Manoj Sharma

Streamlit Dashboard Development Report

Abstract:

I started learning the Streamlit library from the official guide and some of the initial beginner tutorials on YouTube. I initially played around with small test cases like displaying basic text, file upload, and plotting some sample data. After I was comfortable with the functions and the structure, I started using Streamlit with my own data. Most of the learning was trial-and-error. I would try something, and once it did not work, I would look for it or run little parts of the code to understand where it was incorrect. As time passed, I realized how to use things like st.columns(), st.metric(), option_menu, and st.sidebar() to produce a more structured dashboard with multiple pages.

The most frustrating part was debugging the logic when the filters weren't filtering appropriately. In the Demographics page, for example, I had to enable the users to filter the data based on gender, race, state, and assistance type. I initially just placed plain dropdowns, but they weren't filtering the data as I would have wanted them to. I had to alter the logic and try different ways of applying conditions within Pandas. I also tested quite a lot with small datasets to make sure filters were acting as they ought to. I finally realized that with check boxes for each category and check filtering, it was making users more flexible and was not confusing them. It was time and the effort of more than once to have arrived at that solution. I debugged for hours over small issues—most of the time simply because I had the column name or a typo. There were times when I changed the logic several times just to have the right summary figures. I also learned how to use st.columns() in order to make the appearance better and st.metric() in order to show significant figures like average support amount and days to pay. These small adjustments made the dashboard understandable and easy to navigate.

Later on, I tried to make the dashboard update automatically when the data file was changed. I noticed that after updating the data file on GitHub, the website version of the dashboard didn't reflect the changes. I thought maybe the trigger wasn't working, but then I realized it was because of caching. Streamlit was still using the old data from cache. I cleared the cache manually from the Streamlit Cloud dashboard and the new data appeared correctly. This taught me how important caching is when working with live data, so I removed the @st.cache_data from my code and I worked perfectly. I also tested the updates by changing small values in the data and refreshing the app to make sure the changes were showing up.

If I had to do the project again, I would definitely organize my code better. In this project I kept all my code in a single file, app.py, So, it got messy quickly. If I had separated each page into its own Python file and created a dedicated script for data cleaning, it would have been easier to manage and update. That way, if I changed something in the cleaning function, it would automatically apply to all pages. This would save time and make the app easier to understand and expand in the future.

I developed the dashboard in dark mode and styled all charts and visuals accordingly. However, Streamlit follows the user's system or browser theme, so the sidebar and interface may appear in light mode on other devices. Currently, there's no way to force dark mode for all users.

To maintain consistency, I used dark backgrounds and light fonts in the plots so the design still looks clean in both themes.

I would also include more exploratory analysis early in the process. When I started, I jumped into building pages without fully exploring the columns and their quality. In a new version, I would spend more time profiling the data—checking for missing values, standardizing categories, and understanding data distribution. That would have saved time during the plotting and debugging stages.

Another thing I would improve is how I handled filter logic and user interactions. Initially, I added basic dropdown filters, but as I realized users needed more control, I switched to using multiselects and checkboxes and later I again switch to filters, which gave better flexibility. Next time, I would plan these filters earlier and test different layouts to find the most intuitive user experience.

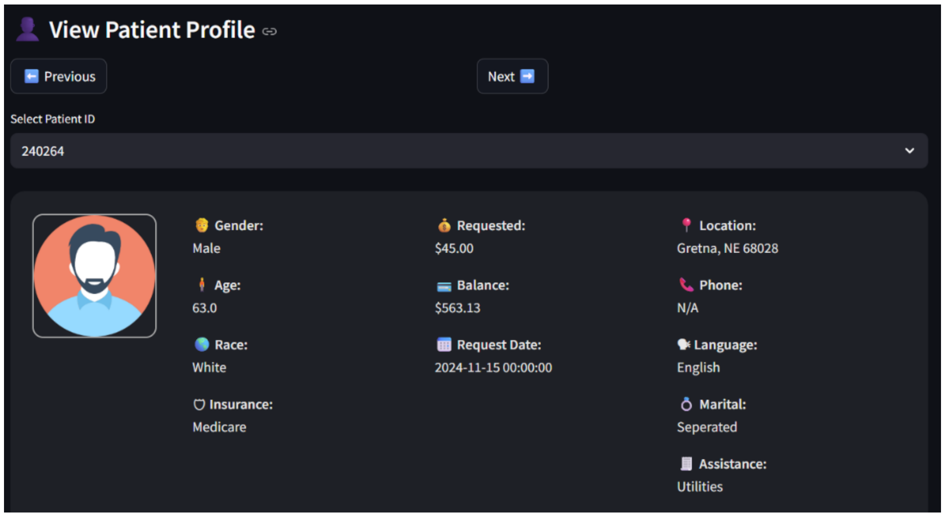
Overall, even though I didn't start with strong programming skills, I was able to build a working Streamlit dashboard by learning step by step, debugging patiently, and testing each part thoroughly. This project gave me a lot of confidence and practical experience in handling real-world data and turning it into something useful.

Additional Features and Functionalities of Some Pages:

Applications Ready for Review:

On the "Applications Ready for Review" page, it was initially confusing to determine which applications were truly ready for review. To make this clearer and more flexible, I added a filter for "Request Status" so users can select a specific status and view all patients whose applications are signed and match that status. This helps users narrow down the list based on the current stage of each request.

In addition, I implemented a "View Patient Profile" section to provide a more detailed view of each patient (Figure 1). This includes key information like age, race, insurance type, requested amount, contact details, and type of assistance. I also added a download button so users can export the filtered list for external use or reporting purposes. These features improve both usability and data accessibility for reviewers.



View Patient Profile

Select Patient ID

240264

Gender:
Male

Age:
63.0

Race:
White

Insurance:
Medicare

Requested:
$45.00

Balance:
$563.13

Request Date:
2024-11-15 00:00:00

Location:
Gretna, NE 68028

Phone:
N/A

Language:
English

Marital:
Seperated

Assistance:
Utilities

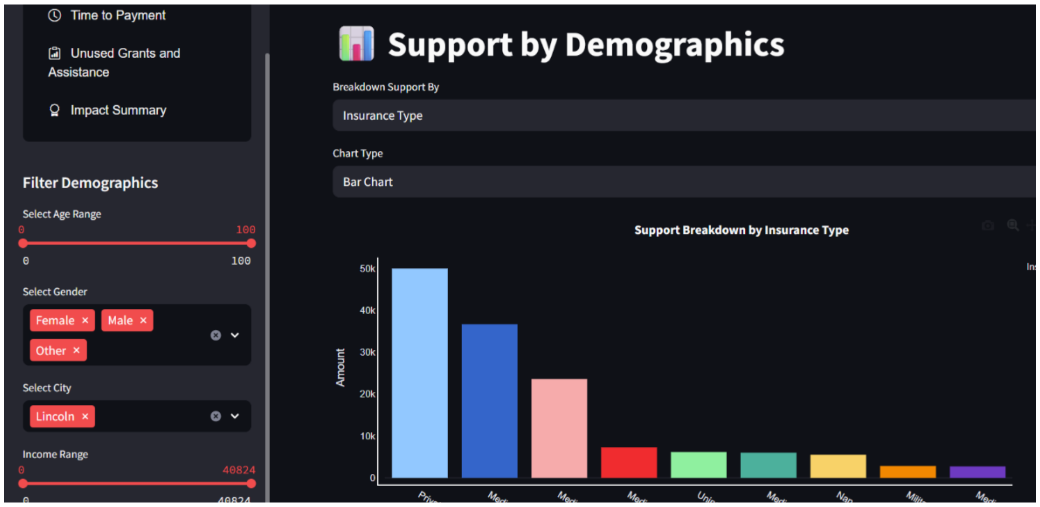
Figure1. Patients Profile

Support by Demographics:

On the "Support by Demographics" page, I implemented a Breakdown Support By feature that allows users to view total support distributed across categories such as gender, race, insurance type, marital status, and city. To enhance the analysis, I also added interactive filters for city, age range, gender, and household income.

For example, if someone wants to see how support is distributed specifically within Lincoln, they can simply select Lincoln from the city filter, and the breakdown charts will update accordingly to reflect only data from that city. Similarly, users can adjust the age and income sliders to focus on specific groups. (Figure 2)

By default, all options are selected, providing an overall summary. However, users can customize the filters to perform more focused and detailed analysis based on their needs.



Time to Payment
Unused Grants and Assistance
Impact Summary

Filter Demographics

Select Age Range: 0 – 100

Select Gender: Female, Male, Other

Select City: Lincoln

Income Range: 0 – 40824

Support by Demographics

Breakdown Support By: Insurance Type

Chart Type: Bar Chart

Support Breakdown by Insurance Type
Amount
50k
40k
30k
20k
10k
0

Figure 2: Filters in Demographics

Time from Request to Payment

The "Time from Request to Payment" (i.e., Days_to_Payment) is calculated as the difference between two dates: Grant_Req_Date, which indicates when the support request was made, and Payment_Submitted?, which records when the payment was submitted. This metric helps evaluate how long it takes for approved applications to receive financial support.

In addition to calculating the average and median values, I extended the analysis by comparing this time across different demographic groups such as race, insurance type, and city. (Figure 3). The results showed that the time to payment does vary among these groups, highlighting potential disparities or inefficiencies in the processing timeline that could be addressed to ensure more equitable support delivery.

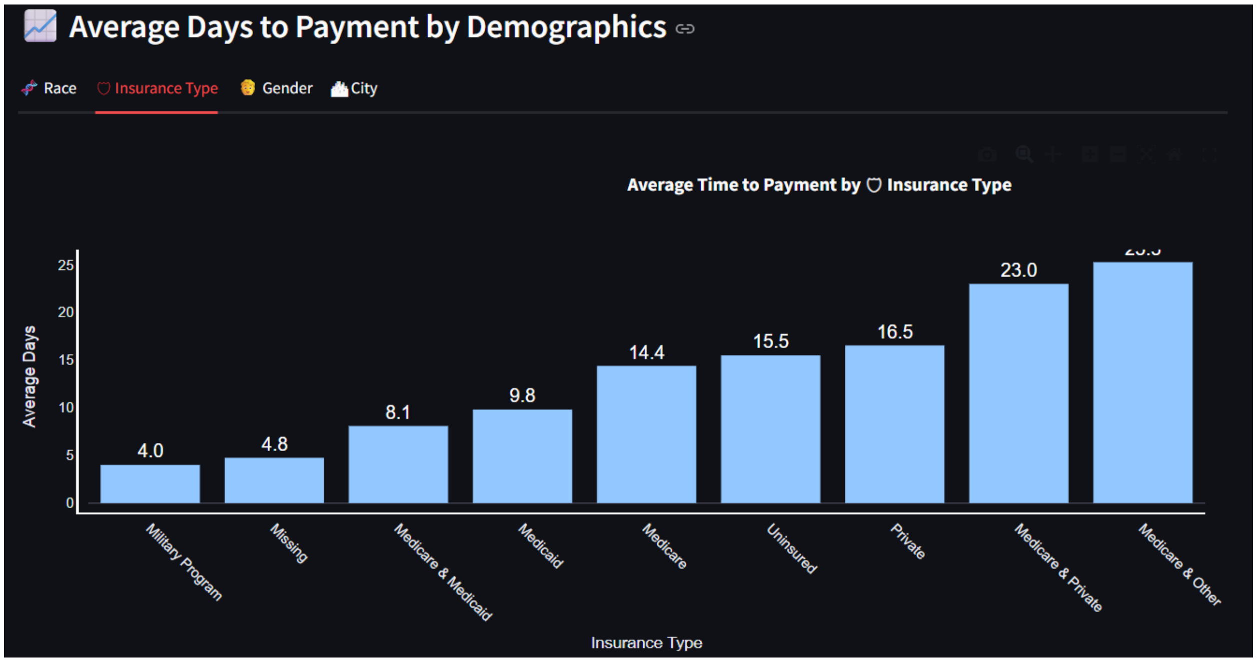


Figure 3: Average days of payment by different demographics