**Bitcot Technologies**

**Weekly Report – 1 (April 18 – April 25 2025)**

**Overview**

This report documents my activities and accomplishments during my initial week at Bitcot Technologies, from April 18 to April 25, 2025. The primary focus was on establishing my development environment, mastering Git fundamentals, and initiating development on Mini-Project #1: Markdown File Analyzer, hosted at https://github.com/tatedmaahi/miniprj1.git. This Python-based application analyzes .md files by counting words, identifying headings, validating hyperlinks, and detecting images, utilizing libraries such as markdown, requests, beautifulsoup4, matplotlib, seaborn, and plotly. My objectives included configuring essential tools, learning Git workflows, developing the core analyzer, enhancing it with a configuration system and visual reporting capabilities, and submitting my first pull request, all in compliance with the company’s coding and documentation standards.

**Accomplishments**

1. **Environment Setup (Day 1–2)**:
   * Successfully installed critical development tools, including VS Code, Git, and Python, to establish a functional workspace.
   * Created a GitHub account and configured SSH keys to ensure secure repository access.
   * Reviewed and adhered to the company’s coding standards and documentation guidelines, focusing on comprehensive commenting and clarity.
   * Completed a foundational Git tutorial using the Git Handbook, gaining proficiency in committing, pushing, and pulling operations.
2. **Git Workflow, Initial Repository Setup, and First Pull Request (Day 2&3)**:
   * Acquired knowledge of the company’s branching strategy (Git Flow: main, develop, feature branches) and implemented it by creating the feature/configure branch.
   * Established the initial repository (miniprj1) with a detailed README.md file, outlining project objectives, installation instructions, and usage guidelines, guided by the Markdown Guide.
   * Developed the initial version of analyzer.py, which performs word and heading counts using regex (e.g., ^#{1,6}\s), parses links and images with BeautifulSoup, validates hyperlinks via HTTP requests, and generates a console-based summary report, tested with example.md.
   * Submitted the first pull request from feature/configure to main, committing initial modifications to analyzer.py and configuring .gitignore to exclude files such as venv/, \*.log, and generated outputs (chart.png, report.html), adhering to the established code review process.
3. **Configuration System Integration (Day 5):**
   * Enhanced analyzer.py by integrating a config.json file to support customizable settings for link validation (e.g., timeout, user agent), analysis options (e.g., inclusion of images, headings, links), and visual report styling (e.g., chart dimensions, colors).
   * Implemented a load\_config function to generate or load config.json with default settings, ensuring flexibility and handling JSON decode errors with fallback defaults.
   * Modified analyze\_markdown to incorporate configurable parameters such as min\_word\_length and conditional analysis (e.g., disabling link validation), aligning with the project’s core functionality.
4. **Visual Report Implementation (Day 4&5):**
   * Developed a generate\_html\_report function in analyzer.py to produce a visual report as an HTML file (report.html), featuring a bar chart visualizing word, heading, link, and image counts using matplotlib and seaborn.
   * Ensured the visual report’s styling options (e.g., font family, colors, chart dimensions) are configurable via config.json. The bar chart is saved as chart.png and embedded in the HTML report, providing a clear and interactive summary of analysis results.
   * Explored an alternative approach with report.py, utilizing pandas and plotly to generate a dynamic bar chart in visrep.html, enhancing the tool’s usability for end users.
5. **Bug Resolution (Day 5):**
   * Identified and resolved an UnboundLocalError in the generate\_html\_report function of analyzer.py, caused by an erroneous assignment (html\_content = f). Corrected the issue by restoring the proper f-string formatting.
   * Conducted thorough testing with example.md, verifying accurate console output and successful generation of the visual report, including the embedded bar chart in report.html.
6. **Dependencies and Documentation (Day 5)**:
   * Updated requirements.txt to include markdown, requests, beautifulsoup4, matplotlib, and seaborn to support the analyzer and visual report functionality.
   * Added detailed comments to analyzer.py and report.py in accordance with company standards, including docstrings for functions such as load\_config, analyze\_markdown, generate\_html\_report, and make\_visual\_report.
   * Enhanced README.md with comprehensive documentation on tool functionality, installation procedures, and usage instructions, including details on the visual report feature, referencing the Python Markdown Library.

**Challenges**

* **Git Learning Curve**: Initially encountered difficulties in understanding Git branching concepts, particularly differentiating between main, develop, and feature branches. Overcame this challenge through repeated practice with the Git Handbook and practical application on the feature/configure branch.
* **Bug Fixing**: Experienced an UnboundLocalError during the implementation of the visual report, requiring detailed debugging of the generate\_html\_report function. Successfully resolved the issue by correcting the f-string assignment.
* **File Management**: Faced challenges in locating generated files (report.html, chart.png, visrep.html) post-execution, a recurring issue similar to previous experiences with HTML file visibility. Resolved by verifying the script’s execution directory and refreshing the file explorer.

**Observations**

* **Performance Impact**: Noted that link validation using requests.head with a 5-second timeout can significantly slow down analysis for files with numerous hyperlinks, as observed during testing with example.md. Disabling this feature via config.json improves processing speed, suggesting a need for optimization in future iterations.
* **User Experience**: The visual report (report.html) with an embedded chart.png provides a clear summary, but the static nature of matplotlib charts limits interactivity compared to the dynamic plotly charts in visrep.html. Users may benefit from additional customization options in the visual output.
* **Code Robustness**: The load\_config function effectively handles missing or invalid config.json files by reverting to defaults, indicating a resilient design. However, the lack of input validation for min\_word\_length could lead to unexpected results with negative or zero values.
* **Scalability Concerns**: Testing revealed that the current implementation struggles with large Markdown files due to memory usage in BeautifulSoup parsing and matplotlib rendering, highlighting a potential need for batch processing or optimization in future enhancements.
* **Documentation Gaps**: While README.md covers basic usage, it lacks detailed examples of config.json customization and visual report interpretation, which could improve user adoption.

**Conclusion**

The period from April 18 to April 25, 2025, I established a functional development environment, acquired essential Git skills, and delivered a robust version of the Markdown File Analyzer, enhanced with a configuration system and a visual report feature that includes bar charts for data visualization. Challenges, including an UnboundLocalError and file management issues, were addressed effectively, with all work conducted in adherence to the company’s coding and documentation standards.