**Building a Password Strength Analyzer**

CYB333 Security Automations

National University

Deven Walker

Aaron Jaeger

June 28, 2025

**Building a Password Strength Analyzer**

For my CYB 333 final project, I developed a Password Strength Analyzer, designed to evaluate the strength of user-entered passwords based on length and character complexity. The goal of this project was to create a functional security automation script that promotes better password practices while strengthening my Python skills. The script checks for the presence of uppercase and lowercase letters, numbers, special characters, and overall length, then gives the user a score out of 6 along with suggestions for improvement. It runs entirely in the terminal and requires no external libraries, making it lightweight and simple to use.

I began by creating a GitHub repository and cloning it into my local CYB333/Final Project folder using PowerShell. From there, I set up my folder structure with dedicated subfolders for AI-generated content and screenshots and created the initial Python script. I originally named the script password\_analyzer.py but later renamed it to analyzer\_code.py to better reflect my naming preferences and avoid confusion. Early on, I made the simple but annoying mistake of saving the file with the wrong extension (.oy instead of .py). It took me a while to figure out why the script wouldn’t run, and once I realized it, I had to rename the file to fix the issue. This was a small detail, but it completely broke the program until it was corrected.

Throughout development, I made several similar small but disruptive mistakes. For example, I tried to run the script using a cd command in PowerShell to access a folder I was already in, which caused “PathNotFound” errors. This happened more than once and left me confused until I realized I didn’t need to move directories. I also forgot to add files before committing to Git, which meant nothing happened when I tried to push changes to GitHub. Learning the correct order of git add ., git commit -m, and git push took trial and error, but eventually clicked. One of the most frustrating moments was when I accidentally deleted the entire project folder from VS Code. I thought I had lost everything but then remembered I could just re-clone it from GitHub, which thankfully restored all my files.

To break the project into manageable chunks, I created a simple development timeline. During the first phase, I focused on setting up GitHub and the local environment. The second phase involved writing the initial structure of the script and adding basic logic like password length checks. In the third phase, I layered in character-type detection using regular expressions. The fourth phase focused on debugging, user input/output testing, and cleanup. The final phase was dedicated to documentation, screenshots, and report writing. This helped me avoid rushing and ensured each part was completed thoroughly.

Writing the actual code was more difficult than I expected. One major area of struggle was using regular expressions correctly. Even when I thought I wrote the right pattern to check for something like uppercase letters, the function wouldn’t always behave the way I expected. I also struggled with spacing and indentation errors in Python, especially when copying code from ChatGPT into VS Code. Sometimes the structure would look fine, but Python would throw an error because an if block wasn’t properly aligned or a colon was missing. These kinds of tiny syntax issues caused big frustration because they weren’t always easy to spot.

Another problem I had was making sure the scoring system was accurate and consistent. At first, I wasn’t sure how to divide the score or decide which parts of the password should contribute more. I had to rework my logic multiple times, especially when I noticed that strong passwords were getting low scores because one check was failing. I added print statements during testing to debug the values being calculated. Eventually, I set the score out of 6 and tied each character type to a point, with length being weighted slightly higher. I also had to make sure that when a condition wasn’t met, a clear suggestion was printed. Sometimes I’d forget to append a message to the list, and nothing would show up even though the password was weak.

Debugging this was challenging because there were no errors—just incorrect behavior. I learned to test one piece of logic at a time. I also learned that even just forgetting a return or misplacing a print statement could lead to confusing output. I had to run the script repeatedly with different passwords, writing out expected vs. actual outcomes until everything was behaving correctly.

I documented my development process through screenshots. I took screenshots of ChatGPT helping me troubleshoot errors, create regex patterns, and understand Python syntax. I also captured the script running in the terminal with both strong and weak passwords. These screenshots are stored in their respective folders in the project directory. The use of AI tools was extremely helpful — not as a shortcut, but as a support system that walked me through things I didn’t understand. It made a huge difference in how smoothly I was able to work through obstacles, especially when dealing with logic errors or syntax details I would’ve otherwise missed.

This project taught me how important small details are in coding and version control. One typo or misnamed file can break everything and even understanding something as simple as which folder I’m in became a big deal when using the terminal. It also reinforced the value of taking things step-by-step and not rushing. When I slowed down and tested my script regularly, I caught problems early and avoided compounding mistakes. I became more comfortable using GitHub, navigating folders, and managing my files through VS Code. I also learned how to write better commit messages, manage file changes, and keep my local and remote repos in sync.

Overall, I spent about 15 hours completing this project. I broke the work into focused chunks: initial setup, coding, testing and debugging, documenting, and writing the final report. Each phase had its own challenges but breaking it up helped me stay organized. I now feel more confident using Python to create basic automation tools, and I understand how AI can be used as a real partner in solving programming problems. I also feel better equipped to troubleshoot my own mistakes and stay calm when things go wrong.

Debugging the script wasn’t just about fixing errors — it was also about learning how to spot subtle mistakes that didn’t throw any error messages but caused incorrect behavior. For example, at one point, my scoring system was giving perfect scores to passwords that lacked special characters. I couldn’t understand why until I realized I had used the wrong regex character class. That one line caused the entire evaluation to be inaccurate. These kinds of logic bugs required me to slow down, walk through each section of code, and test line-by-line using print statements. I learned the value of isolating small blocks of logic, testing with known inputs, and not assuming a script works just because it runs without errors.

AI assistance played a much bigger role in this project than I expected. I used ChatGPT almost like a tutor — not to give me answers, but to explain things I didn’t fully understand. For example, I asked about why a certain regex wasn’t matching and learned that the placement of certain symbols in a character class actually affects its meaning. I also asked for help formatting my output and learned how to use conditionals more effectively in Python. When I ran into frustration, AI helped keep me moving forward instead of stuck for hours. I took screenshots of all the times AI helped me make progress, and those conversations became some of the most important learning moments of the entire project.

In the end, I learned just how fragile a script can be when small details are off and how important it is to test everything carefully. From file extensions to indentations to logic flow, the success of the tool relied on understanding and fixing minor mistakes. This project helped me build better habits as a programmer, especially when it comes to testing, organizing code, and thinking through problems in a structured way.