Visualizing Medical Alarm History with Python

George Xie

Abstract

Medical alarm history is useful in hospital settings because it can be used to understand trends in patient health, allowing doctors to make informed decisions. Current systems for checking alarm history are inconvenient to use. Although visualizations do exist, they do not convey information very effectively. In response to this problem, I utilized Plotly, a Python library, to create a scatterplot that allows users to see at a glance the alarms that occurred during a certain time period, conveying information through size and color while allowing details to be viewable on hovering. In the future, I hope to improve and test this system, aiming to reduce the time spent on reviewing medical alarm data and improve patient outcomes.

Introduction

In hospitals, doctors and nurses often use alarm data to understand a patient's condition. While active alarms are useful for nurses to respond to ongoing situations, alarm data history is important for understanding longer-term trends.

However, current systems for checking alarm history are inconvenient to use. The database is in the form of a table; although the user is given summaries of the types, durations, and times that alarms occurred, there's no way to visualize patterns other than filtering and looking at the database, which makes it harder to interpret and understand data. In Figure 1, for example, the user may see a spike in alarm count just before the halfway point in the view range, but there's no way to tell what type of alarms there are. Similarly, the user can see at a glance the top alarm counts for each bed, but without directly searching the database, they cannot see when they occurred, or how long they lasted.

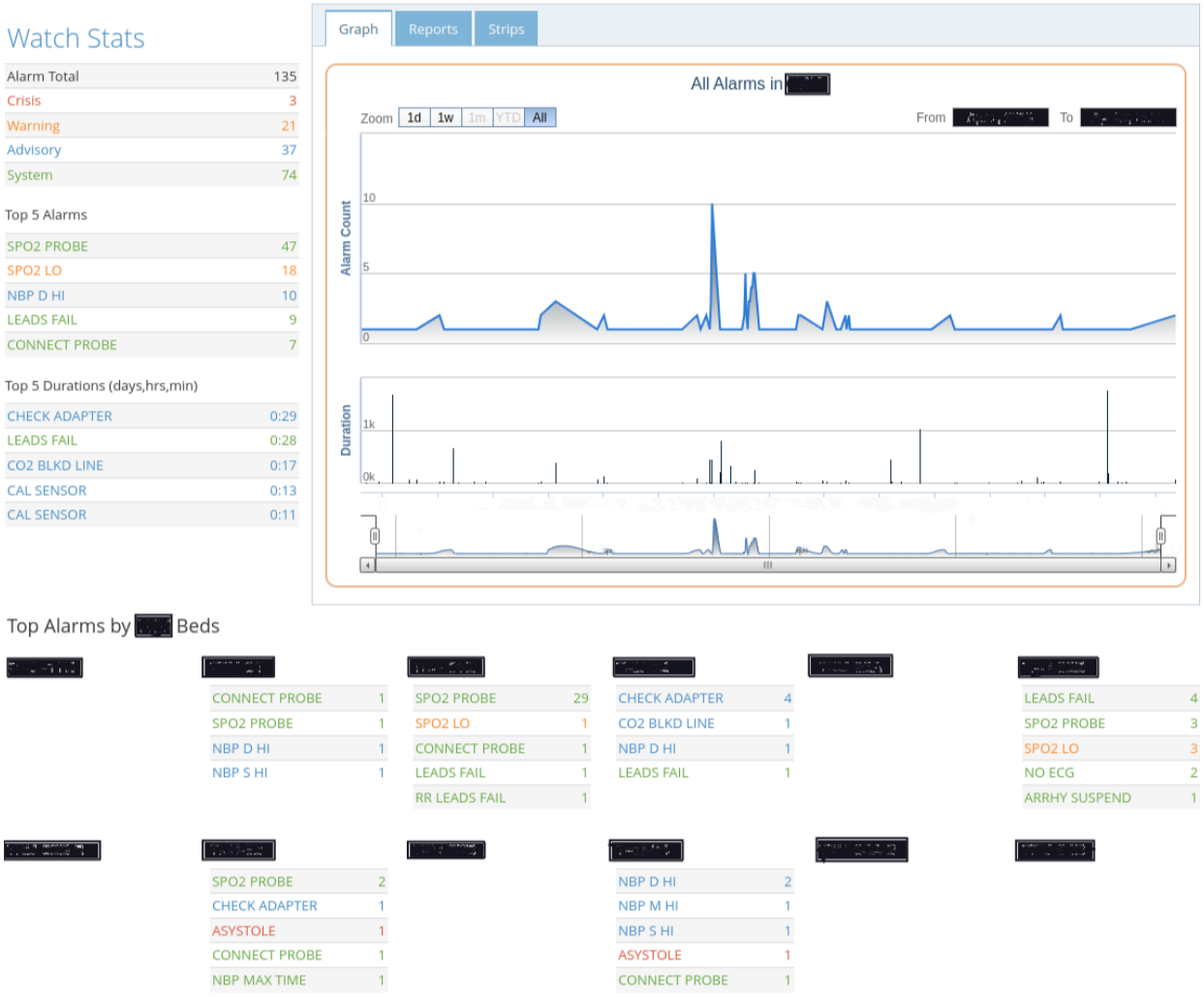


Figure 1: A screenshot of the current alarm database being used at the University of Maryland Medical Center.

In this paper, I created a visualization system for medical alarms, using the Plotly Python library, to improve the interpretability of alarm data. I aim to provide a more comprehensive overview that allows users to see patterns in the data at a glance.

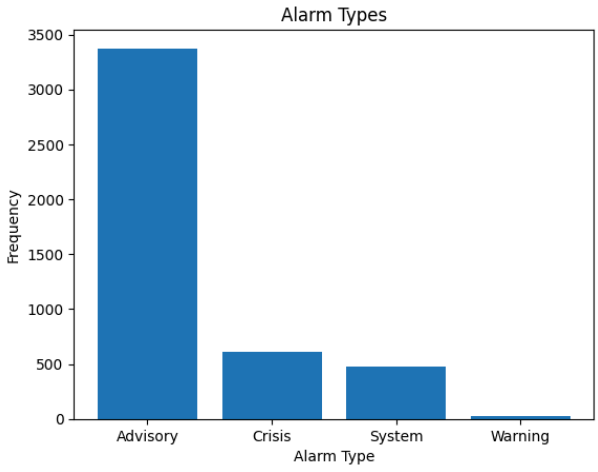
Methods

I was given a sample dataframe, in CSV format, consisting of 4484 alarms gathered from 12 beds at the University of Maryland Medical Center (UMMC) over a 72-hour period. (In compliance with HIPAA regulations surrounding the use of PHI, I have shifted the AlarmTime of the data by an arbitrary amount and applied a random shift to each data point. These changes do not detract from the ideas underlying the visualization.) It includes six features:

- AlarmTime: “YYYY-MM-DD HH:MM:SS” formatted string, the time that the alarm began.
- ID: A seven-digit integer that acts as an alarm identifier.
- BedUID: A three-digit integer ID of the bed where the alarm was triggered.
- AlarmLevel: A one-digit integer encoding one of four alarm types, assigned automatically by a computer. See Figure 2.
- AlarmMessage: A formatted string describing what triggered the alarm. (ex. “HR HI ###”)
- AlarmDuration: An integer representing the duration (in seconds) that the alarm was active.

Crisis	ASYSTOLE	4
	HR HI	68
	HR LO	33
	ICP1 M	116
	ICP2 M	184
	ICP4 M	14
	NO BREATH	39
	V TACH	15
System	ARRHY SUSPEND	147
	CONNECT PROBE	41
	LEADS FAIL	144
	NBP MAX	13
	NO ECG	26
	RR LEADS	28
	SENSOR	24
	SPO2 PROBE	186
	WRONG CABLE	1
Advisory	ART1 D	421
	ART1 M	211
	ART1 S	963
	ART2 D	46
	ART2 M	24
	ART2 S	69
	CAL CO2	5
	CAL SENSOR	1
	CHECK ADAPTER	575
	CO2 BLKD	2
	NBP D	47
	NBP M	7
	NBP S	87
	NO BREATH	101
	SPO2 HI	1
	SPO2 LO	814
Warning	ICP1 M	1
	ICP3 M	2
	NO BREATH	9
	SPO2 LO	16

Figure 2: A table of the first two words of the alarm messages and their respective counts for each alarm type.



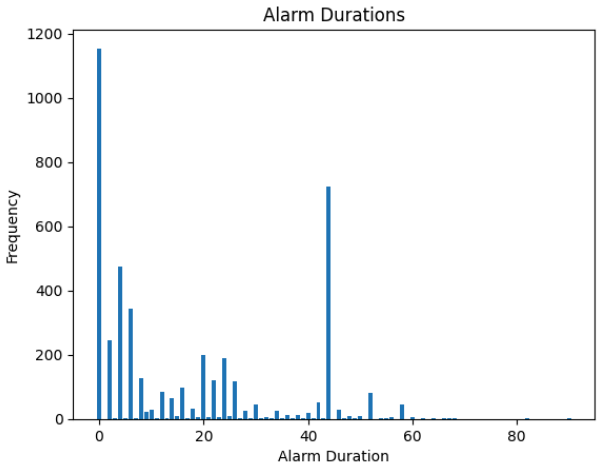


Figure 3 (left): A bar chart showing the distribution of alarm types.
Figure 4 (right): A histogram of all alarm durations.

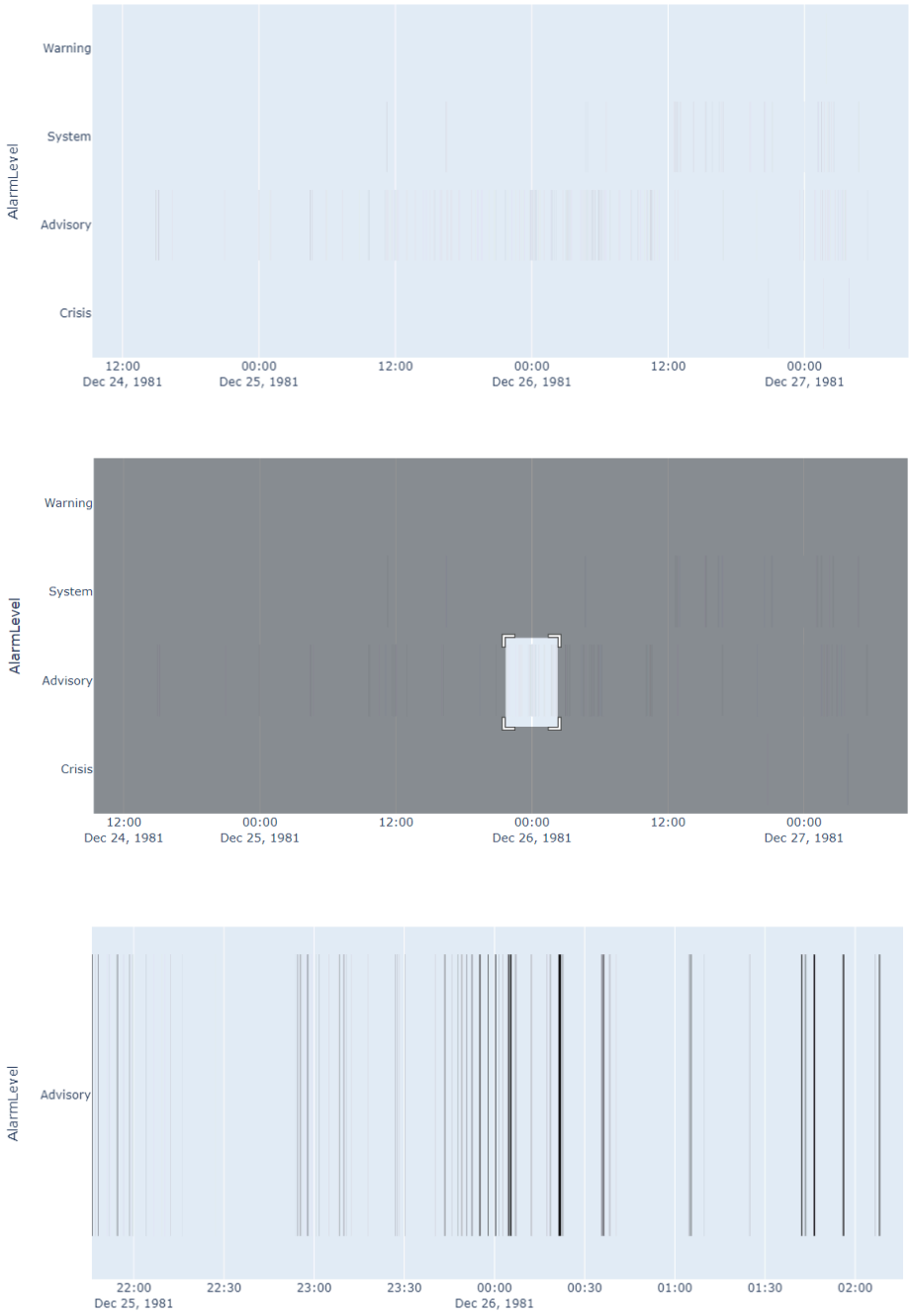
I first created some basic visualizations to get a sense of the data. The two aspects that stood out the most to me were the alarm types and durations: a majority of the alarms were Advisories, and a majority of the alarms were short, with very brief "instantaneous" ones marked as lasting zero seconds.

I understood that there were three big things to communicate: Time of alarm, duration of alarm, and some basic indication of the alarm type. This motivated my first idea.

Timeline

Making a timeline was an intuitive choice; considering the classic example of a schedule, it is immediately apparent when an event starts and how long it lasts. I split the data into the four different alarm types, and after some research, chose to use Plotly for its convenience. After isolating the data from just one bed, I created a timeline that simply had bars representing the start and end times (see Figures 5a, 5b, and 5c).

However, a drawback is immediately visible. Alarms are short — on the order of seconds, rarely lasting longer than a minute — making it difficult to see at a glance where any given alarm is without zooming in. Furthermore, some alarms have a length of zero, making them invisible. While I didn't want the short alarms to clutter the screen, I wanted them to at least be visible at a glance: even short alarms may be clinically significant. This motivated my next idea.



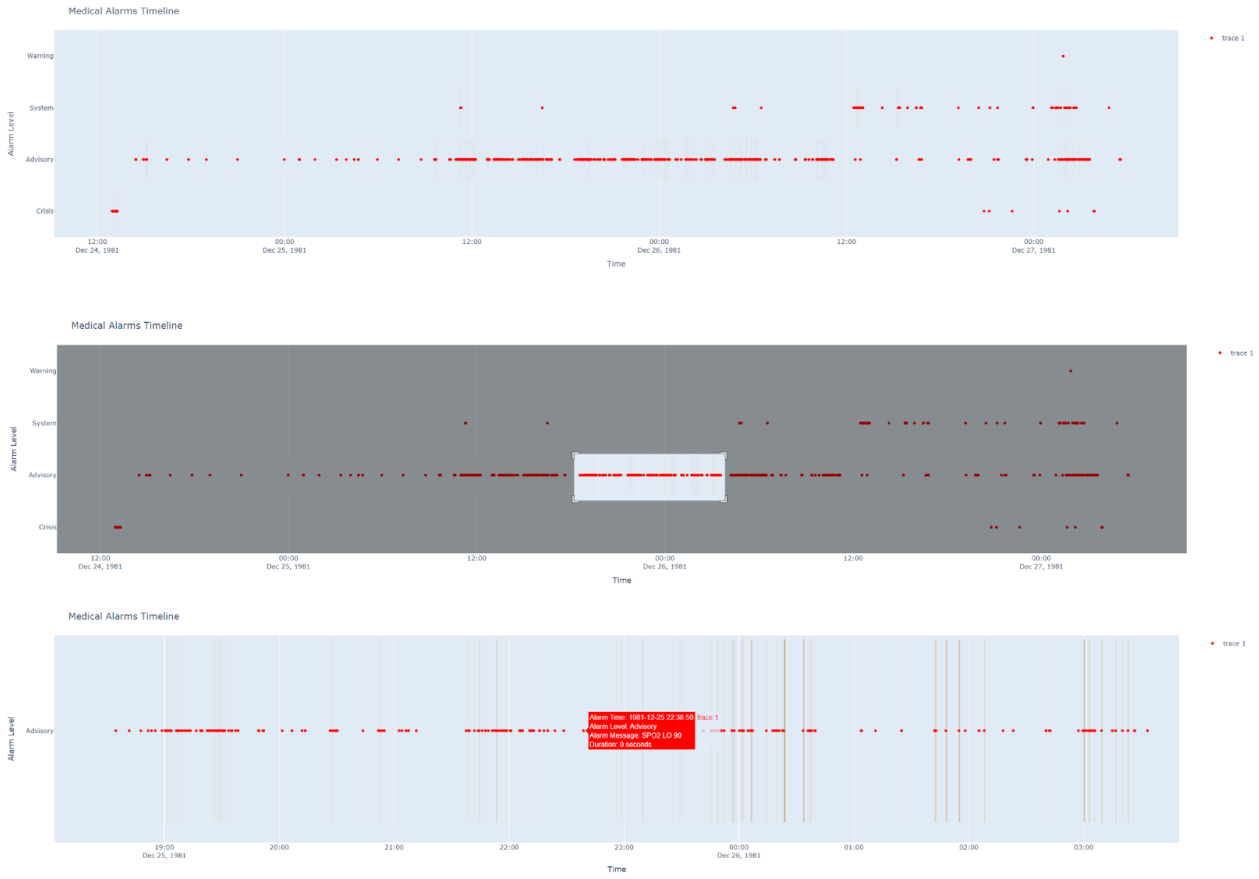
Figures 5a, 5b, 5c: Visualization when zoomed out (top), selecting an area to zoom in on (middle), and results (bottom).

Timeline with markers

At the beginning of each bar, I added a small red marker to indicate that there is an alarm there. My motivation for doing this was so that alarms could at minimum be visible at any zoom level.

While I considered this an improvement, I knew that the alarm type was important, accounting for the context in which I envision this system being used. I could think of two ways to do this; color-coding, and sorting the dots by alarm type instead of alarm level. The former was more compact — allowing the possibility of fitting more information onto a monitor with limited space — but with the tradeoff that it was more cluttered. The latter distinguished more easily between alarm types, making the information more visible at a glance, though this resulted in a lack of severity indicators.

After consideration, I opted to use color coding to represent alarm types. I valued conciseness, and color was more scalable; there were many more alarm types than alarm levels, and I felt that having a longer legend was more beneficial than having more bars. I reasoned that I could implement a filter system to mitigate the possibility of clutter.



Figures 6a, 6b, 6c: Visualization when zoomed out (top), selecting an area to zoom in on (middle), and results with hovertext (bottom).

Color-coded markers

I now faced the question of how to represent the alarm types. When I was given the dataset, the only indication of alarm type I had was the AlarmMessage column, which I couldn't immediately use; since alarm values varied, my code would recognize them as distinct and assign a color coding to each distinct value. In reality, though, this is not only unhelpful but detrimental, cluttering the screen without providing any real use of color coding.

Thus, I created a new column, AlarmMessageSimplified, with the first word of each AlarmMessage. On implementation, I found out that Plotly automatically created a filtering system, where one could click on a category in the legend to toggle; double click, and it would filter all but that category. However, while it worked as expected on the timeline, I found that it left the markers unfiltered. Reviewing my code, I hypothesized that since the markers weren't a part of the timeline itself, instead "layered on top", Plotly may not have considered it part of the figure for filtering purposes. Finally, I noticed how the actual timeline portion did not contribute as much to the visualization as I initially imagined; while it is a nice visual when visible, it was not helpful when zoomed out, only serving as a measure of alarm density. Thus, I decided to create a new system centered around the markers themselves.

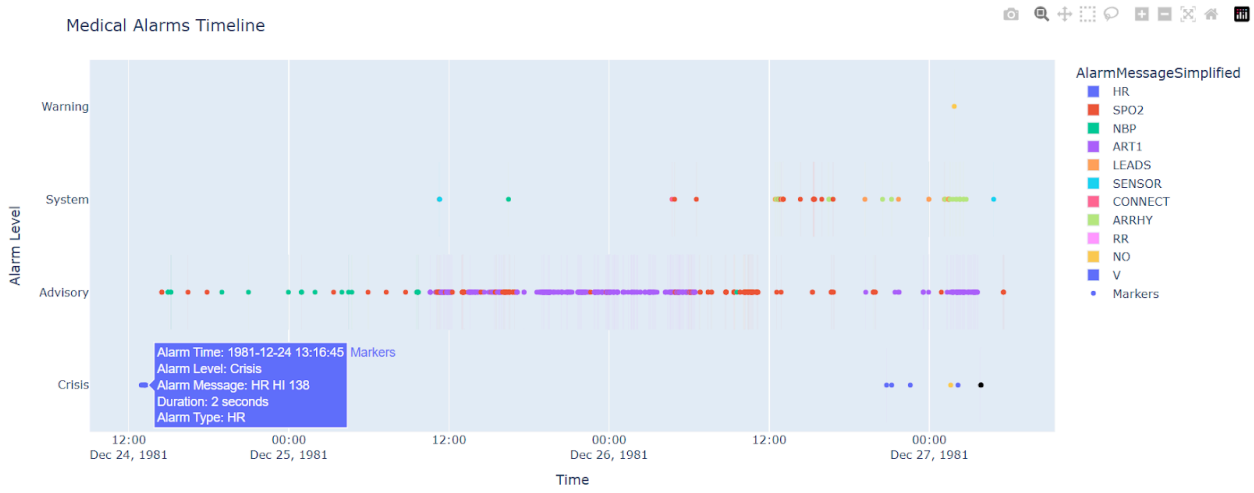


Fig 7: Visualization with color-coded markers.

Duration indicators with size

The implementation and the resulting interface were similar to those of the timeline. The drawback to this implementation was that I no longer had a way to see duration without hovering. Since I still wanted to display duration in a way that emphasized longer alarms, I decided to use dot size. However, I was unable to use AlarmDuration directly; alarms with length 0 would not appear, and the longer alarms were unnecessarily large. Thus, I created a new column, DotSize, to contain the dot sizes. I arbitrarily defined it as AlarmDuration/2 + 1; this guaranteed that all points were at least visible while limiting the size of the larger dots.

During discussions with other members of my lab, I was advised to include more information in AlarmMessageSimplified. It is unclear at a glance what some of them were meant to signify; "NO BREATH" is clearer than "NO", and "V TACH" is clearer than "V". So, I changed the AlarmMessageSimplified to use the first two words instead.

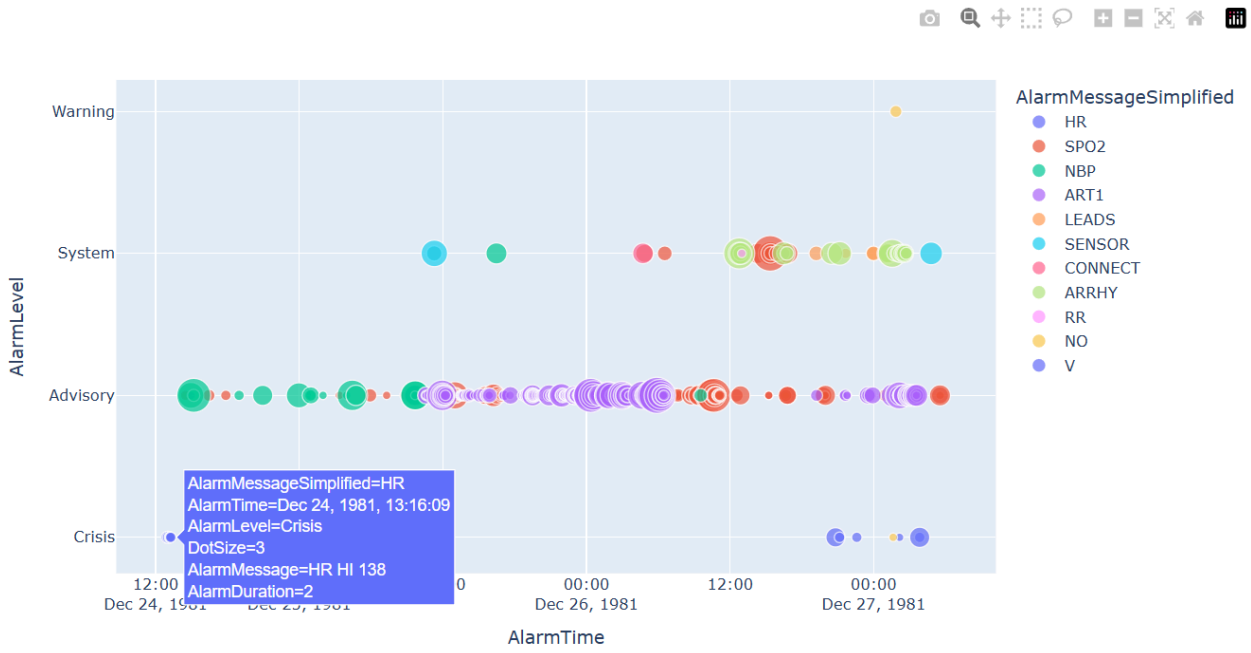


Fig. 8: Scatterplot visualization.

Result

The full implementation of my current working system is below.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go

#File was preloaded.
df = pd.read_csv("/content/alarmdb.csv", names = ["AlarmTime", "id", "BedUID", "AlarmLevel", "AlarmMessage", "AlarmDuration"])

#Alarm Levels are given as integers; this converts them to their corresponding strings.
alarmLevels = ["","","","System", "","Advisory","Warning","Crisis"]
newAlarmLevel = [alarmLevels[i] for i in df['AlarmLevel']]
df['AlarmLevel'] = newAlarmLevel

#Creating AlarmMessageSimplified. Note that the implementation here is for the two-word AlarmMessageSimplifed.
messages = df["AlarmMessage"].str.split().str[:2]
for i,message in enumerate(messages):
  stringMessage = ""
  #These two alarm types are only one word long.
  if message[0] == "ASYSTOLE" or message[0] == "SENSOR":
    stringMessage = message[0]
  else:
    stringMessage = message[0] + " " + message[1]
  messages[i] = stringMessage
df["AlarmMessageSimplified"] = messages

#Convert AlarmTime to datetime and calculate AlarmEnd.
df['AlarmTime'] = pd.to_datetime(df['AlarmTime'])
df['AlarmEnd'] = df['AlarmTime'] + pd.to_timedelta(df['AlarmDuration'], unit='s')

#Establishing DotSize
df['DotSize'] = df['AlarmDuration']/2+2

# For privacy purposes, I will shift all AlarmTime values back by an arbitrary amount of time,
# and scramble the timings by applying some shift to each AlarmTime.
# Note that these two lines would not be here in practice.

df['AlarmTime'] = df['AlarmTime'] - pd.DateOffset([redacted])
df['AlarmTime'] = df['AlarmTime'] + pd.to_timedelta([redacted])

#I will be using information from one specific bed.
dfBed265 = df[df.BedUID == 265]
```

```
fig = px.scatter(dfBed265, x="AlarmTime", y="AlarmLevel", color="AlarmMessageSimplified",
    size="DotSize", hover_data=["AlarmMessage","AlarmDuration"])

# My attempt at removing DotSize, AlarmMessageSimplified, and AlarmLevel -- redundant information -- from hoverdata.
# Not currently functional; may continue work on this as a future step.
# fig.update_traces(customdata=dfBed265[['AlarmTime','AlarmMessage', 'AlarmDuration']],
#           hovertemplate="<br>".join([
#         "Alarm Time: %{customdata[0]}",
#         "Alarm Message: %{customdata[1]}",
#         "Duration: %{customdata[2]} seconds",
#     ]))

fig.show()
```

Fig. 9a, 9b: My code for creating my current system.

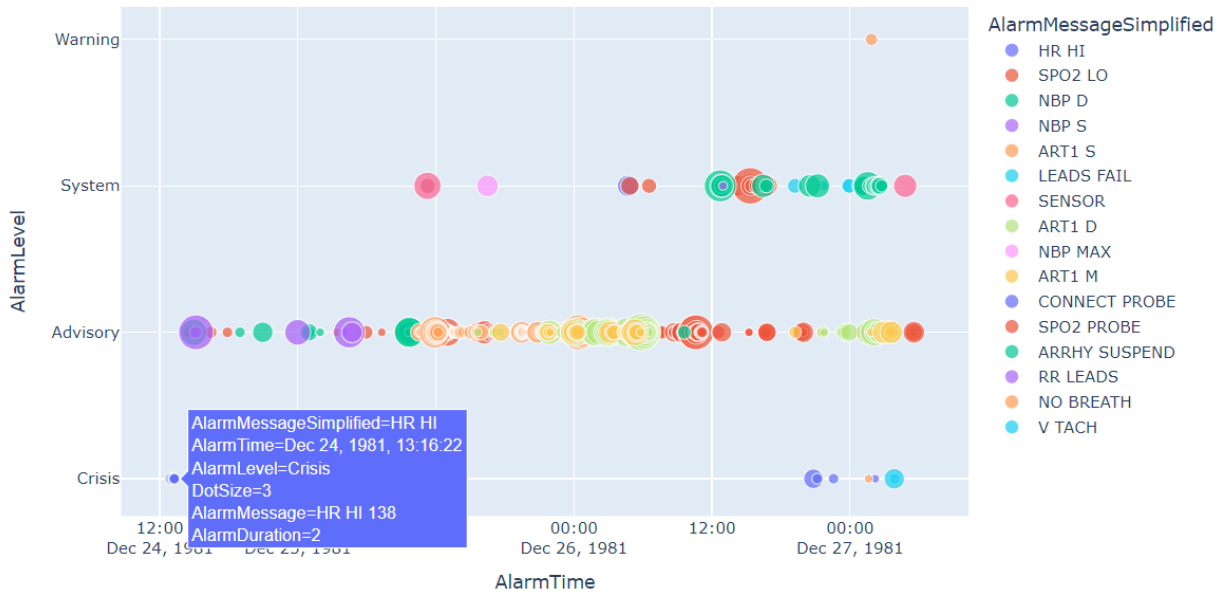


Fig. 10: My current system, compressed for viewing ease, with the cursor hovering over a sample alarm.

In summary:

- This system is a scatterplot with the alarm levels as categories on the vertical axis and time on the horizontal axis, where each point represents an alarm.
- For each point, the color represents the alarm type, and the size indicates duration.
- When hovering over a point, the visualization displays information about the alarm. Importantly, it shows the time, alarm message, and duration.
- The user can select an area on the visualization to zoom in. This allows for more details to be viewable.
- Clicking on an AlarmMessageSimplified option on the right filters it from the screen. Double-clicking filters all alarms except the selected one.

Future steps

The AlarmLevel labels were computer-generated, and I was not given any information on how the alarm system generated the labels for “Advisory”, “Warning”, and “Crisis”. In the future, I hope to investigate the current categorization and attempt to redesign it to improve patient outcomes.

Depending on usability, I may change the organization of my current system further; if space is not as big of an issue, for example, I may split the rows by AlarmMessageSimplified and use a simple color code or some other method to display AlarmLevel. At the moment, the values are not in order of severity; I hope to investigate how Plotly sorts the rows, as well as what other

methods there are to convey the severity of crisis alarms. In addition, I will create documentation that explains how the visualization works.

Note that with this system, there is repetition of color in the legend starting from CONNECT PROBE, and a strong similarity between the colors: SPO2, ART1 S, and ART1 M, for example, are all shades of orange. This reduces its usefulness as a color coding and makes it unlikely to be useful to colorblind users. This shows the tradeoffs between one-word and two-word AlarmMessageSimplified; in the future, I may further analyze this problem to decide between the two, find a compromise, or develop a new solution. I also hope to compare this system to the existing system through testing.

Acknowledgments

This research was done at the STAR-ORC lab at the University of Maryland Medical Center. I thank Dr. Peter Hu for his experience and support, and Mr. Bradford Burdette for his mentorship. I would like to thank Mr. Anthony Wang for providing me with data, and Dr. Megan Watkins for her review of my system. Finally, I would like to thank Dr. Fritz for his review of my paper, and Mr. Peter Ostrander for coordinating the Senior Research Project.