Paper ET05

Integrating R Programming and Generative AI for Prompt-Driven Clinical Trial Data Analysis

Vyom Bhatt, Syneos Health, Ahmedabad, India

ABSTRACT

This paper explores integrating Generative AI with R programming for clinical trial data analysis. By leveraging prompt engineering, schema-driven workflows, and mock shells, analysts can automate TLF generation while maintaining accuracy and compliance. We compare different approaches—natural prompts, schema-based, and mock-shell-driven—and outline best practices for reproducibility, validation, and audit trails. The integration of AI and R accelerates reporting, reduces manual errors, and ensures regulatory adherence in modern clinical workflows..

INTRODUCTION

Generative AI and R programming together offer a transformative approach to clinical data analysis. AI interprets natural language prompts, while R ensures robust statistical outputs. This synergy reduces coding effort, improves reproducibility, and supports compliance, making it ideal for creating validated tables, listings, and figures in regulated environments.

OVERVIEW OF GENERATIVE AI AND R IN CLINICAL RESEARCH

GENERATIVE AI IN CLINICAL RESEARCH

Generative AI can take plain-language requests and turn them into structured outputs or even R code. This helps teams quickly move from ideas to analysis without starting from scratch.

R PROGRAMMING FOR BIOSTATISTICS

R is a trusted tool in clinical research. It handles CDISC datasets, creates tables, listings and figures, and supports advanced statistical models—all while keeping results reproducible and compliant.

INTEGRATION BENEFITS

Bringing AI and R together saves time, reduces manual coding errors, and makes workflows more consistent. It also helps meet regulatory standards by improving traceability and reproducibility.

PRACTICAL IMPLEMENTATION

The key is good prompt design and responsible AI use. With these in place, analysts can generate validated R scripts directly from prompts, speeding up trial reporting without sacrificing quality.

PROMPT ENGINEERING: KEY PRINCIPLES FOR STATISTICAL PROGRAMMERS

ASSIGNING A ROLE TO GENERATIVE AI BEFORE PROMPTING

Assigning a role to Generative AI before prompting is a prompt engineering best practice that helps guide the model's behaviour and output. You explicitly tell the AI who it should act as and what perspective or expertise to adopt before giving the main instruction. This sets context and ensures the response aligns with your domain needs.

- **Sample prompt:** *"You are an experienced clinical statistical programmer following CDISC and regulatory standards."*

BE CLEAR AND SPECIFIC

When writing a prompt, don't assume the AI knows what you mean. Spell out the dataset, population, metrics, and output format so there's zero guesswork.

- **Basic prompt:** *"Create a SOC/PT safety table."* (Too vague—what dataset? Which metrics? What format?)
- **Better prompt:** *"Create SOC/PT table using ADAE dataset with counts and column % by TRT01P; population SAFFL=='Y'; output RTF."*

INCLUDE CONTEXT AND CONSTRAINTS

Give the AI all the details upfront—filters, flags, sorting rules, and any statistical assumptions. This avoids back-and-forth and ensures the output matches your expectations.

- **Basic prompt:** *"Show adverse events."* (No mention of flags, sorting, or precision.)
- **Better prompt:** *"Include SAE and AESI flags; sort rows by overall count; show 1 decimal."*

SPECIFY OUTPUT AND FORMATTING

Don't just ask for an output—tell the AI exactly how it should look. Include file type, naming, and layout details like decimals, footnotes, and sorting.

- **Basic prompt:** *"Give me a table."* (No file type, no naming, no formatting details.)
- **Better prompt:** *"Output as RTF table named t_safety_soc_pt.rtf; add footnote: 'Percentages by column total.'"*

WAYS TO INTEGRATE GENERATIVE AI AND R IN CLINICAL DATA ANALYSIS

QUICK GLOSSARY (TO KEEP US ALIGNED)

- **Natural prompt:** Free-text request (e.g., "Create SOC/PT table using ADAE dataset with counts and column % by TRT01P…").
- **Prompt schema:** Structured spec (YAML/JSON) capturing task, dataset, population, grouping, metrics, output, controls.
- **Mock shell:** The visual/table template (Word/Excel/PDF) used in studies (headers, columns, footnotes, sorting).
- **TLFs:** Tables, Listings, Figures (clinical outputs aligned to mock shells).

WORKFLOWS FOR GENERATING TLFS WITH GENERATIVE AI AND R

- **Natural prompt → R code**
  It's the fastest path. The AI directly converts your plain-language request into R code without an intermediate specification.
- **Natural prompt → Prompt schema → R code**
  The AI first structures your request into a clear, auditable schema (dataset, population, metrics, formatting) usually in YAML/JSON, the validated schema then drives deterministic R code generation.
- **Natural prompt → Prompt schema + Mock shell → R code**
  It's most robust. The AI first structures intent into a schema, then aligns layout to the uploaded mock shell. Validated specs drive code generation, producing tables that match study format.

Let's explore each workflow in detail, using the example of generating an Adverse Event SOC/PT table for better understanding.

NATURAL PROMPT → R CODE

FLOW

- Analyst writes a free-text request.
- LLM generates R code directly (e.g., dplyr + flextable).
- Analyst runs code, manually tweaks layout to match study shell.

STRENGTHS

- Very fast; minimal ceremony.
- Good for ideation and exploratory analyses.

LIMITATIONS

- Ambiguity (percent denominator, sorting, population nuances).
- Harder to audit; no intermediate spec.
- Reproducibility can vary with small prompt changes.

EXAMPLE PROMPT

"Create SOC/PT table using ADAE dataset with counts and column % by TRT01P; population SAFFL=='Y'; output RTF."

TYPICAL R (ILLUSTRATIVE)

```
library(haven); library(dplyr); library(tidyr); library(flextable)
adsl <- read_xpt("adsl.xpt")
adae <- read_xpt("adae.xpt")
pop <- adsl %>% filter(SAFFL == "Y") %>% select(USUBJID, TRT01P)
adae_pop <- inner_join(adae, pop, by = "USUBJID")

tbl <- adae_pop %>%
  count(AEBODSYS, AEDECOD, TRT01P, name = "n") %>%
  group_by(TRT01P) %>%
  mutate(pct = round(100 * n / sum(n), 1)) %>%
  ungroup() %>%
  pivot_wider(names_from = TRT01P, values_from = c(n, pct))

ft <- flextable(tbl) %>% autofit()
print(ft, preview = "rtf")
```

BEST-FIT SCENARIOS

- Brainstorming TLF designs.
- Early feasibility checks.
- Hackathons / internal demos.

NATURAL PROMPT → PROMPT SCHEMA → R CODE

FLOW

- Analyst prompt.
- Convert to schema (manual or LLM) capturing datasets, variables, metrics, flags, output.
- Validate schema (CDISC names, SOP rules).
- Generate R code (preferably via templates).
- Execute; QC (unit tests, numeric checks); audit trail stores prompt + schema + code + outputs.

STRENGTHS

- Low ambiguity: schema pins down intent.
- High reproducibility: schemas are versionable artifacts.
- Audit ready: schema = mini spec consistent with GxP mindset.
- Works well even without a mock, for standard layouts.

LIMITATIONS

- Slightly more steps (prompt → schema → code).
- Requires schema templates and validation rules.

EXAMPLE SCHEMA (YAML)

```
task: "summary_table"
dataset: "ADAE"
population: { include: "ADSL where SAFFL == 'Y'" }
grouping: { rows: ["AEBODSYS", "AEDECOD"], columns: ["TRT01P"] }
metrics: { counts: true, percent: "by column total" }
flags: { include: ["SAE", "AESI"] }
display: { decimal_places: 1, order_by: "descending by counts overall" }
output: { type: "rtf_table", filename: "t_safety_soc_pt.rtf" }
controls: { seed: 202501, code_style: "tidyverse", audit_level: "full" }
```

BEST-FIT SCENARIOS

- Regulated study TLFs with standard shells.
- Cross study reusability and automation.
- Teams that must maintain traceability and validation.

NATURAL PROMPT → PROMPT SCHEMA → R CODE

FLOW

- Upload mock shell → extract layout (columns, header hierarchy, footnotes, sorting).
- Analyst prompt → schema (datasets, population, metrics, flags, output).
- Validation: schema vs. SOP; layout vs. shell rules.
- Code generation (template/LLM) guided by both layout and schema.
- Execute; QC:
  - Numeric checks (denominators, totals).
  - Visual regression vs. mock shell (vdiffr for plots; RTF compare).
- Audit trail: prompt, schema, mock shell reference, code, environment, outputs, reviewer sign off.

STRENGTHS

- Highest reliability: both what (schema/spec) and how (mock layout) are enforced.
- Best compliance posture: traceability and reproducibility.
- Minimizes interpretation errors (denominators, sorting, flags).

LIMITATIONS

- Slightly heavier process (but worth it for study deliverables).
- Needs multimodal capability and schema validation rules.

TYPICAL R (ILLUSTRATIVE FORMATTING CALLS)

```
ft <- flextable(tbl) %>%
```

```
  set_header_labels(AEBODSYS = "System Organ Class", AEDECOD = "Preferred Term") %>%
  add_footer_lines("Source: ADAE; Population: SAFFL=Y; Percentages by column total")
%>%
  width(width = 1.2) %>% align(align = "center", part = "header") %>% autofit()
```

BEST-FIT SCENARIOS

- Final TLFs for CSRs/major milestones.
- Sponsor/CRO settings with strict SOPs and audits.

TYPICAL FAILURE MODES & HOW EACH APPROACH ADDRESSES THEM

Failure Mode	Natural prompt → R code	Natural prompt → Prompt schema → R code	Natural prompt → Prompt schema + Mock shell → R code
Percent denominator wrong	☑ Common	◇ Rare	✗ Unlikely
Wrong population filter	☑ Common	◇ Rare	✗ Unlikely
Layout mismatches to shell	◇ Common	◇ Common	✗ Unlikely
Ambiguous flags/terms	☑ Common	◇ Rare	✗ Unlikely
Audit/trace gaps	☑ Major	✗ Minimal	✗ Minimal

☑ = frequent risk; ◇ = occasional; ✗ = low risk.

IMPLEMENTATION TIPS (TO MAKE THIS ROBUST)

PROMPT TEMPLATES

Create ready-to-use prompt starters for each type of TLF (e.g., Safety tables, Kaplan-Meier plots, Lab shift summaries). This saves time and ensures consistency across analyses.

SCHEMA LIBRARY

Maintain a version-controlled library of YAML templates for common analyses like Safety, KM, and Lab Shift. These act as structured blueprints for generating accurate and auditable R code.

VALIDATION HOOKS

Build checks that verify CDISC variable names, allowed metrics, and denominator rules before code runs. This prevents errors and ensures compliance with SOPs and regulatory standards.

REPRODUCIBILITY

Always set seeds for random processes and log model versions. This guarantees that results can be reproduced exactly, which is critical for regulatory audits and scientific integrity.

AUDIT TRAIL

Capture everything: the original prompt, schema, mock shell reference, generated code, environment details, outputs, and reviewer sign-off. This creates a complete trace for compliance and transparency.

CONCLUSION

Combining Generative AI with R programming streamlines TLF creation, enhances efficiency, and ensures compliance. Structured prompts, schema validation, and audit trails minimize ambiguity and errors. This integrated approach represents a practical, future-ready solution for accelerating clinical trial reporting without compromising quality or regulatory standards.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Vyom Bhatt
Company: Syneos Health
Address: Ahmedabad, Gujarat, India.
Work Email: vyom.bhatt@syneoshealth.com
Email: vyom.b.bhatt@gmail.com
LinkedIn: http://linkedin.com/in/vyom-bhatt-1ab2b2239