ET05: Integrating R Programming and Generative AI for Prompt-Driven Clinical Trial Data Analysis

Presented by: Vyom Bhatt
M.Sc. Statistics
Statistical Programmer

Friday, 20 February 2026

PHUSE APAC Connect 2026

Session Flow

01 | Ways to integrate Generative AI and R in clinical data analysis

02 | Workflows for Generating TLFs with Generative AI and R

03 | Natural prompt → R code

04 | Natural prompt → Prompt schema → R code

05 | Natural prompt → Prompt schema + Mock shell → R code

06 | Typical Failure Modes & How Each Approach Addresses Them

07 | Implementation Tips

Ways to integrate Generative AI and R in clinical data analysis

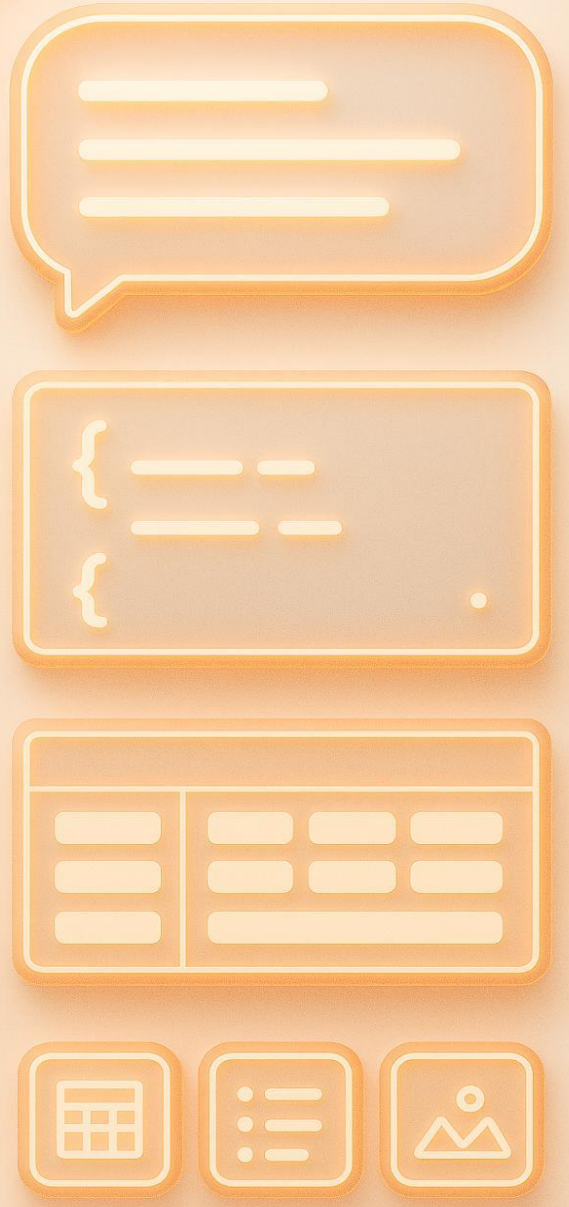
Quick Glossary (to keep us aligned)

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

(A) Natural prompt → R code

Fastest path: The AI directly converts your plain-language request into R code without an intermediate specification.

(B) 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.

(C) Natural prompt → Prompt schema + Mock shell → R code

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.

(A) 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.

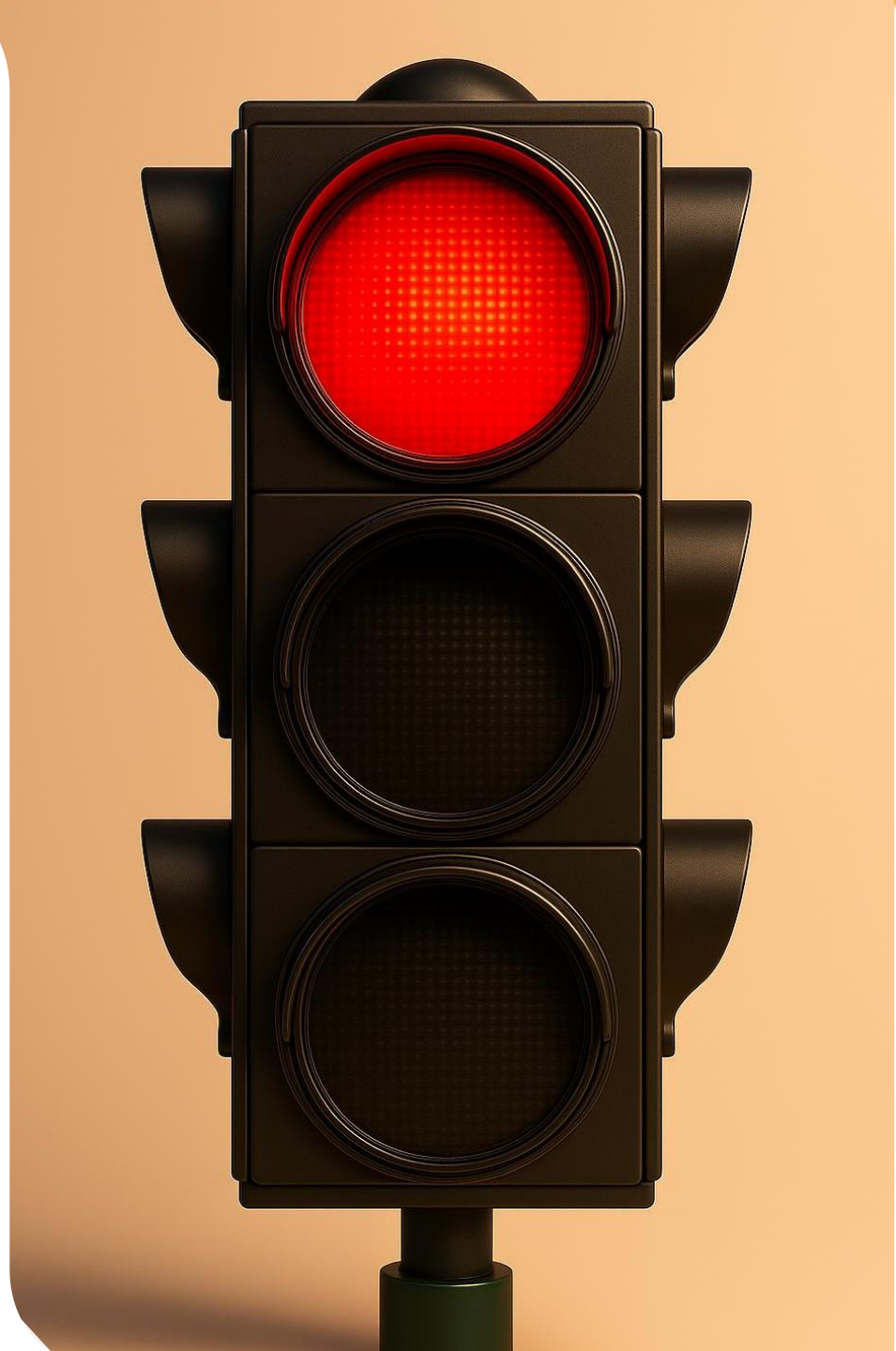
(A) Natural prompt → R code

Example prompt

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

Typical R (illustrative)

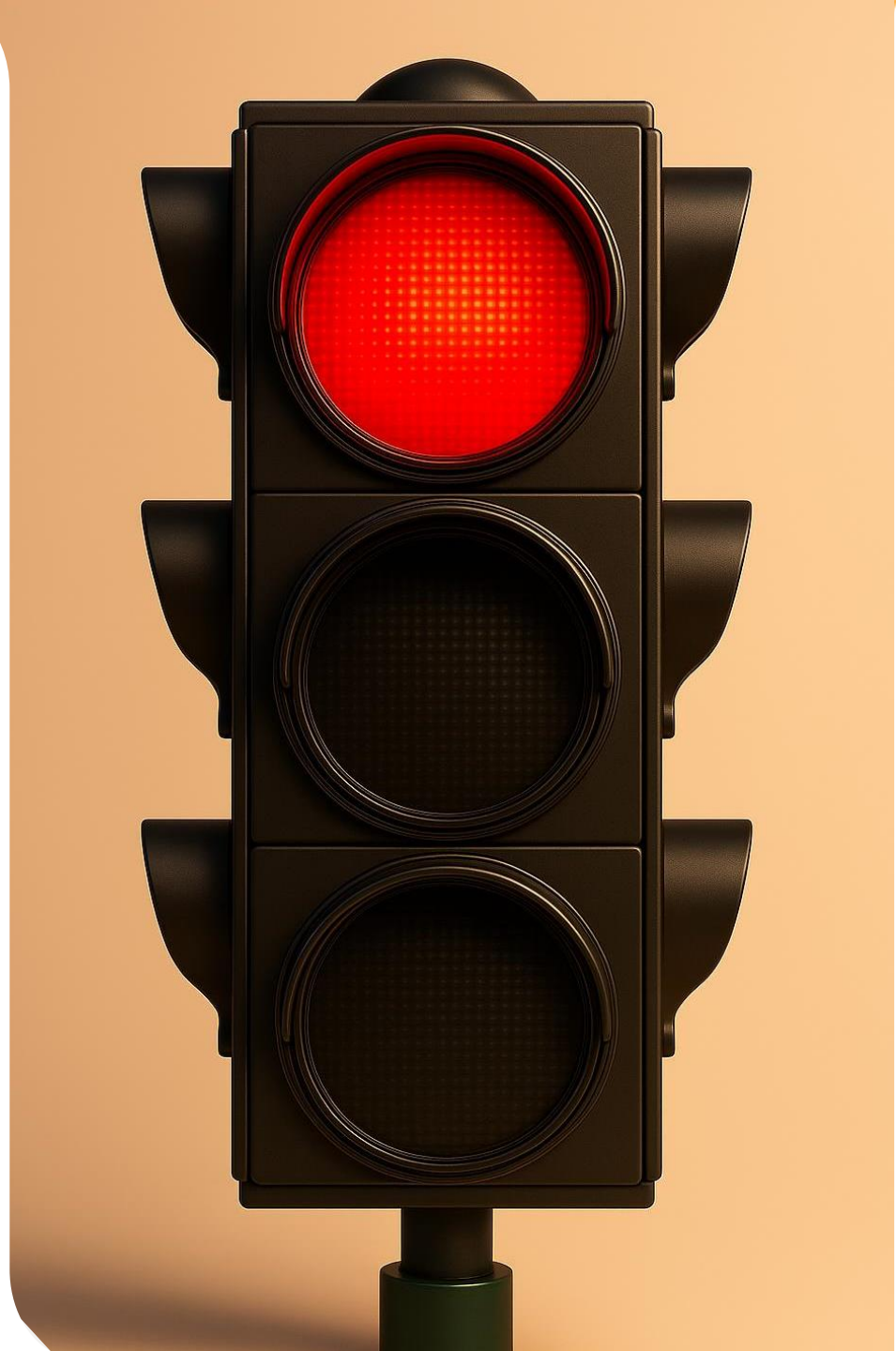
```r
library(haven); library(dplyr); library(tidyr); library(flextable)
adsl <- read_xpt("adsl.xpt")
adae <- read_xpt("adae.xpt")
pop <- adsl %>% filter(SAFFL == "Y") %>% select(USUBJID, TRT01P)
adae_pop <- inner_join(adae, pop, by = "USUBJID")

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")
```



(A) Natural prompt → R code

Best-fit scenarios

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



(B) 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.

(B) Natural prompt → Prompt schema → R code

Limitations

~ Slightly more steps (prompt → schema → code).

~ Requires schema templates and validation rules.

Example schema (YAML)

```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" }
```

(B) Natural prompt → Prompt schema → R code

Best-fit scenarios

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

(C) Natural prompt → Prompt schema + Mock shell → 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.

(C) Natural prompt → Prompt schema + Mock shell → R code

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)

```r
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()
```

(C) Natural prompt → Prompt schema + Mock shell → R code

Best-fit scenarios

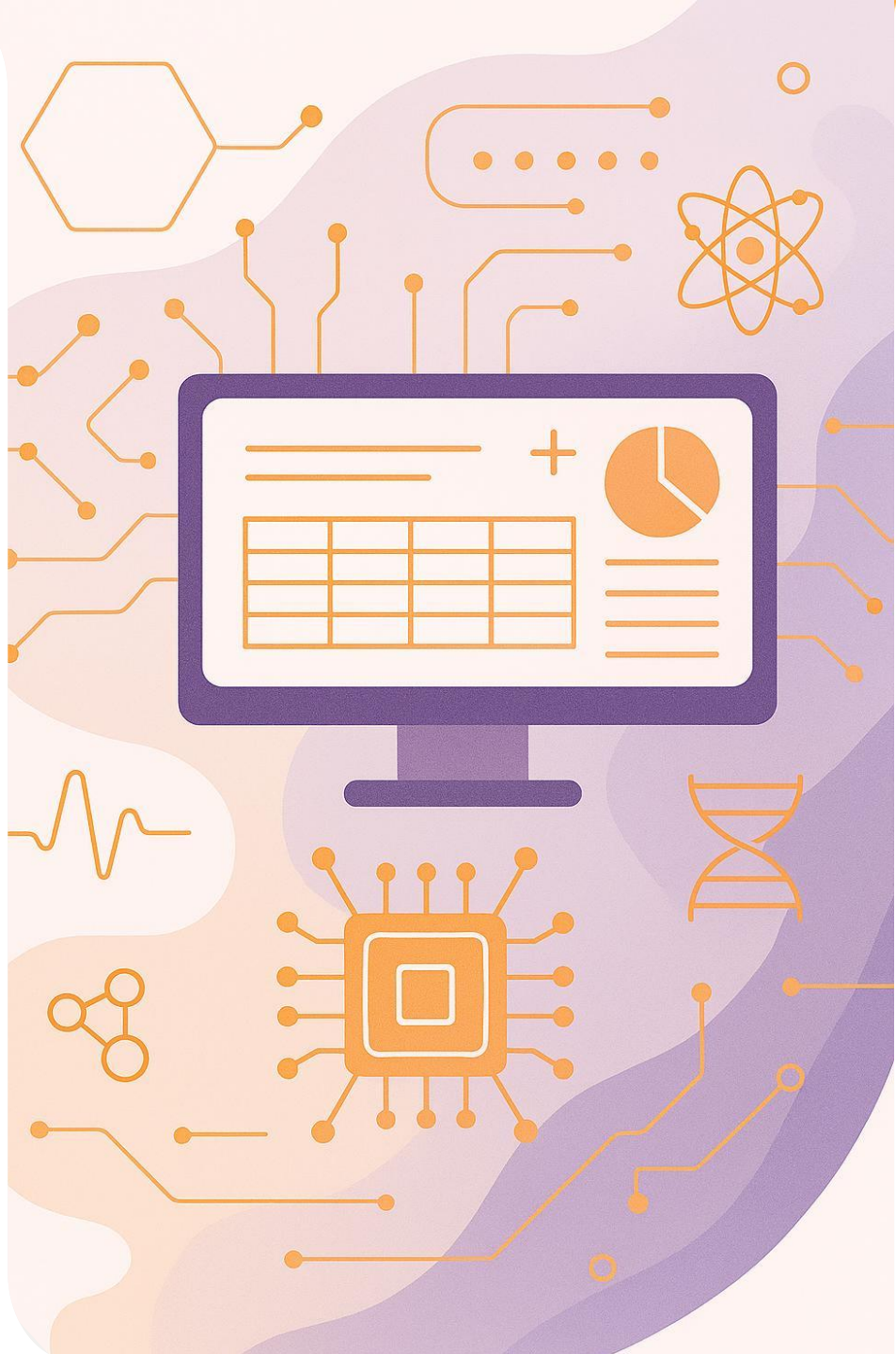
- Final TLFs for CSRs/major milestones.
- Sponsor/CRO settings with strict SOPs and audits.

Typical Failure Modes & How Each Approach Addresses Them

Failure Mode	(A) Natural prompt → R code	(B) Natural prompt → Prompt schema → R code	(C) 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

Implementation Tips (to make this robust)

- **Prompt templates:** Create standard prompt starters per TLF type.
- **Schema library:** Versioned YAML templates (Safety, KM, Lab Shift).
- **Validation hooks:** Check CDISC names, allowed metrics, denominator rules.
- **Reproducibility:** set seeds, model version logging.
- **Audit trail:** Store prompt, schema, mock shell ref, code, environment, output, review.



Syneos® Health

Shortening the distance from lab to life®.

Appreciate Your Engagement!

Shaping the Future of Statistical Programming—Smarter, Faster, Compliant.

Vyom Bhatt

M.Sc. Statistics
Statistical Programmer
vyom.bhatt@syneoshealth.com