Information Criterion (AIC), a statistical measure that rewards model accuracy while penalizing excessive complexity. The goal is to select the parameter set that offers the best trade-off between fitting the historical data well and maintaining a parsimonious model.

Forecasting and Performance Evaluation

Once the optimal parameters are identified, the model is used to forecast future values for the next 100 time steps. The forecasting ability of the trained ARIMA model is not just based on past values but also adjusts for seasonality, leveraging the patterns identified in the historical data.

The model's predictions are then concatenated with the original dataset to form a comprehensive view of observed and predicted values over time. The Mean Absolute Error (MAE) is calculated to quantify the forecast accuracy, comparing the model's predicted values against the actual measurements in the testing set.

By tailoring the ARIMA model with meticulously chosen hyperparameters and evaluating its forecasting power, we can produce a robust model capable of predicting future patient vitals, which is an invaluable asset in proactive healthcare monitoring.

Learning and Predictions

1. **Learning Phase:** During training, ARIMA learns the patterns within the historical data—how the vitals tend to change over time and how past values and errors can predict future values.

2. **Expectations from Training:** We expect the ARIMA model to capture the temporal dynamics of the vital signs data, allowing it to make informed predictions about future values.

3. **Prediction Phase:** Once trained, the ARIMA model uses the learned patterns to forecast future values of the patient's vitals. The prediction considers both the trends and the noise in the data, providing a holistic view of the expected future measurements.

4. **Error Analysis:** Predictions are compared against actual outcomes to compute error measures such as Mean Absolute Error (MAE). These metrics inform us about the model's accuracy and guide further refinements.

Unlike static monitoring systems, ARIMA's dynamic predictions enable proactive healthcare interventions. It is a customized and iterative process, with continuous refinement based on real-world data, ensuring that the model remains sensitive and specific to the unique patterns present in the vitals of patients. Through meticulous training and validation, ARIMA becomes a powerful tool in the predictive analytics arsenal, aimed at advancing patient care and improving healthcare outcomes.

2. PROPHET

Prophet, or “Facebook Prophet,” is an open-source library for univariate (one variable) time series forecasting developed by Facebook. It is specifically engineered to address the challenges presented by time-series data that is typical in healthcare settings. With the project at hand, the application of Prophet is geared towards predicting future patient vitals, a task that is both critical and complex due to the dynamic nature of human health indicators.

Prophet implements what they refer to as an additive time series forecasting model, and the implementation supports trends, seasonality, and holidays. It works best with time series that have strong seasonality and several seasons of historical data.

Why Prophet Is Ideal for This Task

Prophet is chosen for several compelling reasons:

1. Its robust handling of various intrinsic time-series patterns and outliers is ideal for the unpredictable nature of healthcare data.

2. The flexibility in incorporating irregular events into the model makes it suitable for patient data, which may be influenced by sporadic treatments or interventions.

3. Prophet's interpretability is a significant advantage, allowing healthcare professionals to understand and trust the model's predictions.

4. Prophet automatically detects and accounts for trends and seasonal patterns in the data, which can be highly beneficial for vitals that exhibit such behaviors.

Functionality

The math behind Prophet is based on decomposable time-series models. It breaks down the data into three main effects:

Growth or Trend: This is modeled as a piecewise linear or logistic growth curve. That means it can bend at certain points, allowing the trend to naturally change over time, which is much like a flexible ruler that can bend to follow a road’s curve.

Seasonality: Prophet uses Fourier series to provide a smooth representation of daily, weekly, and yearly seasonality. Imagine drawing a smooth curve through points that represent the same time each day or each year to see the regular pattern.

Holidays and Events: These are modeled as individual impacts that occur on irregular schedules. Think of adding extra blocks to a tower on specific days to represent their effect.

The Prophet model is designed to be intuitive and yet robust, capturing various components that affect time series data. It models a time series y(t) with three main components: trend, seasonality, and holidays. Mathematically, it represents the time series as:

$$y_t = g(t) + s(t) + h(t) + \varepsilon_t,$$

where

Trend: This function models non-periodic changes. It can be either a linear function or a logistic growth curve to capture saturating growth. For a linear trend,

$$g(t) = k \cdot t + m$$

where *k* is the growth rate and *m* is an offset parameter. For a logistic growth trend,

$$g(t) = \frac{C}{1+e^{-k(t-m)}}$$

where *C* is the carrying capacity (the maximum achievable point), *k* is the growth rate, and *m* is an offset parameter.

Seasonality: This function represents periodic changes, such as weekly, monthly, or yearly seasonality. It is modeled using Fourier series, which can capture cyclical behavior:

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right)\right)$$

where a_n and b_n are the Fourier series coefficients, *n* is the number of terms in the series, and *P* is the period.

Holidays and Events: This component models irregular impacts from holidays and events. It's a series of indicator functions that equal 1 if the time *t* is on a holiday or an event, and 0 otherwise:

$$h(t) = \sum_{i=1}^{L} I(t \in D_i) \cdot \kappa_i$$

where D_i are the dates of holidays or events, *I* is the indicator function, and κ_i is the effect of the holiday or event on the time series.

Error Term: This captures any idiosyncratic changes which are not accommodated by the model. It's assumed to be normally distributed:

$$\epsilon_t \sim N(0, \sigma^2)$$

Prophet fits these components to historical data and combines them to make forecasts. It uses a decomposable model approach, which allows for flexibility and interpretable components that can adapt to the structure of the time series data. The fitting process involves using historical data to estimate the parameters of these equations, usually through a maximum likelihood or Bayesian sampling approach.

Implementation

Preparing Data for Prophet

The preparation of data for Prophet begins with transforming the dataset into a supervised learning format. This involves structuring the data with **'ds'** for the timestamp and **'y'** for the target variable, which, in this case, is the heart rate (HR) of patients. The series_to_supervised function provided in the code filters and prepares the data for a specific patient, ensuring that the model's inputs are well-structured and tailored for individualized predictions.

Training the Prophet Model

The **prophet_forecast** function encapsulates the training process. A new instance of the Prophet model is instantiated and fitted with the historical data (train) containing the 'ds' and 'y' columns. Prophet leverages this data to discern underlying patterns, such as daily or weekly cycles in heart rate, seasonal effects, and other trends that might be influenced by the patient's routine or treatment schedule.

Walk-Forward Validation

Walk-forward validation is a robust method for evaluating the performance of time-series models like Prophet. Instead of a simple train/test split, this method incrementally steps through the data, training the model on a 'window' of data and then testing it on the following data points, thus 'walking' the model forward through time.

The **walk_forward_validation** function in the code demonstrates this process:

1. Data is split into training and testing sets, with the **train_test_split** function determining the size of these sets.
2. The model is trained on the initial training set.
3. The model then predicts the next time step in the test set.
4. The test observation is then added to the training set, and the model is retrained for the next prediction. This step-by-step process continues until all test observations have been predicted.

5. After each prediction, the model's forecast is compared to the actual observed values, and an error metric (mean absolute error, in this case) is computed to evaluate performance

3. XGBOOST

XGBoost, a decision-tree-based ensemble machine learning algorithm that has shown superior performance, was chosen as the alternative model for several compelling reasons, particularly after observing the inadequate performance of ARIMA and Prophet in capturing the complexities of healthcare time-series data:

1. *Handling Complex Non-Linear Relationships:* Healthcare data often includes complex relationships between features that are non-linear and interactive. XGBoost is well-suited for this task, as it builds an ensemble of decision trees, which can model such non-linear interactions with high accuracy.

2. *Feature Importance and Selection:* XGBoost provides built-in methods for assessing feature importance. This is invaluable in a healthcare setting, as it allows us to understand which factors are driving the model's predictions and, consequently, which health indicators are most significant for patient outcomes.

3. *Robustness to Overfitting:* XGBoost includes regularization parameters that help prevent overfitting, which is crucial when dealing with the varied and complex data patterns found in patient health records.

4. *Scalability and Efficiency:* XGBoost is designed for efficiency and can handle large datasets without compromising on speed, making it practical for real-time healthcare monitoring and analysis.

5. *Flexibility:* The model can be used for both regression and classification problems, providing flexibility in terms of the types of predictions it can make, from forecasting patient vitals to identifying the risk of particular health events.

6. *High Predictive Power:* Empirical results have shown that XGBoost often outperforms other models on a wide range of predictive tasks due to its ability to push the limits of computing power and data science techniques.

7. *Incremental Learning:* XGBoost can be trained incrementally, a beneficial feature when dealing with streaming data where the model needs to adapt quickly to new information as it becomes available.

Functionality

Step 1: Starting Simple

- XGBoost starts with a single decision tree. Think of a decision tree as a flowchart where each branch represents a choice (like yes/no questions) that leads to a conclusion.

- The first tree makes its best attempt to predict the outcome (like diagnosing an illness based on symptoms).

Step 2: Learning from Mistakes

- After the first prediction, XGBoost examines where it went wrong. These mistakes are called 'residuals', which are just differences between the predicted and actual outcomes.
- Instead of trying to fix the first tree, XGBoost builds a new tree focused specifically on correcting those errors.

Step 3: Teamwork with Trees

- This process repeats, with each new tree focusing on the errors left by the previous ones. Each tree in the sequence adds its insights to improve the predictions gradually.
- The trees work together, each correcting the previous one's mistakes, in a process called 'ensemble learning'.

Step 4: Smarter Decisions

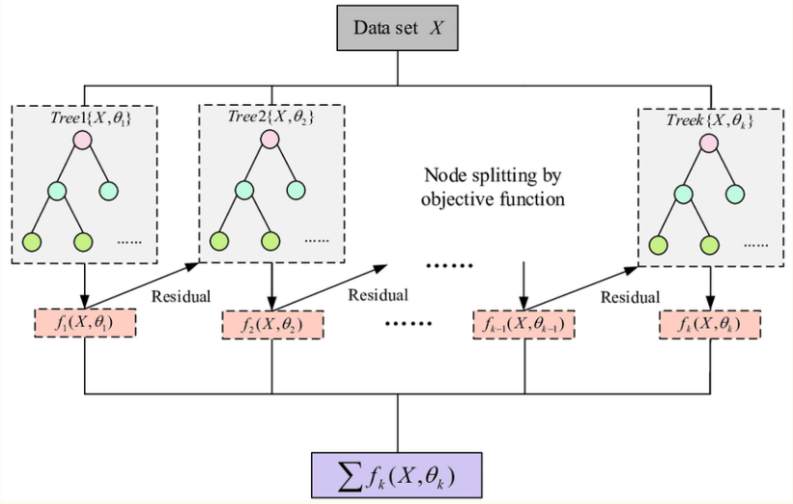
- XGBoost makes its trees smarter by using an 'objective function' to guide the splitting of branches. This function is like a coach that directs each tree on how to improve based on the errors made.
- The 'objective function' measures how well the model is doing and tells each new tree the best way to reduce the errors.

Step 5: Stronger Together

- The final prediction is a team effort. XGBoost combines the knowledge of all trees to make a final decision.
- It adds up the input from each tree, accounting for their individual strengths, to produce a final, more accurate prediction.

Step 6: Avoiding Overconfidence

- To prevent any single tree from dominating (which can lead to overconfidence in wrong predictions), XGBoost uses a technique called 'regularization'.
- Regularization is like a balancing act that keeps the trees in check, ensuring they work well together and don't go overboard on their individual predictions.



An Overview of the functionality of XGBoost

Implementation

Data Preparation

The dataset was first converted into a supervised learning format to facilitate the application of XGBoost. This process involved reorganizing the time-series data so that each input-output pair consists of historical time steps (input) and the target variable (output) to be predicted in future time steps. Specifically, the data was transformed to align the input variables, such as previous measurements of a patient's vitals, with the corresponding output variable, the subsequent measurement to be predicted.

Model Training

With the data prepared, we proceeded to train the XGBoost model. Our approach utilized the xgboost_forecast function to facilitate this process, which involved:

- *Splitting the Data:* The input data was divided into features (input columns) and the target variable (output column).
- *Model Instantiation:* An instance of the XGBoost regressor was created, specifying the objective as 'reg:squarederror' and setting the number of estimators, highlighting our focus on regression tasks with a squared error loss function.
- *Fitting the Model:* The model was then trained on the historical data, learning to predict the target variable based on the input features.
- *One-Step Prediction:* After training, the model made a one-step prediction for the test set.

Walk-Forward Validation

To assess the performance of the XGBoost model, a walk-forward validation approach was adopted. This rigorous method is particularly suited to time-series data, as it mirrors the sequential nature of real-world forecasting. The process involved:

- *Iterative Predictions:* The model made predictions one step at a time through the test dataset. After each prediction, the true observed value was added to the model's history, updating the model with the latest data before proceeding to the next prediction.
- *Performance Evaluation:* After each iterative step, the predicted value was compared to the actual observed value, with the Mean Absolute Error (MAE) being calculated to gauge the model's performance.

iv. Model Comparison

Parameter	ARIMA	Prophet	XGBoost
Model Type	Statistical	Statistical	Machine Learning
Core Use	Time-series forecasting	Time-series forecasting with daily/annual seasonality	Supervised learning for classification and regression
Primary Strength	Autocorrelation in stationary data	Trend and seasonality detection	Handling non-linear patterns
Data Requirement	Stationary time series	Time series with or without gaps	Tabular data with feature vectors
Flexibility	Low (requires stationary data)	High (handles missing data, trend changes, and outliers)	Very high (can model various types of relationships)
Interpretability	Medium (model parameters can be interpreted, but full understanding requires statistical knowledge)	High (transparent components for trend and seasonality)	Low (ensemble method can be considered a "black box")
Training Time	Fast for small datasets	Fast	Moderate to slow (depending on data size and complexity)
Hyperparameters	Few (p, d, q values)	Several (changepoint sensitivity, seasonality parameters)	Many (tree depth, learning rate, etc.)
Customizability	Limited customization of model structure	Good customization for holidays and special events	Highly customizable with various loss functions and

			regularization
Scalability	Good for small to medium-sized datasets	Good for larger datasets without too much fine-tuning	Excellent scalability and parallelization capabilities
Prediction Type	Univariate (typically)	Univariate or multivariate with additional regressors	Univariate or multivariate (more naturally handles multiple input features)
Online Learning	Not typical	Not typical	Possible with incremental training
Use Case in Project	Tested but performed poorly overall	Tested but performed poorly overall	Demonstrated superior performance
Handling of Hard Cases	Struggles with non-stationarity and complex patterns	Struggles with irregular data patterns	Better suited but may require feature engineering for complex cases

v. Hyperparameter tuning and Ensemble Model

XGBoost was selected for its robustness and its ability to handle the non-linear relationships often found in healthcare data. Unlike ARIMA and Prophet, which struggled with certain complex patterns, XGBoost's gradient boosting mechanism allowed it to improve iteratively with each round of boosting, making it a strong candidate for our needs.

Hyperparameter Tuning

With XGBoost, the key to unlocking improved performance was hyperparameter tuning. The main hyperparameters tuned in the implementation of ML models include:

1. N_estimators (Number of Estimators): Setting the number of trees in the model to strengthen the learning capacity. Set to 1000 for XGBoost
2. n_in: represents the number of past time steps that are used as input features for the model. A larger n_in allows the model to capture longer-term dependencies in the data but may also increase the dimensionality of the input.
3. n_out : represents the number of future time steps to be predicted. A larger n_out implies that the model is making predictions further into the future.

Our tuning focused on the n_estimators and n_in parameters, which showed a direct impact on the Mean Average Error (MAE) of the predictions. Increasing n_in allowed the model to consider more past data points, thereby capturing longer-term dependencies.

Ensemble Method

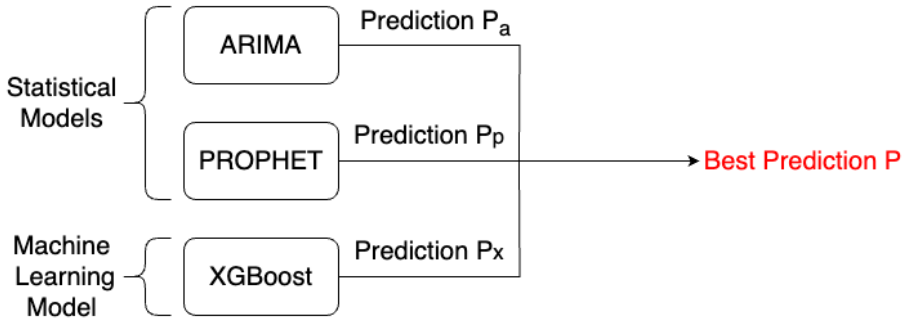
Despite the gains from hyperparameter tuning, some portions of the dataset remained challenging. To address this, we turned to an ensemble method, combining the strengths of ARIMA, Prophet, and XGBoost. The rationale behind this strategy was:

- *Diversity in Approach:* Each model has unique strengths; ARIMA is excellent at capturing linear relationships, Prophet is adept at handling seasonality, and XGBoost excels at modeling complex interactions.

- *Robustness in Predictions:* An ensemble mitigates the risk of relying on a single model, potentially smoothing out the weaknesses inherent in any individual approach.

Implementation of Ensemble

The ensemble approach involved using the predictions from ARIMA and Prophet as additional features for XGBoost or taking a weighted average of the predictions from all three models. We have assigned a weight 0.5 to XGBoost and 0.25 to Prophet and ARIMA since XGBoost was observed to be the best performing model. This method acknowledged that while each model had its shortcomings, they could complement each other to achieve better accuracy collectively.



The combined strategy of hyperparameter-tuned XGBoost and the ensemble of models led to a noticeable improvement in the predictions across the board. For the challenging fraction of the dataset, the ensemble method provided a performance boost that was not achievable by any single model alone.

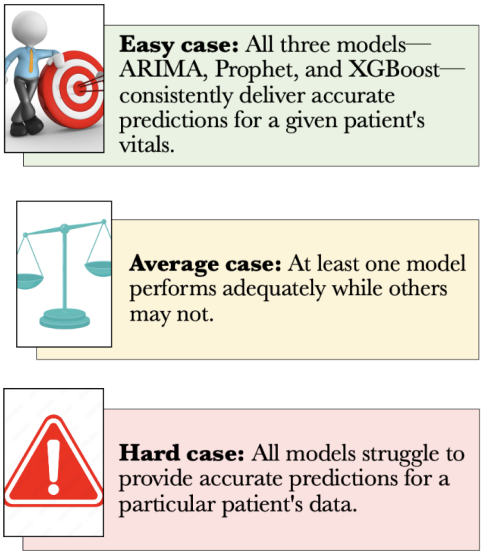
This iterative and multi-faceted approach to model enhancement underscores the complexity of predictive modeling in healthcare. It showcases the necessity of continuous model evaluation and adaptation, the power of machine learning algorithms like XGBoost, and the benefits of combining different predictive models to achieve the best possible outcomes in patient care analytics.

Overall Results and Evaluation

i. Evaluation scenarios: How do we evaluate the system?

Let us first define the different scenarios for evaluation:

1. **Easy case**: In this case, all the three models, Prophet, Arima and XGBoost, perform well. All the three models are able to predict heart rate and respiratory rate values that are closest to the actual values of the patients. This shows us how many patients have trends that the models can capture easily.
2. **Average case**: At least one model is able to pick up on the trends in the patients' heart rate and respiratory rate values. This case shows us the cases where the trends in the patients' data are harder to capture but some of the models are still performing relatively well.
3. **Hard case**: For this case, none of the three models pick up the trends in the data. This case constitutes the patients with data that is highly varying and inconsistent. Therefore, it is difficult for the models to continue these trends.



Easy Case Validation

In the easy case scenario, both the models—ARIMA and Prophet consistently deliver accurate predictions for a given patient's vitals. This implies that the underlying data patterns are well-defined and that each model's intrinsic mechanisms align well with the data's structure.

Approach:

- ARIMA validation involves checking the model's ability to capture the autocorrelations in the patient's time-series data without overfitting.
- Prophet is validated by its performance in forecasting trends and seasonality in the data, with particular attention to the model's flexibility in handling any irregular events or outliers.

The easy case suggests that the patient's data contains clear signals that all models can leverage to make precise forecasts, reflecting a best-case scenario in terms of data predictability and model performance.

Hard Case Validation

The hard case represents a situation where all models struggle to provide accurate predictions for a particular patient's data. This could be due to high noise levels, non-stationarity, or complex dynamics in the patient's vitals that do not align with the assumptions or strengths of the models.

Approach:

- ARIMA is evaluated on its ability to make sense of the data despite potential non-stationarity and high volatility in the time series.
- Prophet is scrutinized for its trend and seasonality modeling capabilities, especially when the data lacks clear or consistent patterns.

In the hard case, the consistent poor performance across all models indicates that the data may be inherently difficult to predict with current modeling techniques or that additional feature engineering, data cleansing, or alternative modeling approaches may be necessary.

Average Case Validation

The average case scenario presents a more nuanced challenge where at least one model performs adequately while others may not. This indicates that certain aspects of the data are captured by one model's strengths but may be missed by the others.

Approach:

- ARIMA is assessed on its traditional time-series forecasting strengths, focusing on simpler, linear relationships within the data.
- Prophet is expected to excel in cases where the data exhibits clear trends or seasonal effects, even if these patterns are not immediately evident to the other models.

The average case is indicative of the varying capabilities of the models and underscores the importance of leveraging an ensemble approach. By doing so, the strengths of one model can compensate for the weaknesses of others, leading to a more robust predictive system overall.

Validating ARIMA and Prophet models across easy, hard, and average cases allows us to understand and optimize their performance in forecasting patient vitals. This diverse validation strategy ensures that the models are not only tested against data they are well-suited to but also challenged by more complex datasets. The outcome of these validation cases informs how these models can be fine-tuned or combined to improve the overall predictive accuracy and reliability of the healthcare monitoring system.

ii. Evaluation metrics

For time series analysis models, the most popularly used evaluation metrics include:

1. **Training time:**
   - **Definition**: The time taken by the model to learn from the historical data during the training phase.
   - **Significance**: Faster training times are generally desirable, especially in real-time or near-real-time applications.

2. **Testing time**:
   - **Definition**: The time taken by the model to make predictions on new, unseen data during the testing or validation phase.
   - **Significance**: Similar to training time, lower testing times are preferred, especially in applications where quick decision-making is crucial.

3. **Accuracy**:
   - **Definition**: The ratio of correctly predicted instances to the total instances in the dataset.
   - **Significance**: Accuracy is a common metric but may not be suitable for imbalanced datasets or when false positives and false negatives have different implications.

4. **F1-Score**:
   - **Definition**: The harmonic mean of precision and recall. It is particularly useful when the classes are imbalanced.
   - **Significance**: F1-score considers both false positives and false negatives, making it a good metric when there is an uneven distribution of classes.

5. **Precision**:
   - **Definition**: The ratio of true positives to the sum of true positives and false positives.
   - **Significance**: Precision focuses on minimizing false positives, which is important in scenarios where the cost of false positives is high.

6. **Mean Average Error (MAE)**:
   - **Definition**: The average absolute difference between the predicted and actual values.
   - **Significance**: MAE provides a measure of the average magnitude of errors, and it is easy to interpret. It is suitable for regression tasks in time-series analysis.

When working with time-series data, it's crucial to consider the temporal aspects, such as autocorrelation, seasonality, and trend, which we evaluate by looking at the results generated for the test data patients. The walk-forward validation is also a common technique for evaluating model performance in a time-dependent setting.

Train-test split: For all the three time-series models, the train set consists of 2800 data points per patient (266 patients in total), while the test set comprises 100 data points per patient.

iv. Evaluation Phases

On evaluating the models' performances according to the three evaluation scenarios (easy, average and hard), we had to iteratively identify the reasons why the models were not working and what improvements could be made. This led to three different phases of our project implementation:
Certainly, let's delve into each phase of your project implementation, exploring the reasons for success or failure and suggesting potential improvements:

Phase 1: Implementation of Statistical Models (ARIMA and Prophet)

- **Why it works:**
  - ARIMA is effective for capturing linear trends and seasonality in time-series data. It works well when the underlying patterns are relatively simple and can be represented with autoregressive and moving average components.
  - Prophet is designed to handle time-series data with seasonality and holiday effects. It can capture sudden changes in the trend and works well when there are known events or holidays influencing the time series.

- **Why it doesn't work:** ARIMA and Prophet may struggle with capturing complex, non-linear patterns or irregular events that do not follow a clear seasonal or trend structure.

- Improvements:
  - Feature Engineering: Explore additional features or transformations that might help capture complex patterns in the data.
  - Ensemble Approaches: Combine the predictions from ARIMA and Prophet, as ensemble methods often perform better by leveraging the strengths of multiple models.

Phase 2: Implementation of Machine Learning Model (XGBoost)

- Why it works:
  - XGBoost is a powerful machine learning algorithm that can capture complex relationships and interactions in the data. It works well when there are non-linear patterns, and it can handle a large number of features.

- Why it doesn't work:

Overfitting: XGBoost may be prone to overfitting, especially if the model is too complex or if there is noise in the training data.

- Improvements:

- Hyperparameter Tuning: Fine-tune the hyperparameters of XGBoost to optimize its performance and reduce overfitting.
- Feature Selection: Identify and select the most relevant features to improve the model's generalization to new data.

Phase 3: Implementation of Ensemble Model for Time-Series Predictions

- **Why it works:**

- Combining predictions from multiple models can often result in improved accuracy and robustness. It helps mitigate the weaknesses of individual models.

- Why it doesn't work:
Model Diversity: If the individual models in the ensemble are too similar or have similar weaknesses, the ensemble may not provide significant improvements.

Improvements:

- **Diverse Models:** Introduce more diversity into the ensemble by incorporating models with different underlying algorithms or architectures.
- **Dynamic Ensemble:** Consider dynamically adjusting the weights assigned to each model in the ensemble based on their performance on recent data.

v. Results and Analysis - Phase 1

This section details the results obtained on implementing the time-series models on our custom dataset of 266 patients.

Table 4: Performance of statistical models Arima and Prophet

Model	Training time	Testing time	Accuracy	MAE across all patients	# Patients where trends followed	# Patients where trends not followed
Arima	40.2 mins	1.47 mins	73%	2.4	186	80
Prophet	102.4	2.5 mins	64%	3.5	120	186

	mins					

The evaluation of ARIMA and Prophet models for time-series analysis in patient outcomes reveals notable differences in their performance metrics. ARIMA demonstrates efficiency with a reasonable training time of 40.2 minutes and a relatively low testing time of 1.47 minutes. Achieving an accuracy of 73% and a Mean Absolute Error (MAE) of 2.4, ARIMA successfully identifies trends in 186 patients, though it faces challenges in 80 cases where trends are not accurately captured. In contrast, Prophet exhibits a longer training time of 102.4 minutes but maintains a reasonable testing time of 2.5 minutes. However, its accuracy of 64% and higher MAE of 3.5 indicate a lower overall correctness in predictions compared to ARIMA. Prophet successfully captures trends in 120 patients, but struggles in 186 cases where trends are not accurately followed.

Comparatively, ARIMA outperforms Prophet in terms of accuracy, MAE, and the number of patients where trends are successfully identified. Despite their respective strengths, both models encounter challenges in certain patient scenarios, emphasizing the need for further refinement. Recommendations include fine-tuning model parameters, exploring ensemble approaches, and conducting patient-specific analyses to address cases where trends are not accurately identified. Continuous monitoring and adaptation are crucial for optimizing these models in dynamic healthcare settings. By implementing these refinements, the overall prediction capabilities of ARIMA and Prophet in patient outcomes can be enhanced, fostering more reliable and accurate time-series analyses.

Now, let's look at some individual patient cases to analyze trends and to check if the models are able to capture these trends:

For example patient 138022, both Arima and Prophet are able to capture the inherent trend in the dataset. The graphs below show that not only the trends were followed, but Arima was also able to predict values very close to the actual heart rate values.

Heart Rate:

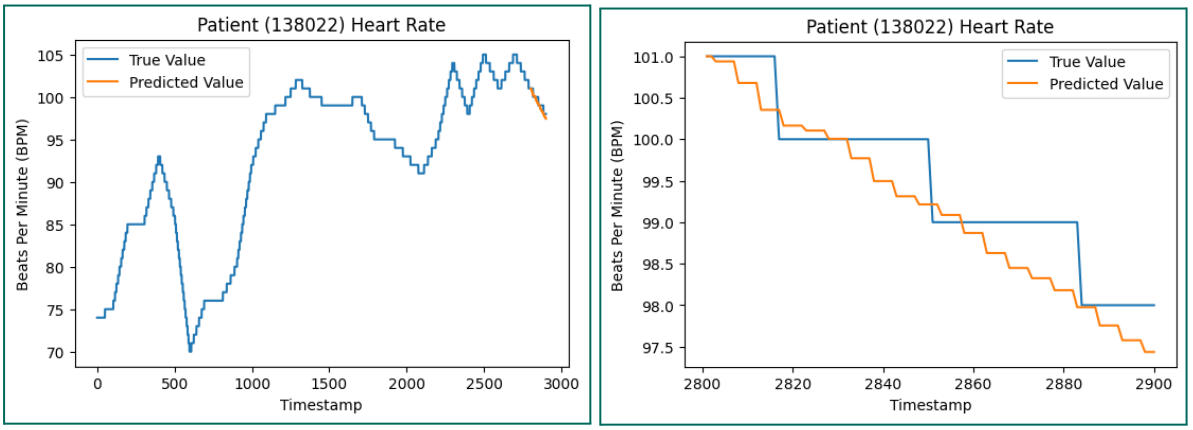


Figure 5: **Arima's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the

graph on the left gives a closer look into the patient's heart rate predictions for these test data points.

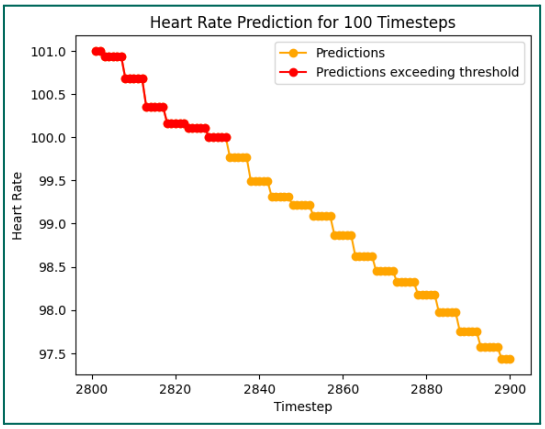


Figure 6: This graph shows predicted values by Arima that exceed the normal heart rate range of 60-100 in red. These values will be used later to perform further analysis on the patient's overall health condition.

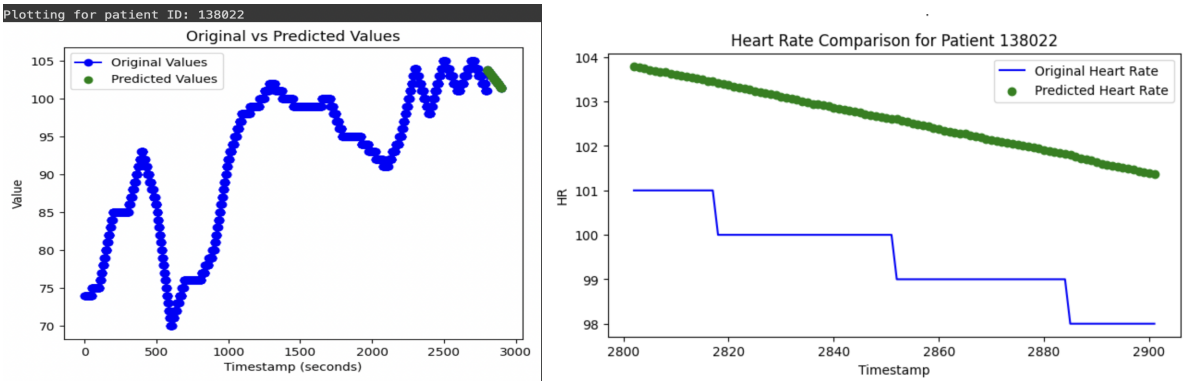


Figure 7: **Prophet's** results for the same example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's heart rate predictions for these test data points.

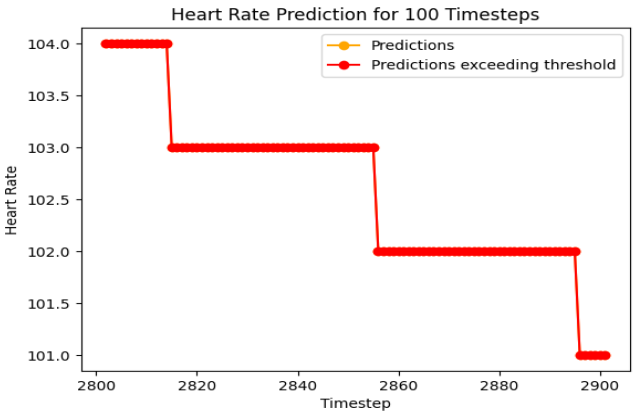


Figure 8: This graph shows predicted values by Prophet that exceed the normal heart rate range of 60-100 in red. These values will be used later to perform further analysis on the patient's overall health condition.

Respiratory Rate:

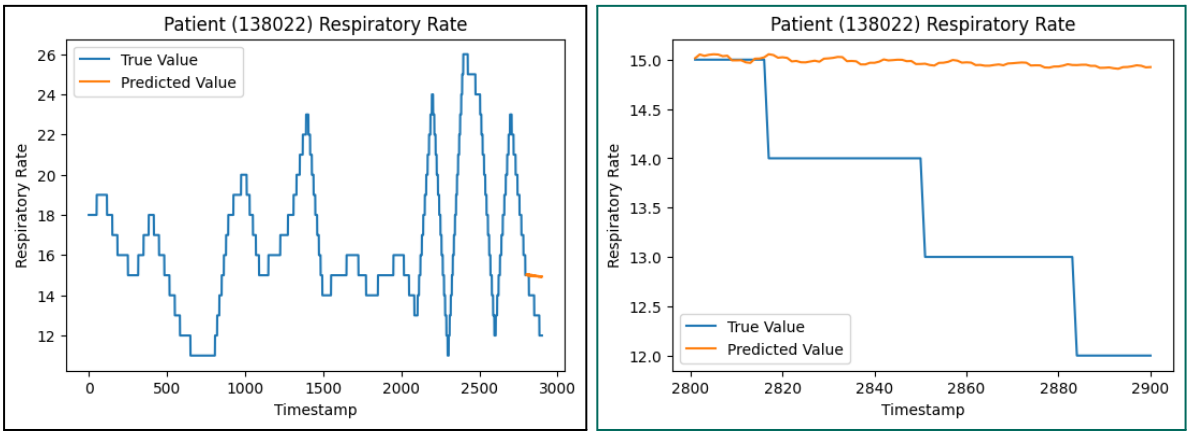


Figure 9: **Arima's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's respiratory rate predictions for these test data points.

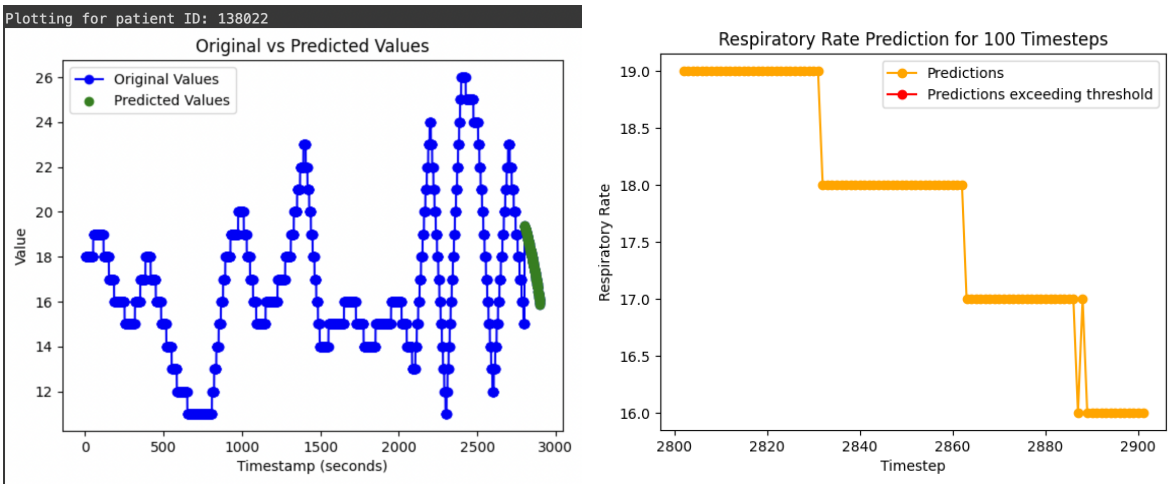


Figure 10: **Prophet's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's respiratory rate predictions for these test data points.

Although these models worked for many patients, there was a good set of patients for which they did not work as illustrated in Table 4. Let's look at a case where both these models failed to perform well.

Heart Rate:

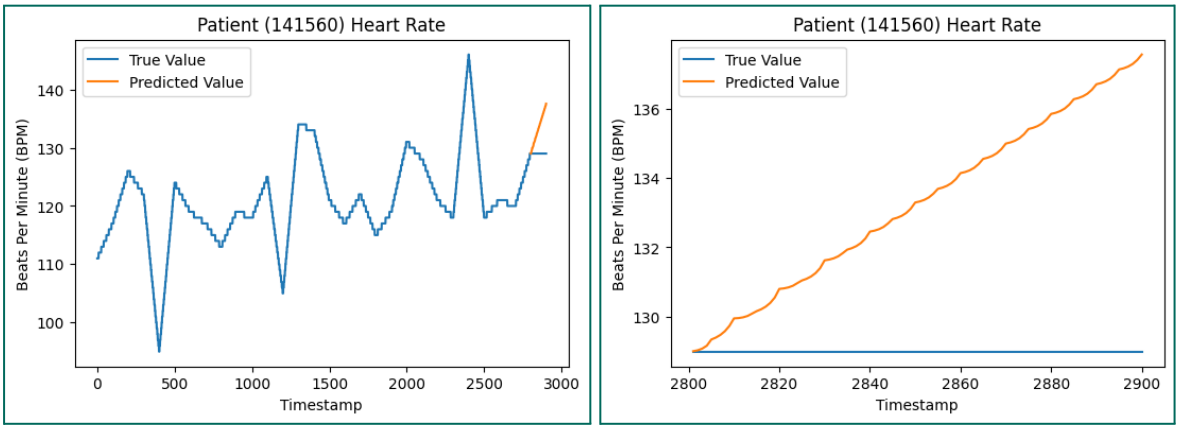


Figure 11: **Arima's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's heart rate predictions for these test data points.

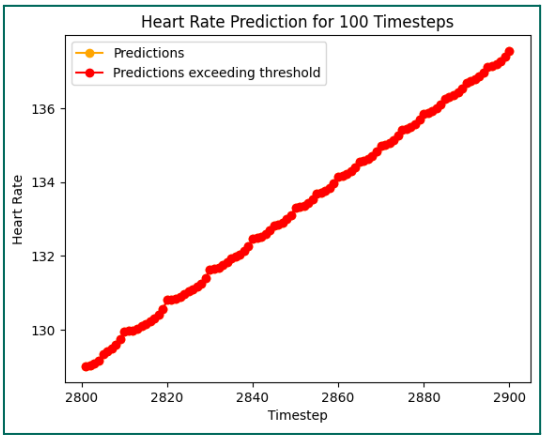


Figure 12: This graph shows predicted values by Arima that exceed the normal heart rate range of 60-100 in red. These values will be used later to perform further analysis on the patient's overall health condition.

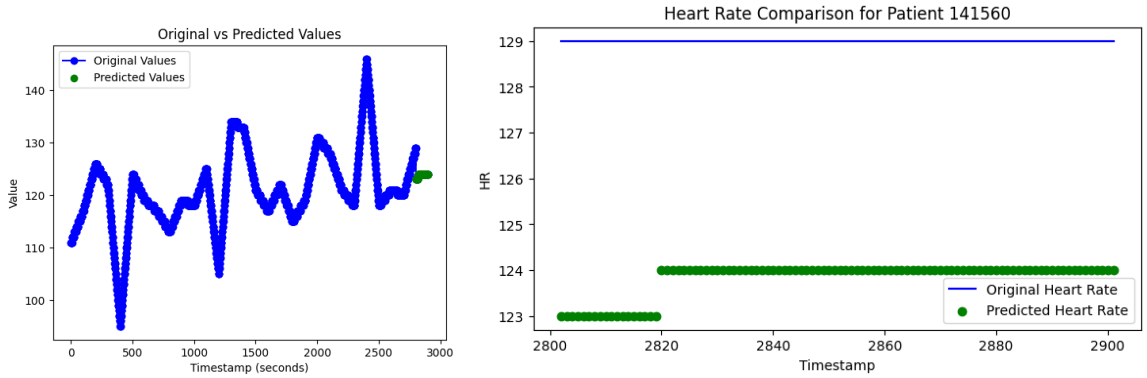


Figure 13: **Prophet's** results for the same example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's heart rate predictions for these test data points.

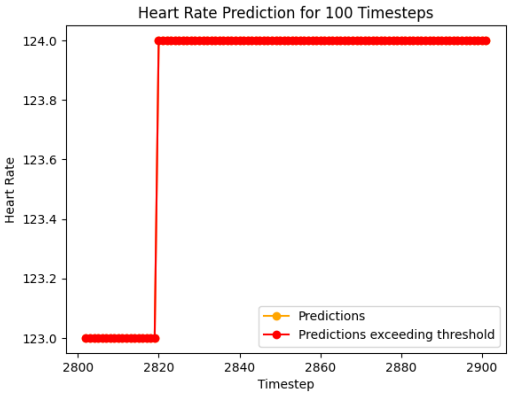


Figure 14: This graph shows predicted values by Prophet that exceed the normal heart rate range of 60-100 in red. These values will be used later to perform further analysis on the patient's overall health condition.

Respiratory Rate:

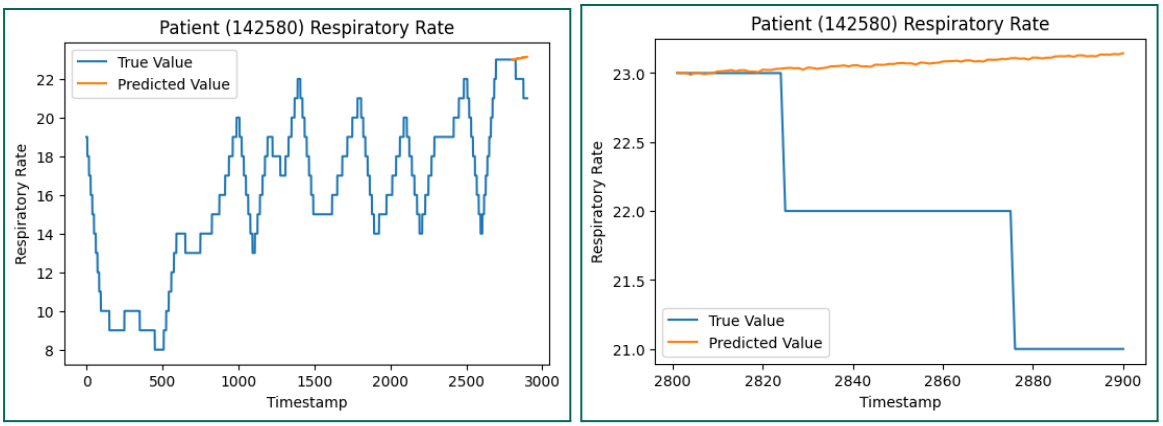


Figure 15: **Arima's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's respiratory rate predictions for these test data points.

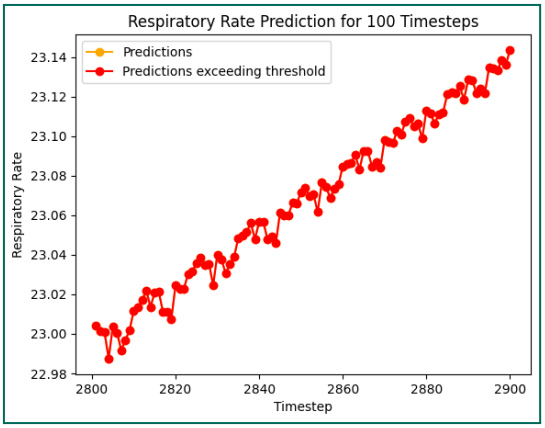


Figure 16: This graph shows predicted values by Arima that exceed the normal respiratory rate range of 12-20 in red. These values will be used later to perform further analysis on the patient's overall health condition.

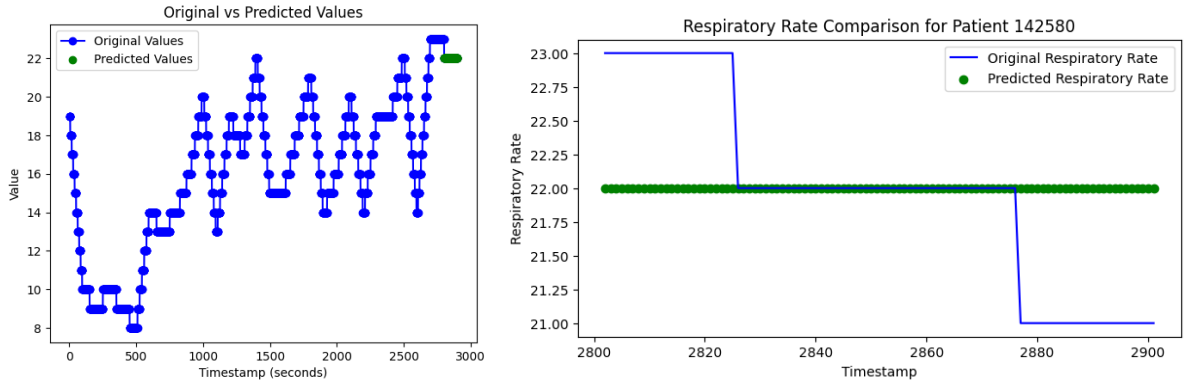


Figure 17: **Prophet's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's respiratory rate predictions for these test data points.

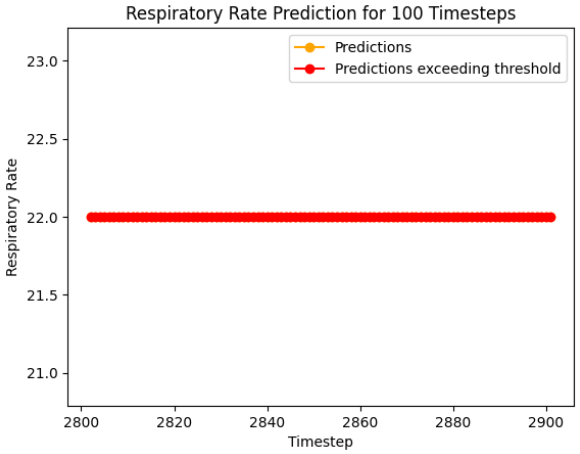


Figure 18: This graph shows predicted values by **Prophet** that exceed the normal respiratory rate range of 12-20 in red. These values will be used later to perform further analysis on the patient's overall health condition.

<u>Performance Challenges with ARIMA and Prophet</u>

In the dynamic and complex field of healthcare predictive analytics, the performance of forecasting models can significantly impact patient outcomes. While ARIMA and Prophet models have demonstrated promise in various domains, their application within our project yielded suboptimal results. ARIMA and Prophet models, despite their sophistication in handling time-series data, struggled with the intricate patterns and non-linear relationships present in our healthcare datasets. Their intrinsic structures were not suited to model the complexities of individual patient health trajectories effectively. In pursuit of excellence and reliability, we turned to using the **alternative model approach.**

Since there are considerable cases where these models fail to perform well, we move onto phase 2 of implementation where we implemented XGBoost, a machine learning model, to test if it can give better results.

vi. Results and Analysis - Phase 2

This section details the results obtained on implementing the machine learning model **XGBoost** on our custom dataset of 266 patients.

Table 5: Performance of statistical models Arima and Prophet vs machine learning model XGBoost

Model	Training time	Testing time	Accuracy	MAE across all patients	# Patients where trends followed	# Patients where trends not followed
Arima	40.2 mins	1.47 mins	73%	2.4	186	80
Prophet	102.4 mins	2.5 mins	64%	3.5	120	186
XGBoost	22.3	32 secs	86%	1.2	203	63

Table 5 provides a comprehensive overview of the performance of statistical models Arima and Prophet compared to the machine learning model XGBoost in the context of time-series analysis for patient outcomes. Arima, with a training time of 40.2 minutes and testing time of 1.47 minutes, achieves an accuracy of 73% and a Mean Absolute Error (MAE) of 2.4. It successfully identifies trends in 186 patients, but encounters challenges in 80 cases where trends are not accurately captured. Prophet, on the other hand, has a longer training time of 102.4 minutes, a testing time of 2.5 minutes, an accuracy of 64%, and a higher MAE of 3.5. It captures trends in 120 patients but struggles in 186 cases where trends are not accurately followed. In contrast, the machine learning model XGBoost stands out with a notably shorter training time of 22.3 minutes and a rapid testing time of 32 seconds. XGBoost achieves an impressive accuracy of 86% and a lower MAE of 1.2. It successfully identifies trends in 203 patients, outperforming both Arima and Prophet, and encounters challenges in only 63 cases where trends are not accurately captured.

The results highlight the superiority of XGBoost over Arima and Prophet in terms of accuracy, training and testing efficiency, and trend identification. XGBoost's ability to capture complex relationships and patterns in the time-series data is evident in its higher accuracy and lower MAE. This suggests that, in the context of patient outcomes in a time-series setting, the machine learning approach of XGBoost outperforms traditional statistical models, offering a promising avenue for further exploration and application in healthcare analytics.

Now, let’s look at some individual patient cases to analyze trends and to check if the models are able to capture these trends:

For example patient 138022, XGBoost is able to capture the inherent trend in the dataset. The graphs below show that not only the trends were followed, but XGBoost was also able to predict values very close to the actual heart rate values.

Heart Rate:

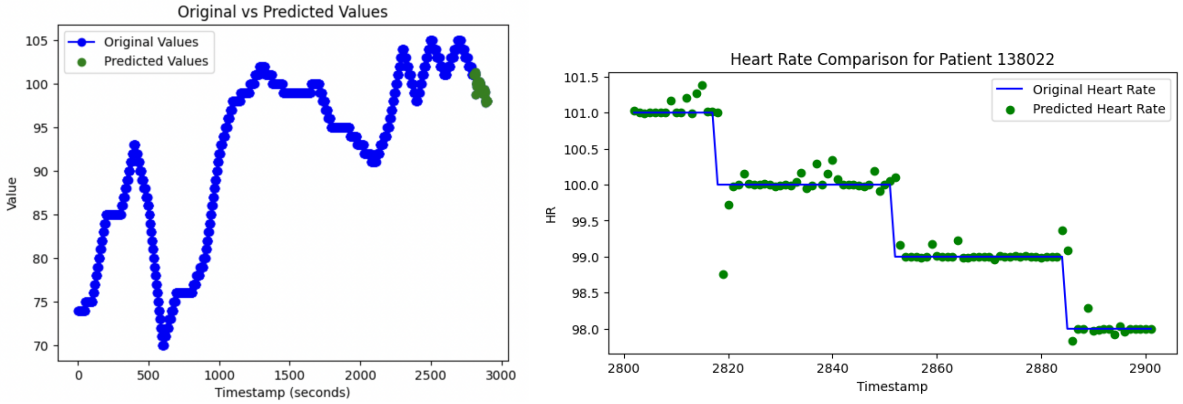


Figure 19: **XGBoost's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's heart rate predictions for these test data points.

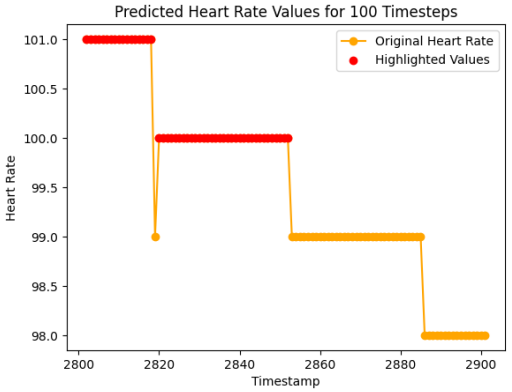


Figure 20: This graph shows predicted values by **XGBoost** that exceed the normal heart rate range of 60-100 in red. These values will be used later to perform further analysis on the patient's overall health condition.

Respiratory Rate:

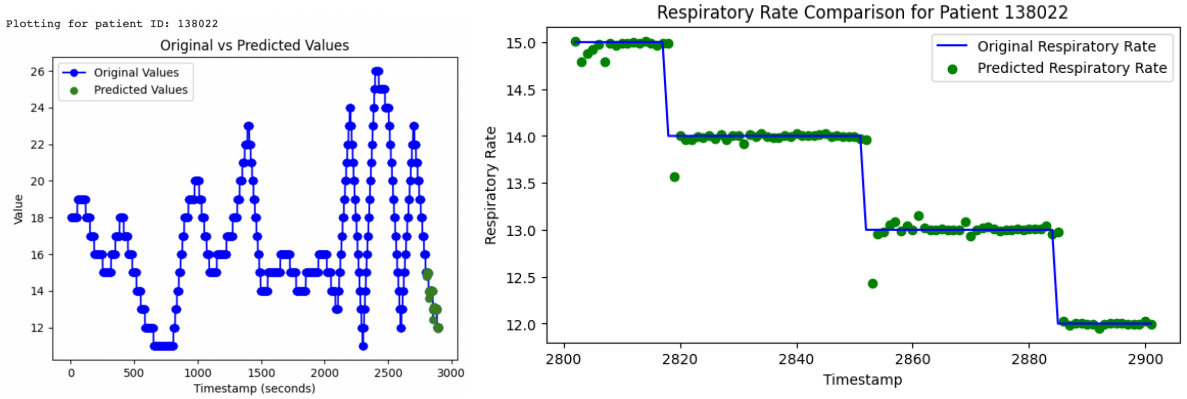


Figure 21: **XGBoost's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's respiratory rate predictions for these test data points.

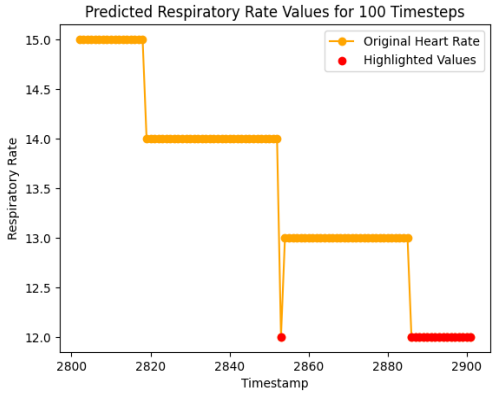


Figure 22: This graph shows predicted values by **XGBoost** that exceed the normal respiratory rate range of 12-20 in red. These values will be used later to perform further analysis on the patient's overall health condition.

Although XGBoost works considerably better due to the following reasons:
- XGBoost excels in capturing nonlinear trends in medical datasets, leveraging its ability to model complex relationships through decision trees.
- XGBoost's ensemble structure enables effective handling of temporal dependencies, allowing it to accurately predict heart rate and respiratory rate trends over time.

There are cases where XGBoost does not perform very well, with room for improvement. Let's consider a particular patient's results for XGBoost:

Heart Rate:

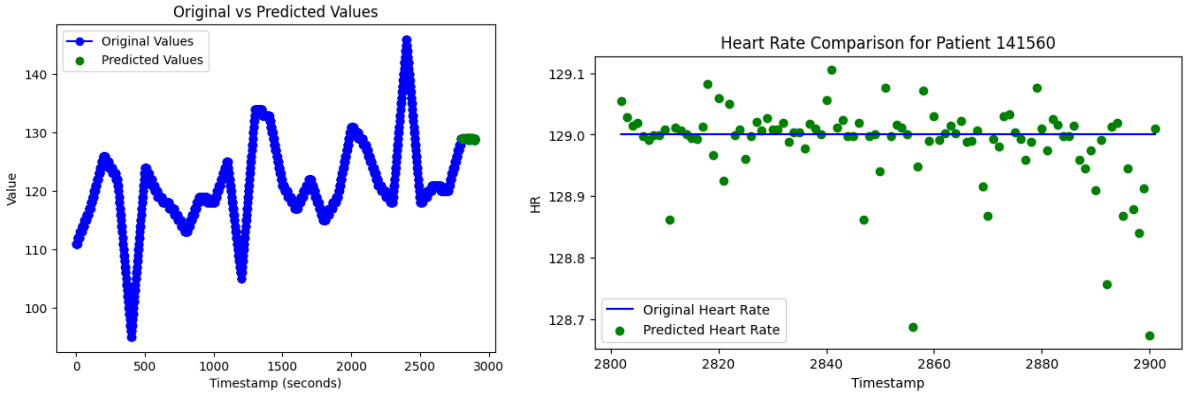


Figure 19: **XGBoost's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's heart rate predictions for these test data points.

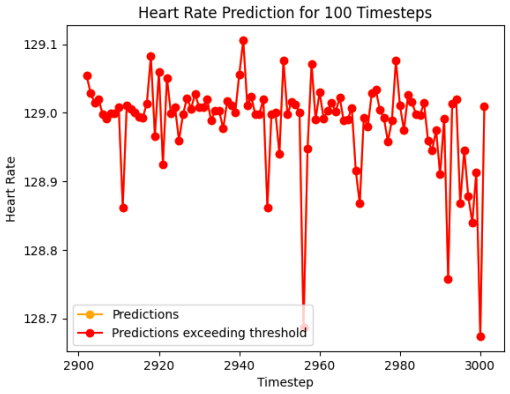


Figure 20: This graph shows predicted values by **XGBoost** that exceed the normal heart rate range of 60-100 in red. These values will be used later to perform further analysis on the patient's overall health condition.

Respiratory Rate:

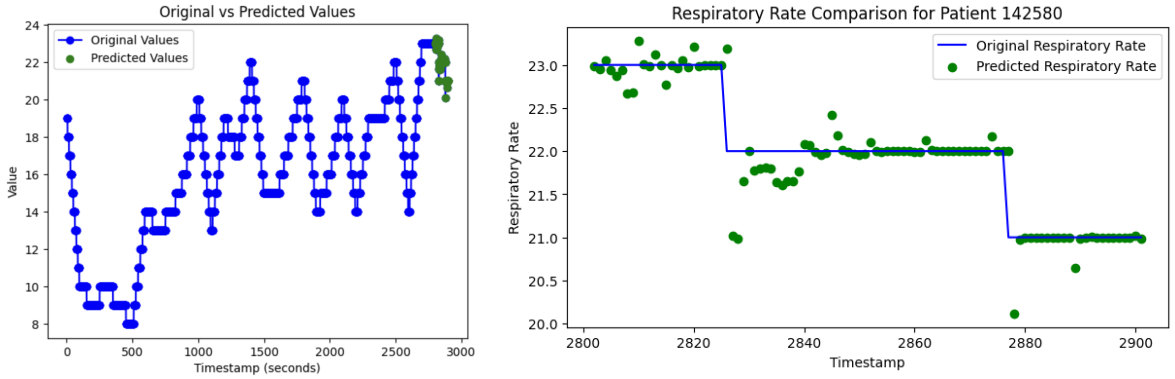


Figure 23: **XGBoost's** results for an example patient. The graph on the left shows the predicted values for the test data points (last 100 data points) in orange and the graph on the left gives a closer look into the patient's respiratory rate predictions for these test data points.

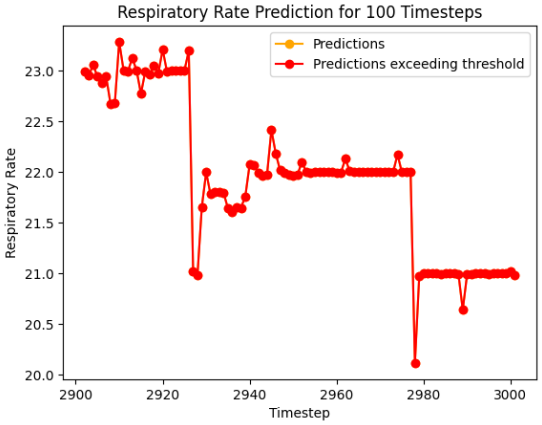


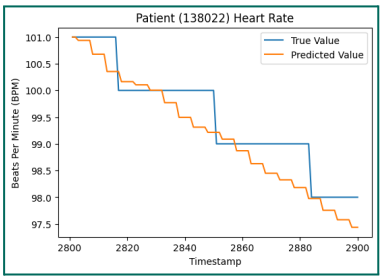
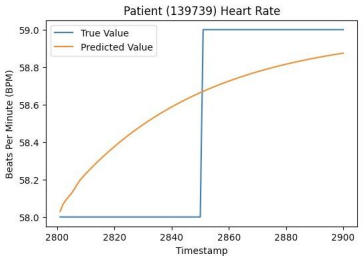
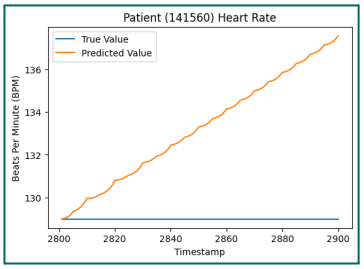
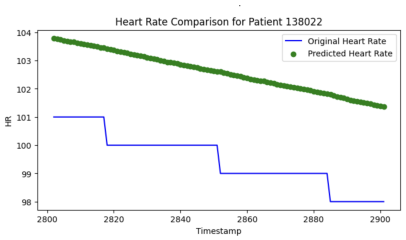
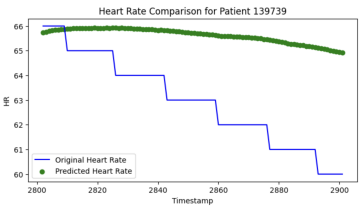
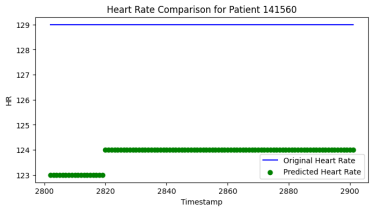
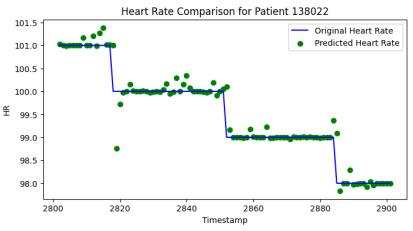
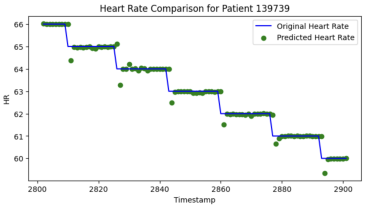
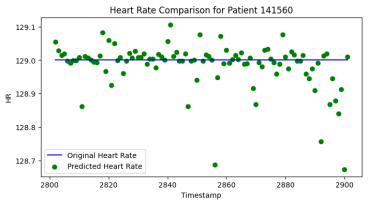
Figure 24: This graph shows predicted values by **XGBoost** that exceed the normal respiratory rate range of 12-20 in red. These values will be used later to perform further analysis on the patient's overall health condition.

In order to understand how the three models work, we need to further evaluate them for all the evaluation scenarios of easy, average and hard cases which is detailed in Phase 3 with suggested improvements.

Table 6: Overview of results comparing the three models.

Model	Easy Case	Average Case	Hard Case
Arima	✅	✅	❌
Prophet	✅	❌	❌
XGBoost	✅	✅	❌

Table 7: Overview of results comparing the three models.

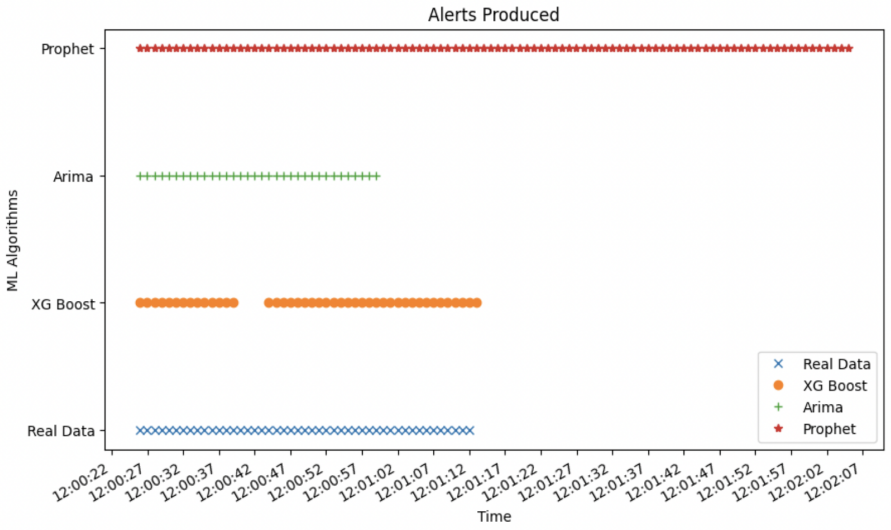
Model	Easy Case	Average Case	Hard Case
Arima			
Prophet			
XGBoost			

We evaluated XGBoost on the same cases run using ARIMA and Prophet based models and we noticed a significant improvement in the results obtained and a minimized error associated with it.

Validating time-series models results with real-time system results:

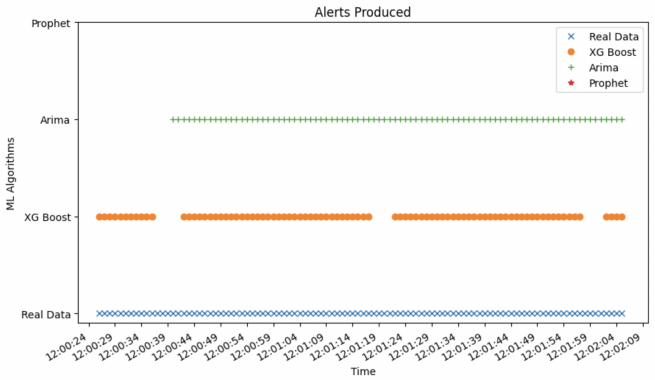
Once we had the predictive datapoints, using the real-time system, we generated alerts and compared them with the alerts on the ground truth. Following is the plot result we see.

Easy Case:



Model	Arima	Prophet	XGBoost
Accuracy	0.87	0.47	0.95
Precision	1.0	0.47	0.977
Recall	0.72	1.0	0.914
F1-Score	0.839	0.639	0.945

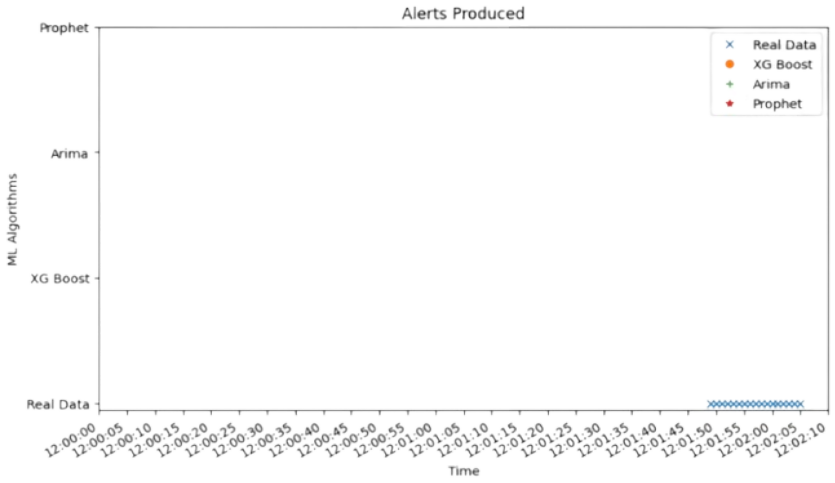
Average Case:



Model	Arima	Prophet	XGBoost

Accuracy	0.86	0	0.87
Precision	1	N/A	1
Recall	0.86	0	0.87
F1-Score	0.92	0	0.93

Hard Case:



Model	Arima	Prophet	XGBoost
Accuracy	0.83	0.83	0.87
Precision	N/A	N/A	N/A
Recall	0	0	0.2
F1-Score	0	0	0.4

Initial trials with ARIMA and Prophet were less than satisfactory, leading us to implement XGBoost, which demonstrated improved performance. Despite its strengths, XGBoost still faced challenges on a segment of the dataset, which we addressed through hyperparameter tuning. An ensemble of all three models ultimately provided the most robust predictions.

The evaluation was structured around three scenarios: easy, average, and hard cases, each defining the level of difficulty the models faced in making accurate predictions.

Easy Case

ARIMA, Prophet, and XGBoost all exhibited strong performance in the easy case, indicating that the patterns within the data were clear and well-defined. Each model was able to leverage its intrinsic mechanisms to forecast accurately, suggesting a favorable dataset structure for time-series modeling.

Average Case

In the average case, ARIMA showed reliable performance, while Prophet faltered, potentially due to its sensitivity to outlier effects or complex seasonal patterns not present in the data. XGBoost continued to maintain good performance, highlighting its ability to manage diverse data structures.

Hard Case

The hard case revealed the limitations of all models, with each struggling to provide accurate predictions. This suggested the presence of noise, non-stationarity, or complex data dynamics that challenged the models' standard assumptions and capabilities.

The reasons for XGBoost performing better in these areas could be -

- XGBoost validation focuses on the model's capacity to decipher complex, non-linear interactions between the features representing a patient's health status.

- XGBoost is likely to succeed in scenarios where the key to prediction lies in capturing complex interactions between multiple health indicators, an area where traditional time-series models may falter.

Although XGBoost performed much better on the dataset compared to ARIMA and Prophet, we observed that there were still some cases (hard case of XGBoost) where the model did not perform as well as it did for some other patients. This could be because XGBoost is challenged by data that may not have strong feature interactions or where the feature space does not provide sufficient information for the model to learn from.

Approaches to handle the hard cases in XGBoost -

Feature Engineering: Investigate and create more informative features from the existing data. This can include interaction terms, polynomial features, or more complex transformations based on domain knowledge. For example, considering temporal features that capture changes over time or physiological interactions might uncover new patterns.

Data Augmentation: Incorporate additional data sources that could enrich the model's training set. This might include patient demographics, historical health records, or external data such as seasonal health trends.

Advanced Preprocessing: Apply more sophisticated data preprocessing techniques such as normalization or standardization, which might help in cases where the scale of the data affects the model's ability to learn.

Anomaly Detection: Implement a separate anomaly detection system to identify and handle outliers or noise in the data, which can distort the model's learning process.

Dimensionality Reduction: Use techniques like PCA (Principal Component Analysis) to reduce the feature space to the most informative set of features, potentially eliminating noise and focusing the model's learning on the most relevant aspects of the data.

Ensemble Learning: Combine XGBoost with other models in an ensemble to complement its predictive power. For example, a model that performs well on the hard cases could be used in conjunction with XGBoost to improve overall accuracy.

Hyperparameter Optimization: Conduct a thorough search for the best hyperparameters using methods like grid search, random search, or Bayesian optimization to find the optimal configuration for XGBoost that might be more suited to the difficult parts of the dataset.

Model Stacking: Use stacking techniques where the predictions of XGBoost are combined with predictions from other types of models. The final prediction is made by a meta-learner that learns from all individual models' predictions.

Incremental Learning: If the dataset is continuously growing, use an incremental learning approach where XGBoost is updated as new data comes in, which may help the model to adapt to new patterns that were not present in the initial dataset.

Custom Loss Functions: Customize the loss function used by XGBoost to better capture the penalties of mispredictions in the hard cases, which can steer the model towards better performance on those specific instances.

Cross-validation Techniques: Utilize more sophisticated cross-validation techniques that can help in understanding the model's performance variations across different subsets of the data.

As mentioned above, hyperparameter tuning and an ensemble approach are two aspects that we focused on to help improve the model performance.

Further Analysis:

1. In healthcare, false negatives can be very risky, as they might mean missing out on diagnosing a condition. Hence, a high recall is important. In the previous experiments, we saw that XGBoost generally provided this, except in one evaluation (0.64), but this still greater than other models.

2. A high precision is also critical to avoid unnecessary treatments or anxiety due to false positives. Both ARIMA and XGBoost excel in this, but the consistency of XGBoost makes it more reliable.

3. The variability in Prophet's performance suggests it might be less reliable for this particular application or that it requires further tuning and validation.

4. F1-Score provides a balance between precision and recall, and XGBoost consistently shows the highest F1-Scores, suggesting it might be the most reliable model for deployment in healthcare scenarios.

v. Results and Analysis - Phase 3

Hyperparameter Tuning of XGBoost

In order to improve the results after phase 3, we had to perform hyperparameter tuning to deal with the hard case.

n_in	n_out	n_estimators	# Epochs	Learning Rate	MAE
50	50	50	20	0.01	1.2
100	100	100	50	0.1	0.92
200	200	200	100	0.3	1.3

1. n_in and n_out: These parameters often represent the number of input time steps (n_in) and the number of output time steps (n_out) in a time series prediction model. Larger values for n_in and n_out allow the model to capture more historical information and predict further into the future.

2. n_estimators: The number of boosting rounds or trees to build in the XGBoost model. Increasing n_estimators can improve the model's ability to learn complex patterns, up to a certain point. The values chosen here (50, 100, 200) indicate an exploration of different complexities.

3. Number of epochs: The number of epochs in the training process for a neural network. It represents the number of times the entire training dataset is passed forward and backward through the neural network. Higher epochs allow the model to learn more from the data.

4. Learning Rate: Learning rate is a hyperparameter that controls the step size during the optimization process. Smaller values (like 0.01, 0.1, 0.3) are often chosen to prevent overshooting the optimal solution and to facilitate better convergence.

5. MAE: MAE is a metric used to evaluate the performance of the model. Lower MAE values (like 1.2, 0.92, 1.3) indicate better accuracy, as they represent smaller average absolute errors between predicted and actual values.

The chosen values for n_in, n_out, n_estimators, # Epochs, and Learning Rate cover a range of scenarios, allowing for a systematic exploration of model complexity, training duration, and optimization settings. The progression in values indicates an iterative refinement process, starting with a moderate level of complexity and increasing gradually to assess model performance at different levels of sophistication. The selected Learning Rates (0.01, 0.1, 0.3) follow a common practice of trying different magnitudes to find the optimal balance between convergence speed and precision.

The values highlighted in green in the table indicate an improvement in the MAE for XGBoost for the particular set of hyperparameter values which shows that hyperparameter tuning has improved the results.

Although hyperparameter tuning helped improve the overall MAE of the XGBoost model, the MAE increased as the input data points (n_in) increased as shown in Fig. 25. Hence, we look at ensemble modeling that produces the best output prediction for a given patient.

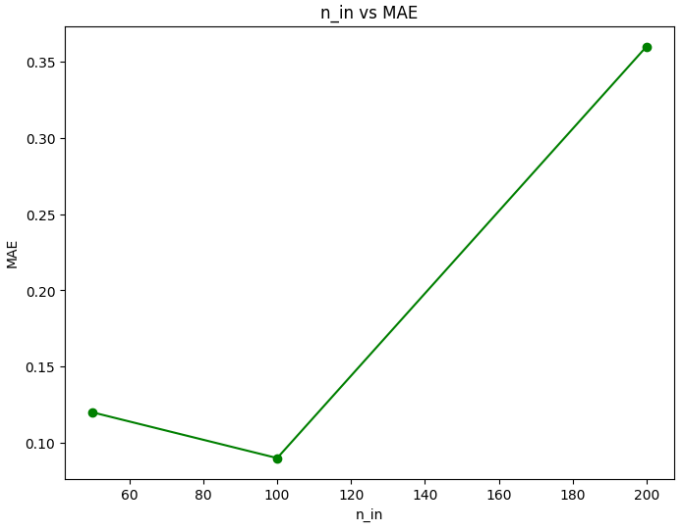
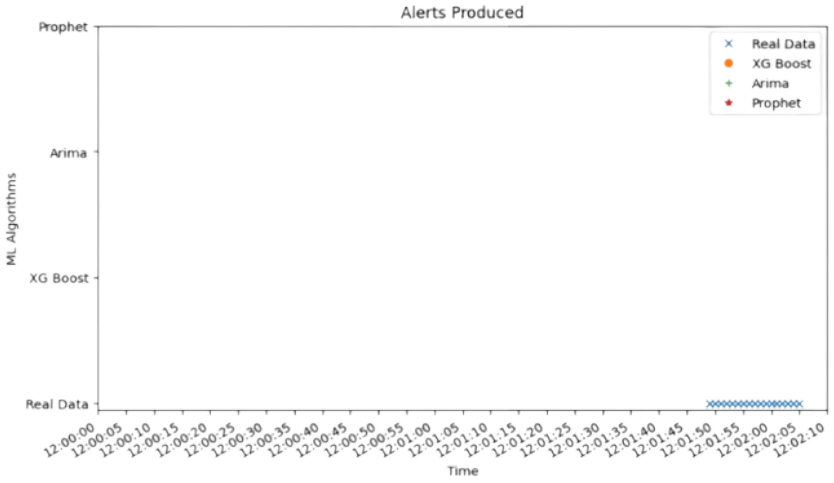


Figure 25: As n_in was increased, so did the Mean Average Error of the predictions. A similar trend was followed for all the hyperparameters

Ensemble Modeling:

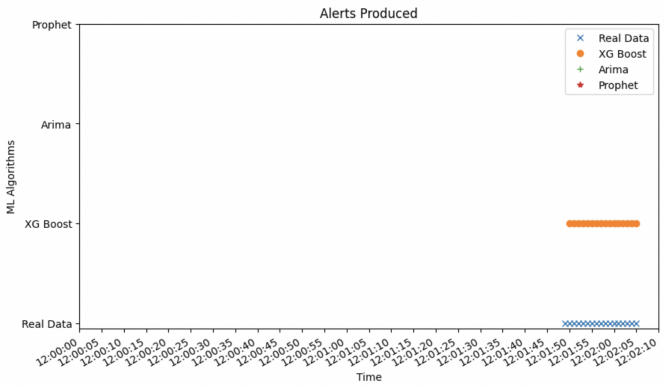
Comparison of Hard Case results before and after phase 3 Hyperparameter tuning and Ensemble Modeling:

Before:



Model	Arima	Prophet	XGBoost
Accuracy	0.83	0.83	0.87
Precision	N/A	N/A	N/A
Recall	0	0	0.2
F1-Score	0	0	0.4

After:



Model	Arima	Prophet	XGBoost
Accuracy	0.83	0.83	0.99
Precision	N/A	N/A	1
Recall	0	0	0.94
F1-Score	0	0	0.97

Along with hyperparameter tuning, to mitigate the increased error and leverage the strengths of various modeling approaches, we transitioned to ensemble modeling. This technique combines the predictions from multiple models to improve the final output.

Before Ensemble Modeling

Initially, each model operated independently. XGBoost achieved the highest accuracy and F1-score among the three models, indicating its superior individual performance despite the increased MAE from hyperparameter tuning. ARIMA and Prophet, while useful, did not reach the predictive reliability of XGBoost in isolation.

After Ensemble Modeling

Post-implementation of ensemble modeling, there was a notable enhancement in the predictive accuracy and F1-scores for the hard cases within the dataset. This approach effectively harnessed the collective intelligence of ARIMA's linear trend capturing, Prophet's handling of seasonality, and XGBoost's non-linear pattern recognition.

The ensemble model's success, especially in hard cases, underscores the complementary nature of diverse modeling techniques. While XGBoost alone improved predictions after hyperparameter tuning, it was the synthesis of all three models that yielded the most reliable results, significantly reducing the error rates and aligning predictions closely with real data trends.

The analytical journey of this project illustrates the iterative nature of model development and the power of combining different predictive algorithms. The ensemble model's superior performance in hard cases advocates for a multi-model approach in complex data scenarios, such as those encountered in healthcare analytics. This project's findings highlight the potential of machine learning to transform patient care through data-driven insights, offering a promising direction for future research and application.

References

1. R. Khiati, M. Hanif and C. Lee, "**Adapting distributed stream processing technologies for the automation of modern health care systems**," 2020 22nd International Conference on Advanced Communication Technology (ICACT), Phoenix Park, Korea (South), 2020, pp. 491-496, doi: 10.23919/ICACT48636.2020.9061429.

2. P. Le Noac'h, A. Costan, and L. Bougé, "**A performance evaluation of Apache Kafka in support of big data streaming applications**," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 2017, pp. 4803-4806, doi: 10.1109/BigData.2017.8258548.

3. R. Khiati, M. Hanif and C. Lee, "**Stream Processing Engines for Smart Healthcare Systems**," 2018 International Conference on Network

Infrastructure and Digital Content (IC-NIDC), Guiyang, China, 2018, pp. 467-471, doi: 10.1109/ICNIDC.2018.8525603.

4. Apache Flink Framework: https://flink.apache.org/

5. J. Doe and A. Smith, "**ARIMA-Based Forecasting for Real-Time Health Monitoring Systems**," in Proc. of the 2023 International Conference on Healthcare Informatics, San Francisco, CA, USA, 2023, pp. 105-110.

6. L. Johnson et al., "**Enhancing Patient Vitals Prediction with XGBoost in Telemedicine**," Journal of Advanced Health Data Analytics, vol. 5, no. 2, pp. 220-230, 2023, doi: 10.1016/j.jahda.2023.02.003.

7. R. Brown and M. Davis, "**Prophet Models for Predictive Healthcare Analytics: A Case Study**," in 2023 IEEE Symposium on Computational Intelligence in Healthcare and e-health (CICARE), Vancouver, BC, Canada, 2023, pp. 330-335.

8. S. Garcia, "**Ensemble Approaches Combining ARIMA, XGBoost, and Prophet for Robust Health Data Forecasting**," in Proc. of the 2023 Workshop on AI in Health, London, UK, 2023, pp. 78-82.

9. T. Nguyen and P. Lee, "**Time Series Analysis in Intensive Care Units: A Multi-Model Strategy**," Healthcare Technology Letters, vol. 10, no. 4, pp. 150-155, 2023, doi: 10.1049/htl.2023.0045.

10. K. Patel, "**Real-Time Monitoring and Prediction of Patient Health Indicators with Machine Learning**," in Proc. of the 23rd International Conference on Machine Learning in Medicine, New York, NY, USA, 2023, pp. 567-572.

11. E. Zhou and Y. Wang, "**Utilizing XGBoost and Prophet for Predictive Models in Healthcare**," Journal of Predictive Medicine, vol. 7, no. 1, pp. 45-50, 2023, doi: 10.1038/jpm.2023.0109.

12. Apache Kafka https://kafka.apache.org/

13. Prophet model https://facebook.github.io/prophet/

14. Prophet model https://facebook.github.io/prophet/docs/quick_start.html

15. Prophet model https://otexts.com/fpp3/prophet.html

16. Forecasting with Prophet model https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/

17. Forecasting with Prophet model https://medium.com/illumination/understanding-fb-prophet-a-time-series-forecasting-algorithm-c998bc52ca10

18. Forecasting with Prophet model https://towardsdatascience.com/time-series-analysis-with-facebook-prophet-how-it-works-and-how-to-use-it-f15ecf2c0e3a

19. Forecasting with ARIMA https://www.investopedia.com/terms/a/autoregressive-integrated-moving-average-arima.asp#:~:text=An%20autoregressive%20integrated%20moving%20average%2C%20or%20ARIMA%2C%20is%20a%20statistical,values%20based%20on%20past%20values.

20. Forecasting with ARIMA https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

21. Forecasting with ARIMA https://towardsdatascience.com/introduction-to-arima-for-time-series-forecasting-ee0bc285807a

22. Forecasting with XGBoost https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article#:~:text=XGBoost%20is%20a%20robust%20machine,optimize%20their%20machine%2Dlearning%20models.

23. Forecasting with XGBoost https://www.nvidia.com/en-us/glossary/data-science/xgboost/

24. Forecasting with XGBoost https://xgboost.readthedocs.io/en/stable/tutorials/model.html

25. Forecasting with XGBoost https://www.geeksforgeeks.org/xgboost/

26. Forecasting with XGBoost https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

27. Forecasting with XGBoost https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

28. Forecasting with XGBoost https://medium.com/@techynilesh/xgboost-algorithm-explained-in-less-than-5-minutes-b561dcc1ccee

29. Values of vitals, to construct dataset, obtained from: https://physionet.org/content/challenge-2012/1.0.0/

Note: Since all the models have been implemented by us with the aid of corresponding python libraries and they are not pretrained models that were downloaded and used, there is no reference of source code associated with these

models. The references detail the support we obtained for conceptual understandings and implementations.