Response Time Analysis

Technical Report

Eshan Wickramarachchi

Technical Report **1**
Eshan Wickramarachchi **1**
Introduction 3
Overview 3
Data Sources and Loading 4
Data Cleaning 4
Handling Missing Data 4
Grouping Multiple Submissions 4
Merging Datasets 5
Calculating Response Times 5
Tracking Response Time Trends Over Time 6
Analyzing Team Performance 6
Visualizing Response Time Distribution 6
Day of the Week Analysis 7
Key Metrics Calculation 7
Conclusion 7

Introduction

This technical report provides a detailed explanation of the methods and processes used to analyze healthcare response times. It outlines how data was sourced, cleaned, and prepared for analysis, followed by the steps taken to merge datasets, calculate response times, and generate insights into team performance. Special attention is given to key decisions made during data processing, including handling missing data, grouping submissions, and calculating metrics. Each section of the report includes relevant code to ensure transparency and reproducibility.

The aim of this report is to provide a clear and structured guide to the entire process, from data preparation to the generation of performance insights. This report complements the response time analysis by giving a deeper look into the technical aspects of the workflow and the rationale behind each decision.

Overview

This report outlines the entire process of analyzing healthcare response times, from data acquisition and cleaning to calculating key metrics and generating insights into team performance. Each decision made during the process is explained, and the corresponding code is provided for transparency.

Data Sources and Loading

The project uses two datasets:

- **Patient Entries**: This dataset contains submission details such as timestamps (`createdAt_time`) and patient identifiers (`patientId`), which track when patients submit requests.
- **Audit Actions**: This dataset logs the responses from healthcare teams, recording when they responded (`timeResponded`) and which team handled the submission.

The first step was to load both datasets into Python using pandas.

```
patient_entries_df =
pd.read_excel(r'C:\Users\eshan\Documents\Serious\Patient_Entries.xlsx')
audit_actions_df =
pd.read_excel(r'C:\Users\eshan\Documents\Serious\Audit_Actions.xlsx')
```

Data Cleaning

Handling Missing Data

To ensure I had complete data for calculating response times, I needed to address missing values. In particular, any rows missing `createdAt_time` (when the submission was made) or `timeResponded` (when the team responded) were removed. Without these fields, I wouldn't be able to accurately calculate response times, so this was a necessary step to avoid distortions in the analysis.

```
patient_entries_df = patient_entries_df.dropna(subset=['createdAt_time'])
audit_actions_df = audit_actions_df.dropna(subset=['patientId',
'timeResponded'])
```

Grouping Multiple Submissions

Some patients may submit multiple entries (like photos or documents) within a short period for the same issue. Treating each of these entries as a separate submission would artificially increase the number of submissions and potentially reduce calculated response times.

To address this, I grouped submissions made within a **3-minute window**. This threshold was chosen after testing various time frames and observing that, beyond 3 minutes, it made little difference to the overall averages. Shorter thresholds risked overdefineing distinct submissions, while longer ones didn't provide additional benefit.

```
patient_entries_df = patient_entries_df.sort_values(by=['patientId',
'createdAt_time'])
```

```
patient_entries_df['time_diff'] =
patient_entries_df.groupby('patientId')['createdAt_time'].diff()

threshold = pd.Timedelta(minutes=3)
cleaned_patient_entries = patient_entries_df[
    (patient_entries_df['time_diff'].isna()) |
(patient_entries_df['time_diff'] > threshold)
].copy()

cleaned_patient_entries = cleaned_patient_entries.drop(columns=['time_diff'])
```

Merging Datasets

After cleaning and grouping the patient entries, the next step was to merge this data with the audit actions to link each submission to its corresponding response. I merged the datasets using the `patientId` column, ensuring that each submission was paired with the correct team response.

However, I also had to ensure that responses were valid—i.e., only keeping responses that were recorded **after** the submission. This step was necessary to eliminate data entry errors where responses were logged with incorrect timestamps.

```
merged_df = pd.merge(cleaned_patient_entries, audit_actions_df,
on='patientId', how='left')
merged_df = merged_df[merged_df['timeResponded'] >=
merged_df['createdAt_time']]
```

Calculating Response Times

With the data merged, I calculated the response time for each valid entry. This was done by subtracting the submission time from the response time and expressing the result in days. Focusing on the **first response** for each submission is important because it reflects how quickly the team addressed the patient's initial concern.

In cases where multiple responses were logged for the same submission, I kept only the closest response—the one with the shortest response time.

```
merged_df['Response_time'] = (merged_df['timeResponded'] -
merged_df['createdAt_time']) / pd.Timedelta(days=1)

merged_df = merged_df.sort_values(by=['patientId', 'entryId',
'Response_time'])
closest_responses = merged_df.drop_duplicates(subset=['entryId'],
keep='first')
```

Tracking Response Time Trends Over Time

To track improvements or declines in response time performance, I plotted average response times by month. This trend analysis allows us to see if teams have been getting faster or slower over time and identify any periods of improvement or decline. This also gets a good initial macro overview of the data which allows for a deeper dive in possible areas brought to light through it.

```
response_df['year_month'] = response_df['createdAt_time'].dt.to_period('M')
response_time_by_month =
response_df.groupby('year_month')['Response_time'].mean().reset_index()
```

Analyzing Team Performance

I analyzed team performance by grouping the data by team name and calculating the average response time for each team. This comparison helps highlight which teams respond faster and which teams may need to improve.

Additionally, I filtered the data to submissions made after June 2013 to assess whether teams had improved their response times over time.

```
bardf = closest_responses.groupby('team
name')['Response_time'].mean().reset_index()

filtered_responses = closest_responses[closest_responses['createdAt_time'] >=
'2013-06-01']
bardf_filtered = filtered_responses.groupby('team
name')['Response_time'].mean().reset_index()
```

Visualizing Response Time Distribution

Since response times are typically skewed (with many quick responses and a few that take much longer), a logarithmic scale was used in the histogram. This helps visualize both the shorter, more common response times, as well as the longer delays, providing a clearer picture of the full range of response times.

```
plt.figure(2, figsize=(10, 6))
sns.histplot(closest_responses['Response_time'], bins=1000, kde=False,
color='orange')
plt.xscale('log')  # Logarithmic scale for skewed data
```

Day of the Week Analysis

I also wanted to see if response times varied depending on the day of the week, particularly over weekends when healthcare teams may be understaffed. By grouping the data by the day the submission was made, I could identify patterns in response time performance across different days.

```
response_df['day_of_week'] = response_df['createdAt_time'].dt.day_name()
response_time_by_day =
response_df.groupby('day_of_week')['Response_time'].mean().reset_index()
```

Key Metrics Calculation

To summarize the overall system performance, I calculated several key metrics: the average, median, minimum, and maximum response times. I also flagged any responses that took longer than 30 days as outliers, as these unusually long delays typically indicate system issues or exceptional cases that require further investigation.

```
average_response_time = closest_responses['Response_time'].mean()
median_response_time = closest_responses['Response_time'].median()
max_response_time = closest_responses['Response_time'].max()
min_response_time = closest_responses['Response_time'].min()

outliers = closest_responses[closest_responses['Response_time'] > 30]
outlier_count = len(outliers)
```

Conclusion

This report outlines the entire process used to analyze healthcare response times, from data loading and cleaning to merging, calculating response times, and visualizing the results. Each step—whether grouping submissions, filtering invalid data, or focusing on the first response—was designed to ensure that the analysis accurately reflects how quickly healthcare teams respond to patient needs. By incorporating key metrics and tracking performance trends over time, the analysis provides a clear and actionable view of team performance, highlighting areas for improvement.

The code included in each section shows how these processes were implemented, making it easy to follow the workflow and understand the rationale behind each decision.