Project Essay: Developing an AI-Driven Financial Health Assessment Tool

Introduction: Problem Definition & Business Context

The financial literacy gap continues to be a significant obstacle to personal stability in an era of economic fluctuation, as many people find it difficult to transform disjointed spending data into a cohesive wealth-preservation strategy. The 50/30/20 budgeting framework addresses this by providing a clear guideline, recommending that 50% of income be allocated to essential expenses, 30% to discretionary spending, and 20% to savings or debt repayment (HSBC UK, 2025).

This project, *Finance Health Check 50/30/20*, translates that framework into a Python-based diagnostic tool that bridges raw financial data and actionable guidance. Rather than merely tracking expenses, the system evaluates financial health against structured benchmarks and delivers interpretable insights. The framework was chosen for both practical and personal reasons. While I follow the 50/30/20 rule for my own budgeting, I lack effective visualization to interpret spending patterns. Developing this tool transforms abstract numbers into intuitive visual insights that support daily financial decision-making.

The solution also carries broader business relevance. Demand is rising for hyper-personalized financial coaching that leverages AI to provide prescriptive, not just descriptive, insights (Smith, 2024). The tool could also serve as a secure, privacy-conscious employee wellness benefit, allowing staff to assess financial resilience. As the FinTech sector moves toward agentic solutions that automate complex analysis, systems combining structured financial logic with LLM-driven advisory capabilities are becoming central to modern digital financial health strategies (Zhang et al., 2025).

Project Motivation and Evolution

Initially, I envisioned the project as a standalone .exe application. I started with a random name and applied a methodology similar to BMAD to ensure precise and comprehensive requirements, leveraging my prior experience as a product manager, where defining requirements is a key strength. During development, I discovered that while agentic AI is effective at generating ideas, it sometimes overlooks best practices unless explicitly instructed. For example, the initial project structure did not adhere to standard practices, prompting a complete restructuring of the project and renaming it *Finance Health Check*. This temporarily disrupted the continuity of the project's GitHub history and the agentic coding workflow.

Additionally, the initial .exe user interface was visually limited, as shown in Figure 1, which motivated the transition to a Streamlit web application. This shift provided a more user-friendly interface, improved cross-platform accessibility, and enabled the implementation of dynamic visualizations, resulting in a smoother overall user experience.

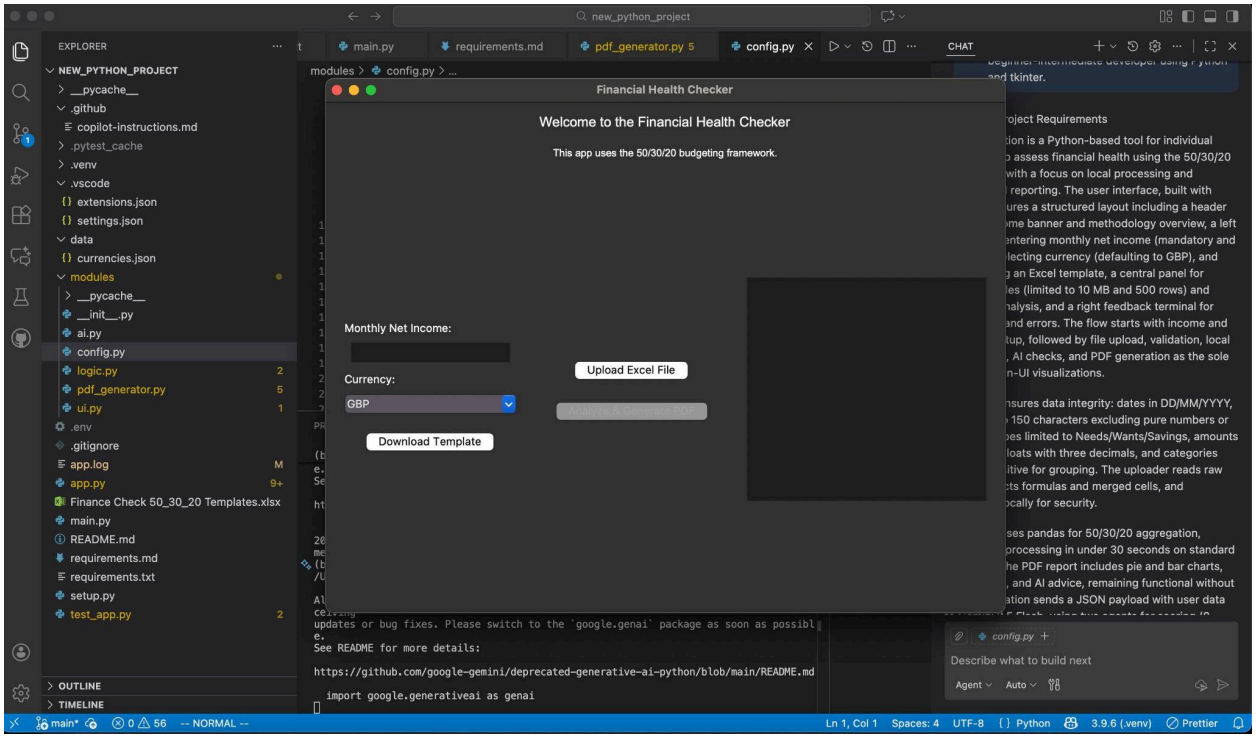

Figure 1. Finance Health Check .exe user interface displaying budget input and AI-generated advice

The core logic of the app relies on strictly structured templates for financial data entry to maintain data integrity. Users input their expenses according to these templates and select a currency from the available list, after which the app groups the data into the 50/30/20 categories (Needs, Wants, Savings) and further subcategories such as groceries, bills, or dining out. Based on these groupings, the app calculates a monthly financial score (1–100). The data and score are then sent to Gemini Flash 2.5 for AI-driven feedback and advice on improving or maintaining financial health. Token usage is limited to 300 per request to prevent hitting API limits, with a fallback mechanism in place. The app also generates PDF reports featuring bar and pie charts, providing immediate visual insights. To reduce complexity, no database is used, and all files are temporary and deleted after processing.

System Architecture & Technical Logic

The application is built on a modular Python backend that transforms raw financial data into a structured 50/30/20 analysis. Using Pandas, Excel files are ingested and validated to ensure dates, categories, and amounts follow strict data types. Expenses are categorized into Needs, Wants, and Savings through case-insensitive string matching, and a Financial Health Score (0–100) is generated based on deviations from the ideal allocation, as shown in Figure 2. Technical complexity across 84 currencies was managed with localized formatting and TDD, ensuring consistent scoring and preventing calculation errors, demonstrating robust enterprise-level engineering.

```python
def calculate_health_score(income, needs, wants, savings):
    """Calculate monthly budget health score based on 50/30/20 budgeting model.

    Uses directional deviation logic to penalize only risky behaviors:
    - Needs overspend (> 50%) - small penalty
    - Wants overspend (> 30%) - heavier penalty
    - Under-saving (< 20%) - heaviest penalty

    Weighted deviations:
    Dn = max(0, n - 0.5)      # Needs overspend
    Dw = max(0, w - 0.3)      # Wants overspend
    Ds = max(0, 0.2 - s)      # Under-saving
    Score = 100 × (1 - (0.2×Dn + 0.5×Dw + 0.6×Ds))

    Hard risk rules:
    - If savings < 0: Score = 0 (financial danger - debt/overspend)
    - If needs > 75%: Score -= 10 (extreme overspend)

    Final score is clamped to [0, 100].

    Args:
        income: Total monthly income
        needs: Total needs spending
        wants: Total wants spending
        savings: Total savings

    Returns:
        int: Health score from 0 to 100
    """
```

Figure 2. Implementation of the 50/30/20 budgeting logic used to compute the financial health score

Transitioning from a desktop .exe app to a Streamlit web interface enhanced accessibility, cross-platform compatibility, and real-time visualizations. Matplotlib and ReportLab were used to create dynamic charts and PDF reports. The system follows a Security by Design approach: all files are processed in-memory without persistent storage, and logs capture only operational metadata, excluding sensitive financial details (Government Security Group, 2025). Once a session ends or a report is generated, no financial data remains on the server, minimizing risk and meeting enterprise-grade privacy standards.

Methodology: Agile, TDD, and Iteration

Development followed an iterative, Agile-inspired workflow, beginning with rapid agentic AI-driven prototyping to establish core functionalities. As complexity grew, particularly with

multi-category expense handling and health score calculations, it became clear that AI alone could not ensure precision. I therefore adopted Test-Driven Development (TDD), defining 58 unit tests before implementing new logic. This approach allowed systematic identification of edge-case errors, promoted a “correct-by-design” mindset, reduced bug density, and provided a robust regression safety net for ongoing development.

A significant technical challenge involved integrating Gemini 2.5 Flash. Initial synchronous API calls caused UI latency and triggered fallback responses. Refactoring the module to an asynchronous (async/await) pattern resolved these issues, maintaining a high-performance user experience during intensive financial calculations. Alongside this, AI was used to automate Git commits, generate inline code comments for readability, and even review its own code for performance optimizations after major changes. Features such as preview tables and financial metrics were iteratively added, tested, and refined in line with Agile principles.

Critical Evaluation: The Agentic Coding Experience

Developing the Finance Health Check application provided a profound case study in the evolving paradigm of software engineering, where the developer’s role shifts from a low-level “coder” to a “concept-oriented designer.” In this agentic workflow, my main contributions were framing the problem, defining architectural constraints, and guiding the AI through complex logic changes, such as transitioning from synchronous to asynchronous API calls. While this approach enabled rapid prototyping, it came with a significant trade-off: a loss of granular, low-level control. Relying on AI occasionally produced “abstraction gaps,” where the generated code functioned correctly but lacked the intentionality of a manually crafted solution.

This challenge was particularly evident when auditing the AI's mathematical logic. For example, the AI initially calculated the Health Score using a simple average, overlooking the greater risk of under-saving compared to overspending on essentials. I addressed this by implementing a weighted penalty system, emphasizing the necessity of human oversight to ensure the algorithm reflected nuanced financial realities.

A broader evaluation of the AI’s performance revealed further limitations with standard software engineering practices. Without precise, high-context prompting, the AI often ignored fundamental best practices, such as modular logging and strict error handling. Early iterations over-generated files or used broad exception handling until I enforced centralized logging and specific try/finally blocks for resource cleanup. This demonstrates that while AI is a powerful engine, it lacks inherent “engineering rigor” unless guided by clear architectural rules.

Managing an agentic workflow also imposed substantial cognitive load. Unlike traditional coding, where the developer retains the entire logic flow, agentic coding requires a continuous human-in-the-loop approach. This includes auditing AI outputs and managing context challenges, a difficulty I encountered when project renaming disrupted GitHub Copilot suggestions. Although productivity increased in terms of code output, mental effort shifted toward validation and verification. Ultimately, this experience confirms that agentic tools are most effective when the developer maintains a requirements-first mindset, using AI to accelerate implementation without relinquishing responsibility for structural integrity and security.

Reflection & Future Directions

Developing the Finance Health Check application fundamentally redefined my perspective on software creation, shifting my identity from a traditional programmer to an AI orchestrator. In an agentic workflow, the developer's role increasingly mirrors that of a Product Owner. The primary challenge was not fixing syntax errors but maintaining a high-level architectural vision to ensure AI-generated modules remained cohesive, secure, and aligned with user requirements.

This shift reflects a broader trend in software development: the democratization of coding. Generative AI is lowering barriers to entry, allowing domain specialists such as financial analysts or managers to build robust, enterprise-grade systems without exhaustive knowledge of low-level programming syntax. By focusing on "intent" rather than "implementation," I was able to integrate complex features, such as asynchronous API calls and a 58-test suite, that would have traditionally required extensive manual labor.

Looking ahead, the project provides a foundation for more sophisticated financial modeling. Potential refinements include integrating a persistent database (e.g., PostgreSQL) to track multi-period trends and expanding multi-currency analysis using real-time exchange rate APIs. This evolution from a static tool to a dynamic data system would further enhance its business utility. Ultimately, this journey has taught me that while AI can provide the "muscle" for coding, the human developer remains the essential "brain," offering ethical judgment and strategic oversight necessary for responsible enterprise innovation.

References

Government Security Group (2025) *Secure by Design Principles*. UK Government Security. Available at: https://www.security.gov.uk/policy-and-guidance/secure-by-design/principles/ (Accessed: 28 January 2026).

HSBC UK (2025) *What is the 50-30-20 rule?* | *Budgeting methods*. Available at: https://www.hsbc.co.uk/financial-fitness/everyday-budgeting/spending-your-income/ (Accessed: 27 January 2026).

Smith, A. (2024) 'The Future of AI in Personal Wealth Management: From Tracking to Prescriptive Coaching', *FinTech Journal of Innovation*, 12(3), pp. 45–60.

Zhang, Q., Wang, Y., Hua, C., Huang, Y. and Lyu, N. (2025) 'Knowledge-augmented large language model agents for explainable financial decision-making', *arXiv* [Preprint]. Available at: https://arxiv.org/abs/2512.09440 (Accessed: 27 January 2026).

AI Declaration

This work was authored by me with the support of Generative AI tools (ChatGPT and Gemini), which were used for transcription of recorded materials and structural refinement of written drafts. All AI-generated suggestions were critically evaluated and substantially reworked to ensure they reflect my original analysis, interpretation, and research. The use of generative AI has been acknowledged and referenced within the assessment in line with the guidance provided in the student handbook. I retain full responsibility for the accuracy and academic integrity of the final submission.