AI Prompting Journal – Text Analysis Tool

# 1. Project Overview

The Text Analysis Tool is a console-based Python application that accepts user-input text, processes it, and generates statistics like word count, sentence count, average word length, and a bar chart of the most frequent words. This project was developed using intentional prompting and modular programming principles.

# 2. 6-Step Programming Methodology

## Step 1: Restate the Problem

Prompted the AI to outline the scope of the Text Analysis Tool including all required features: input text, modular design, statistics, visualization, and pyinputplus-based UI.

## Step 2: Identify Input/Output

Clarified inputs such as user-entered text and outputs including statistics and charts. Edge cases and data types were also identified.

## Step 3: Try Solving Manually

Manually analyzed a sample input text to determine word and sentence counts, average word length, and expected chart values.

## Step 4: Request Pseudocode

Asked the AI to create modular pseudocode that matched the planned logic, which was later used to write actual functions.

## Step 5: Convert to Python

Gradually converted pseudocode to Python by prompting for clean implementations of each function, starting from text input to output display.

## Step 6: Test With Edge Cases

Tested extreme input examples such as empty strings, only punctuation, all caps, and repetitive words. Captured and exported results.

# 3. Reflection (300–400 Words)

Working with AI as a collaborative programming assistant helped break the problem into manageable steps. Each intentional prompt was crafted to focus on a particular outcome: understanding the task, solving logic manually, generating modular code, and refining output. The AI responded well to detailed prompts and especially excelled when asked for pseudocode or visual summaries.  
  
One of the key takeaways was the importance of iteration. Initially, the logic for counting sentences or calculating average word length was too simplistic, but refining prompts allowed for optimization. Testing edge cases ensured the tool was robust and handled invalid or short input gracefully.  
  
Prompting strategies like 'Challenge' and 'Clarification' helped troubleshoot errors and improve modularity. Using a foundation of pseudocode also made Python implementation smoother.  
  
Overall, this exercise emphasized that intentional prompting is not just about getting code, but thinking clearly, breaking problems down, and refining ideas step-by-step. It simulates pair programming with a very patient and smart assistant. This approach will definitely be valuable in future projects.

# 4. Before/After Prompting Examples

## Example 1: Word Frequency Counter

💬 Initial Prompt:

Can you count how often each word appears in a list of words?

❌ Initial AI Response:

Used a manual loop with a dictionary to count frequencies — worked, but was verbose and inefficient.

💬 Follow-Up Prompt:

Can we use `collections.Counter` to make this shorter and more efficient?

✅ Improved AI Response:

Used `Counter(words)` which automatically returns a dictionary of word frequencies.

## Example 2: Sentence Tokenization

💬 Initial Prompt:

Split text into sentences using periods.

❌ Initial AI Response:

Split using `text.split('.')` — did not handle `!`, `?`, or extra spaces.

💬 Follow-Up Prompt:

Can we split by `.`, `!`, or `?` and remove whitespace using regex?

✅ Improved AI Response:

Used `re.split(r'[.!?]+', text)` followed by `.strip()` to get clean sentence tokens.

## Example 3: Word Frequency Chart

💬 Initial Prompt:

Create a chart for most frequent words.

❌ Initial AI Response:

AI tried to print words with asterisks, not a real plot.

💬 Follow-Up Prompt:

Can we use `matplotlib` to create a bar chart from a dictionary of word frequencies?

✅ Improved AI Response:

Generated a clear bar chart using `plt.bar()` and `plt.show()`.