# Name: sevan hurdoyal

Bachelor in cybersecurity

# AI Prompting Journal – Text Analysis Tool

## 1. Project Selection & Justification

The Text Analysis Tool offered a clear structure and allowed me to practice modular programming, data handling, and visualizations using natural language input.   
  
Key Reasons for Selecting the Text Analysis Tool:

- Easier to test and debug due to direct user input  
- Well-aligned with required skills like loops, functions, and pandas  
- Good scope for edge case testing for example, punctuation-only input, repetitive words  
- Provided better opportunities to refine AI-generated code step-by-step

Based on these factors, I selected the Text Analysis Tool as it allowed me to apply intentional prompting techniques in a manageable and educational way.

## 2. Structured AI Prompt Logs

Here are five intentional AI prompts I used during development which are aligned with the 6-step programming methodology

* Prompt 1: What are the basic features of a text analysis tool in Python that includes user input, stats, and a bar chart?

AI Response (Summary): Listed features like word count, sentence count, most frequent words, and matplotlib for visualization.

What I refined/changed: Asked follow-up: “How do I structure this with functions and imports?” to start modular planning.

**6-Step Stage: Step 1 (Restate the problem)**

* Prompt 2: How should the tool handle empty strings or punctuation-only text?

AI Response (Summary): Suggested using regex and checking for valid input length before processing.

What I refined/changed: Prompted for a reusable function to sanitize and filter input.

**6-Step Stage: Step 6 (Test with edge cases)**

* Prompt 3: Can you provide a modular pseudocode structure for the text analysis logic?

AI Response (Summary): Provided a basic top-down structure with get\_input(), analyze\_text(), display\_results().

What I refined/changed: Split analyze\_text() further into smaller helper functions.

**6-Step Stage: Step 4 + 5 (Pseudocode + Convert to Python)**

* Prompt 4: How do I generate a bar chart of top 5 frequent words from a dictionary in Python?

AI Response (Summary): Suggested using matplotlib.pyplot.bar() and sorting the dictionary values.

What I refined/changed: Prompted again to add axis labels and fix overlapping issues.

**6-Step Stage: Step 5 (Convert to Python)**

* Prompt 5: How can I use pyinputplus to get validated input from the user for a text input?

AI Response (Summary): Showed how to wrap input in a pyip.inputStr() function with validation.

What I refined/changed: Integrated this with a custom error message and re-prompt on empty input.

**6-Step Stage: Step 2 (Identify Input/Output)**

## 3. Before/After Prompting Examples

1. 🔄 Sentence Count Logic

Initial Prompted Code:

def count\_sentences(text):  
 return text.count('.')

Follow-Up Prompt:

“This doesn’t work for ‘?’ or ‘!’ endings. Can you update it?”

Final Improved Code:

import re  
def count\_sentences(text):  
 return len(re.findall(r'[.!?]', text))

Why It Was Effective:

Covers more punctuation types and uses regex for flexible pattern matching.

1. 🔄 Bar Chart Visualization

Initial Prompted Code:

plt.bar(word\_dict.keys(), word\_dict.values())  
plt.show()

Follow-Up Prompt:

“How do I fix overlapping words and sort by frequency?”

Final Improved Code:

top\_words = sorted(word\_dict.items(), key=lambda x: x[1], reverse=True)[:5]  
words, freqs = zip(\*top\_words)  
plt.bar(words, freqs)  
plt.xticks(rotation=45)  
plt.title('Top 5 Frequent Words')  
plt.show()

Why It Was Effective:

Improves readability and relevance of the chart.

1. 🔄 Average Word Length

Initial Prompted Code:

def avg\_word\_length(words):  
 return sum(len(word) for word in words) / len(words)

Follow-Up Prompt:

“What if the list is empty or words include punctuation?”

Final Improved Code:

def avg\_word\_length(words):  
 clean\_words = [w.strip('.,!?') for w in words if w.isalpha()]  
 return round(sum(len(w) for w in clean\_words) / len(clean\_words), 2) if clean\_words else 0

Why It Was Effective:

Handles punctuation and prevents division by zero.

## 4. Prompting Journal

Throughout the project, I used multiple prompting strategies to refine, fix, and improve AI-generated code. I found that clarification prompts helped most when narrowing down input/output issues, while challenge prompts were useful for testing edge cases. For modular design and reusable functions, foundation prompts worked best.  
  
Over time, my prompting evolved from asking broad questions, such as “Can you write a function for this?” to more specific and structured prompts, like “Can you modularize the input handling using pyinputplus with error validation?” I also learned to ask for pseudocode before requesting final code.  
  
In a few cases, the AI misunderstood the context, especially with sentence boundaries and regular expression (regex) use. I redirected it by breaking the problem into smaller prompts and testing each part. This helped improve both accuracy and reliability.

One of the biggest takeaways from this project is that using AI doesn’t mean giving up control it made me feel more involved in the process. I still had to think through the logic, make decisions, and adjust the code when things didn’t work perfectly. The AI just helped me stay on track and made the process less overwhelming.

This experience helps me understand that an AI tool support throughout the process. The AI won’t get you better output unless you put more efforts and detailed prompting go hand in hand. Prompting makes AI understand your process in depth and get you better results. This didn’t build the project for me, but it helped me think more clearly and write better, more organized code. I now feel more confident about using prompting techniques in future projects, not just to save time, but to improve how I plan and build my solutions.

**Lessons Learned:**

While working on this project, I learned how to use AI in a better way. At first, I was just asking normal questions, but later I realized that clear and simple prompts gave better results. The more specific the prompting was, the better and more useful the responses were. I also realized that AI isn’t perfect sometimes; the output was incorrect, too simple, or missed key points. In those cases, I had to adjust my prompt or fix the code myself.

Another important lesson was the importance of proper planning. By following the 6-step methodology, I didn’t get stuck trying to solve everything at once. Understanding the problem in smaller parts made it easier to debug and improve each feature. I also got better at writing clean, modular code that can be reused later.

I now understand how valuable it is to test with **edge cases** like empty input or strange characters. I now understand that AI is useful, but I still need to think on my own. It can help me, but it can’t do everything for me. This project made me feel more confident in using both AI and Python together.

## 5. GitHub Commit Log Summary

* Initial project setup and created main file
* Added text input and validation using pyinputplus
* Implemented word count, sentence count, and average word length
* Created bar chart for top 5 frequent words
* Tested edge cases and improved input handling
* Final clean-up and added README.md file