A 55-Item Initiation Checklist for zzcollab Data Analyses

Ronald 'Ryy' G. Thomas

2026-04-29

Figure 1: A clean workspace with a printed checklist, an open notebook, and a single CSV file open on screen, signalling deliberate setup before the first line of analysis code is written.

A documented, numbered checklist turns the first week of a data analysis project from improvisation into routine. The gain is not speed; it is the disappearance of the small omissions that surface, weeks later, as untestable models or unreproducible figures.

Introduction

A clinical research collaborator emails a single attachment. The body of the email reads, in part:

> Attached is the analytic dataset for Study A, frozen as of last Friday. Twelve columns, two hundred and eighty-seven rows, one row per participant per study visit. The protocol is IRB #2024-XYZ. Please treat as confidential; no PHI but a DUA is in place. An updated extract may follow if a few late withdrawals come in, but the structure will be the same. We expect to write this up for submission in Q3.

Opening the file reveals a header row that reads:

```
ParticipantID,Status,Sex,Group,VisitAge,VisitType,
Outcome_A_Imp,Outcome_B_Imp,Sub_a,Sub_b,Sub_c,Sub_total_Imp
```

Some columns are immediately legible (`ParticipantID`, `Sex`, `Group`). Others are not. What does the `_Imp` suffix mean? Is `VisitAge` the participant's age at the visit, or the age of the visit measured from baseline? Is `Sub_total_Imp` a sum of `Sub_a`, `Sub_b`, `Sub_c`, an average, or something else? Twelve columns, two hundred and eighty-seven rows, no codebook, no README. The clock starts on reply 'received, will review'. By Q3 a reproducible compendium is needed that another analyst could clone and re-run.

The fifty-five items below are a six-phase preparation checklist that moves this attachment from inbox to publishable compendium. The structure is the same whether the dataset is a clinical trial, an observational cohort, a simulation output, or any other tabular extract that will be analysed in a zzcollab research compendium.

The Hypothetical Client Dataset

To anchor the items, assume the following throughout.

File: `studyA_data_2026Q1.csv` (a single UTF-8 CSV with a BOM in the header).

Shape: twelve columns, two hundred and eighty-seven rows. One observation per participant per study visit.

Columns (in header order):

Column	Apparent type	Notes from the email
`ParticipantID`	integer	participant identifier
`Status`	categorical	enrolment status
`Sex`	categorical	M or F
`Group`	categorical	randomised arm
`VisitAge`	numeric	age (years)
`VisitType`	categorical	baseline + follow-up
`Outcome_A_Imp`	numeric	primary outcome A
`Outcome_B_Imp`	numeric	primary outcome B
`Sub_a`	integer	subscale item
`Sub_b`	integer	subscale item
`Sub_c`	integer	subscale item
`Sub_total_Imp`	numeric	subscale composite

What is missing:

- A codebook or data dictionary.
- A README documenting provenance, IRB protocol, embargo, and the meaning of `_Imp`.
- Any indication of which columns can be missing (structural NAs versus unexpected ones).

- The pre-specified statistical analysis plan (SAP), other than 'we will write up Outcomes A and B as primary'.

These gaps are the substrate the checklist operates on.

Why a Checklist?

A checklist is not a workflow. It is a set of named, verifiable end states ordered such that earlier items unblock later ones. The annotation paragraphs that accompany each item are scaffolding for first-time use; the durable feature is the numbering.

A numbered checklist solves three problems that recur on every project. First, it surfaces the steps that are easy to skip under deadline pressure (data dictionary, baseline integrity test, fresh-container reproducibility run). The items most often skipped are also the items reviewers most often ask for. Second, it gives collaborators a shared vocabulary: 'Item 17 is failing in the new tinytest file' is a precise bug report. Third, it forces reproducibility verification before results are circulated, not after.

The zzcollab Workspace

The target state is a zzcollab research compendium: an R package layout extended with `analysis/`, an `inst/tinytest/` directory for data-integrity tests, and the Five Pillars (Dockerfile, renv.lock, .Rprofile, source code, data) that together specify the computational environment. The checklist below is written against that layout. A fresh workspace is created with `zzc analysis` and contains, in broad outline:

```
projectname/
|-- analysis/
|   |-- data/
|   |   |-- raw_data/         (the client's CSV lives here)
|   |   `-- derived_data/     (cleaned files, model RDS)
|   |-- scripts/              (numbered analysis scripts)
|   |-- report/               (Rmd / qmd reports)
|   `-- figures/              (publication-ready outputs)
|-- R/                        (reusable functions)
|-- inst/tinytest/            (data integrity assertions)
|-- tests/testthat/           (function unit tests)
|-- docs/                     (data dictionary, checklist)
|-- DESCRIPTION
|-- Dockerfile
|-- renv.lock
|-- .Rprofile
|-- Makefile
`-- zzcollab.yaml
```

Every checklist item references a specific path inside this tree. Reading the items alongside the layout shows how the work product accumulates.

The 55-Item Preparation Checklist

The items are organised into six phases. Each phase corresponds to a deliverable that lives at a documented location.

Phase	Items	Deliverable
1	1-8	A buildable workspace with documented raw data
2	9-22	A cleaned, validated derived dataset
3	23-31	An EDA report with Table 1 and trajectory plots
4	32-38	Tested R functions for summary, modelling, plotting
5	39-45	A primary-analysis report with sensitivity analyses
6	46-55	A reproducible compendium ready for archival

Items not applicable to a particular project should be marked 'N/A' with a one-line reason rather than removed. The numbering is meant to be stable across projects so that two analysts can compare progress unambiguously.

Phase 1: Project Setup (Items 1-8)

Phase 1 produces a buildable workspace with raw data in place and provenance documented.

- ☐ **Verify `Dockerfile` exists and builds successfully.** The Dockerfile defines the computational environment that every subsequent step will run inside; an unbuildable Dockerfile blocks every phase that follows.

```
zzc docker --build
docker images studyA            # confirm recent build
```

- ☐ **Verify `renv.lock` contains required packages.** The lockfile should list only direct dependencies at this stage; transitive dependencies are resolved by `renv::restore()` inside the container. The lockfile is the second of the Five Pillars and pins the package versions that make the analysis reproducible.

```
jq '.Packages | keys' renv.lock | head -20
```

- ☐ **Verify `.Rprofile` has appropriate R options.** Confirm `.Rprofile` activates renv on session start:

```
# .Rprofile
source('renv/activate.R')
options(renv.config.auto.snapshot = TRUE)
```

  The zzcollab template also configures auto-restore on session start so that a fresh container picks up the lockfile automatically.

☐ **Verify source code directories exist.** The layout is the contract the rest of the checklist relies on; if `R/` does not exist, Phase 4 has nowhere to put its functions.

```
ls -d R/ analysis/scripts/ analysis/data/ tests/ \
      inst/tinytest/ docs/

# Create any missing directories
mkdir -p R analysis/scripts analysis/data/raw_data \
         analysis/data/derived_data \
         analysis/figures analysis/report \
         tests/testthat inst/tinytest docs
```

☐ **Verify raw data is in `analysis/data/raw_data/` (read-only).** With write permissions on the raw file, an analyst can open it in Excel, edit a cell, save, and never know they have changed the upstream data. With `chmod 444` set, every save attempt fails loudly.

```
mv ~/Downloads/studyA_data_2026Q1.csv \
   analysis/data/raw_data/
chmod 444 analysis/data/raw_data/studyA_data_2026Q1.csv
ls -la analysis/data/raw_data/
```

☐ **Update `analysis/data/README.md` with data source, date received, and use restrictions.** Document who provided the file (the collaborator's name and institution), the date and time of receipt, the IRB protocol number (`#2024-XYZ`), the DUA status, and the embargo. Quote the relevant sentences from the email verbatim. A minimal template:

```
# Study A Raw Data

- **Source:** Dr. <Name>, <Institution>
- **Received:** 2026-04-29 via email attachment
- **IRB protocol:** #2024-XYZ
- **DUA:** in place; treat as confidential
- **Embargo:** none stated
- **Update window:** 'an updated extract may follow if
  a few late withdrawals come in' (per email).

## Files

- `studyA_data_2026Q1.csv` (12 cols, 287 rows)
```

☐ **Record data dictionary: column definitions, units, coding schemes.** Create `docs/data-dictionary.md` with one entry per column. Resolve undocumented suffixes (`_Imp`) with the data provider before writing the entry; do not guess. A minimal template for one column:

```
## ParticipantID

- **Type:** integer
- **Range:** 1001-9999
- **Missing:** never
```

```
- **Interpretation:** unique participant identifier
- **Notes:** 4-digit; first digit is study site
```

Repeat for all twelve columns. Treat the dictionary as a contract between the data provider and the analyst; the integrity tests in Phase 2 should validate the contract, not the data.

☐ **Install analysis packages: `tidyverse`, `here`, `janitor`, `skimr`.** Enter the container, install the packages, snapshot the lockfile, and add each package to the DESCRIPTION Imports field so the declared dependency set matches the installed environment.

```
make r
```

```
install.packages(c(
  'tidyverse', 'here', 'janitor', 'skimr',
  'tinytest', 'gtsummary', 'broom', 'broom.mixed',
  'lme4', 'patchwork', 'assertthat'
))
renv::snapshot()
```

Phase 2: Data Ingestion and Validation (Items 9-22)

Phase 2 produces a cleaned, validated derived dataset and a test file that asserts every contract from the data dictionary. The deliverables are `R/load_data.R`, `inst/tinytest/test_data_integrity.R`, and `analysis/scripts/01-clean-data.R`.

☐ **Create `R/load_data.R` with a function to read raw data.** The function returns the raw data frame, never a printed object. Use `here::here()` for the path so the function works from any working directory. The CSV carries a UTF-8 BOM in the header; `read_csv()` handles this automatically.

```
# R/load_data.R
load_raw_data <- function() {
  readr::read_csv(
    here::here(
      'analysis/data/raw_data/studyA_data_2026Q1.csv'
    ),
    show_col_types = FALSE
  )
}
```

☐ **Create `inst/tinytest/test_data_integrity.R`.** The test file sources `load_raw_data()` once and runs every assertion (Items 11-17) against the resulting data frame. Run with `tinytest::run_test_file()`. The test file is the living version of the data dictionary contract.

```
# inst/tinytest/test_data_integrity.R
library(tinytest)
source(here::here('R/load_data.R'))

dat <- load_raw_data()
```

☐ **Test: expected number of columns (12).** A mismatch signals that the upstream data export has changed or that the file was corrupted during transfer.

```
N_EXPECTED_COLS <- 12L
expect_equal(ncol(dat), N_EXPECTED_COLS)
```

☐ **Test: expected column names present.** A failure indicates the data provider has renamed a field or that an export script has changed; either way, the analysis cannot proceed without resolving it.

```
expected_cols <- c(
  'ParticipantID', 'Status', 'Sex', 'Group',
  'VisitAge', 'VisitType',
  'Outcome_A_Imp', 'Outcome_B_Imp',
  'Sub_a', 'Sub_b', 'Sub_c', 'Sub_total_Imp'
)
expect_true(all(expected_cols %in% names(dat)))
```

☐ **Test: ID columns have no NAs and follow expected format.** `ParticipantID` is integer-valued with no missing values; the documented range is 1001-9999. A missing identifier breaks every downstream join.

```
expect_true(all(!is.na(dat$ParticipantID)))
expect_true(is.numeric(dat$ParticipantID))
expect_true(all(dat$ParticipantID >= 1001 &
                  dat$ParticipantID <= 9999))
```

☐ **Test: pre-specified categorical variables match allowed levels.** Catches typos and miscoded values before they propagate into the model.

```
allowed_groups <- c('Active', 'Control')
expect_true(all(dat$Group %in% allowed_groups))

allowed_status <- c('Enrolled', 'Withdrawn',
                    'Completed')
expect_true(all(dat$Status %in% allowed_status))
```

☐ **Test: demographic categorical variables match coding scheme.** Investigate any unexpected values; do not silently recode them. Differences from the documented set are sometimes legitimate and sometimes errors; only the data provider can tell.

```
allowed_sex <- c('M', 'F')
expect_true(all(dat$Sex %in% allowed_sex))
expect_true(all(!is.na(dat$Sex)))
```

☐ **Test: visit variables fall in the pre-specified set.** P1 is the baseline assessment; V2-V4 are follow-ups. Any other value indicates a data export error or a silently extended protocol.

```
allowed_visits <- c('P1', 'V2', 'V3', 'V4')
expect_true(all(dat$VisitType %in% allowed_visits))
```

☐ **Test: numeric columns are within plausible ranges.** Use `na.rm = TRUE` so that missing follow-up data does not trip the check.

```
# Age (years)
expect_true(all(dat$VisitAge >= 18 &
                  dat$VisitAge <= 100,
                na.rm = TRUE))

# Likert subscale items (0-3)
for (col in c('Sub_a', 'Sub_b', 'Sub_c')) {
  x <- dat[[col]]
  expect_true(all(x >= 0 & x <= 3, na.rm = TRUE))
}

# Subscale composite (0-9)
expect_true(all(dat$Sub_total_Imp >= 0 &
                  dat$Sub_total_Imp <= 9,
                na.rm = TRUE))

# Outcomes (placeholder bounds; refer to dictionary)
expect_true(all(dat$Outcome_A_Imp >= 36 &
                  dat$Outcome_A_Imp <= 78,
                na.rm = TRUE))
expect_true(all(dat$Outcome_B_Imp >= 0 &
                  dat$Outcome_B_Imp <= 28,
                na.rm = TRUE))
```

☐ **Create `analysis/scripts/01-clean-data.R`.** The script sources `load_raw_data()`, applies the cleaning steps in Items 19-21, and writes the result to derived data (Item 22). Structure as a linear pipe so the transformations are auditable in sequence.

```
# analysis/scripts/01-clean-data.R
library(tidyverse)
library(janitor)
library(here)
source(here('R/load_data.R'))

dat_clean <- load_raw_data() |>
  clean_names() |>
  mutate(
    group = factor(group,
                   levels = c('Control', 'Active')),
    sex = factor(sex, levels = c('F', 'M')),
    visit_type = factor(
      visit_type,
      levels = c('P1', 'V2', 'V3', 'V4'),
      ordered = TRUE
    ),
    status = factor(status,
                    levels = c('Enrolled',
                               'Withdrawn',
```

```
                                'Completed'))
  )

saveRDS(
  dat_clean,
  here('analysis/data/derived_data/cleaned_data.rds')
)
```

☐ **Clean column names using `janitor::clean_names()`.** `clean_names()` converts headers to snake_case and normalises the BOM on the first column. Update the data dictionary to record both raw and cleaned names.

```
# Before clean_names():
#   ParticipantID, Outcome_A_Imp, Sub_total_Imp, ...

dat_clean <- dat |> janitor::clean_names()

# After clean_names():
#   participant_id, outcome_a_imp, sub_total_imp, ...
```

☐ **Convert categorical variables to factors with explicit levels.** Place the reference level first (`Control` before `Active`, `F` before `M`) so model contrasts produce intuitive coefficients. Preserve the time ordering for `visit_type` (`P1 < V2 < V3 < V4`).

```
dat_clean <- dat_clean |>
  mutate(
    # Control as reference => coefficient is the
    # 'Active vs Control' effect
    group = factor(group,
                   levels = c('Control', 'Active')),

    # Female as reference (alphabetical)
    sex = factor(sex, levels = c('F', 'M')),

    # Ordered factor preserves visit sequence
    visit_type = factor(
      visit_type,
      levels = c('P1', 'V2', 'V3', 'V4'),
      ordered = TRUE
    )
  )
```

☐ **Handle missing values explicitly and document approach.** Tabulate NAs per column and per visit before deciding on a strategy. The `_Imp` suffix suggests imputation was already done upstream; confirm with the data provider before writing any imputation yourself.

```
dat_clean |>
  summarise(across(everything(),
                   ~ sum(is.na(.x)))) |>
  pivot_longer(everything(),
```

```
               names_to = 'column',
               values_to = 'n_missing') |>
  arrange(desc(n_missing))

# Per-visit missingness for the primary outcome
dat_clean |>
  group_by(visit_type) |>
  summarise(
    n_total = n(),
    n_missing_a = sum(is.na(outcome_a_imp))
  )
```

☐ **Save cleaned data to `analysis/data/derived_data/`.** All downstream scripts read from derived data, never from raw. This makes the cleaning step the single point of transformation.

```
saveRDS(
  dat_clean,
  here::here(
    'analysis/data/derived_data/cleaned_data.rds'
  )
)

# Optional: also write a CSV for non-R consumers
readr::write_csv(
  dat_clean,
  here::here(
    'analysis/data/derived_data/cleaned_data.csv'
  )
)
```

Figure 2: Workspace ambiance: an analyst at the second day of a project, with the printed checklist on the desk and the loaded data frame on screen.

Phase 3: Exploratory Analysis (Items 23-31)

Phase 3 produces the first knittable report. The deliverables are `analysis/scripts/02-eda.R` and `analysis/report/01-eda.Rmd` (or `.qmd`).

- ☐ **Create `analysis/scripts/02-eda.R`.** The second step in the linear pipeline. It reads cleaned data from derived data, never from raw.

```
# analysis/scripts/02-eda.R
library(tidyverse)
library(skimr)
library(here)

dat <- readRDS(
  here('analysis/data/derived_data/cleaned_data.rds')
)
```

- ☐ **Generate summary statistics by treatment group.** For each numeric outcome and each pre-specified grouping, produce mean, SD, median, IQR, min, max, and count of non-missing observations. Save the summary as a CSV in derived data.

```r
summary_by_group_visit <- dat |>
  group_by(group, visit_type) |>
  summarise(
    across(
      c(outcome_a_imp, outcome_b_imp, sub_total_imp),
      list(
        n    = ~ sum(!is.na(.x)),
        mean = ~ mean(.x, na.rm = TRUE),
        sd   = ~ sd(.x, na.rm = TRUE),
        med  = ~ median(.x, na.rm = TRUE)
      ),
      .names = '{.col}_{.fn}'
    ),
    .groups = 'drop'
  )

readr::write_csv(
  summary_by_group_visit,
  here('analysis/data/derived_data/summary_by_group_visit.csv')
)
```

☐ **Check baseline balance.** For randomised trials, imbalance at baseline is informative for downstream covariate adjustment, not a basis for re-randomisation.

```r
baseline <- dat |> filter(visit_type == 'P1')

baseline |>
  group_by(group) |>
  summarise(
    n          = n(),
    mean_age   = mean(visit_age, na.rm = TRUE),
    sd_age     = sd(visit_age, na.rm = TRUE),
    pct_female = mean(sex == 'F') * 100,
    mean_a     = mean(outcome_a_imp, na.rm = TRUE),
    mean_b     = mean(outcome_b_imp, na.rm = TRUE)
  )
```

☐ **Visualise outcome distributions.** Inspect for floor and ceiling effects, bimodality, heavy tails, and outliers.

```r
p_dist <- ggplot(dat,
                 aes(x = outcome_a_imp,
                     fill = group)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~ visit_type) +
  labs(title = 'Outcome A by group and visit',
       x = 'Outcome A (imputed)',
       y = 'Density')
```

```
ggsave(
  here('analysis/figures/dist-outcome-a.png'),
  p_dist, width = 8, height = 5, dpi = 300
)
```

☐ **Identify potential outliers or data quality issues.** Flag observations that fall outside the plausibility ranges from Item 17, or that deviate substantially from the within-group distribution. Document each flagged row with the decision in a tracking file.

```
flags <- dat |>
  mutate(
    flag_age   = visit_age < 18 | visit_age > 100,
    flag_sub_a = sub_a < 0 | sub_a > 3,
    flag_sub_b = sub_b < 0 | sub_b > 3,
    flag_sub_c = sub_c < 0 | sub_c > 3
  ) |>
  filter(flag_age | flag_sub_a | flag_sub_b |
           flag_sub_c)

if (nrow(flags) > 0) {
  readr::write_csv(
    flags,
    here('analysis/data/derived_data/data-flags.csv')
  )
}
```

☐ **Create `analysis/report/01-eda.Rmd` (or `.qmd`).** The first report is a stand-alone Rmd or qmd that knits to HTML or PDF. It loads cleaned data, reads the EDA outputs from derived data, and presents the results in narrative form.

```
quarto render analysis/report/01-eda.qmd
```

☐ **Document sample size and missingness patterns.** Report the number of participants enrolled, randomised, and analysed at each visit. Tabulate missingness by variable and by visit so the reader can judge the analytic implications.

```
# Per-visit sample sizes by group
dat |>
  count(visit_type, group) |>
  pivot_wider(names_from = group, values_from = n)

# Per-visit missingness for the primary outcome
dat |>
  group_by(visit_type) |>
  summarise(
    n_total    = n(),
    n_observed = sum(!is.na(outcome_a_imp)),
    pct_obs    = 100 * n_observed / n_total
  )
```

☐ **Include baseline characteristics table (Table 1).** Use `gtsummary::tbl_summary()` stratified by `group`. For randomised trials, avoid statistical tests of baseline balance; they are uninformative when randomisation succeeded.

```
library(gtsummary)

tab1 <- dat |>
  filter(visit_type == 'P1') |>
  select(group, sex, visit_age,
         outcome_a_imp, outcome_b_imp,
         sub_total_imp) |>
  tbl_summary(
    by = group,
    statistic = list(
      all_continuous() ~ '{mean} ({sd})',
      all_categorical() ~ '{n} ({p}%)'
    )
  ) |>
  add_overall()

tab1
```

☐ **Include outcome trajectory plots by visit and group.** Plot the mean trajectory over `visit_type` with error bars; spaghetti plots of individual trajectories are a useful supplement when sample sizes are modest.

```
p_traj <- dat |>
  group_by(group, visit_type) |>
  summarise(
    m  = mean(outcome_a_imp, na.rm = TRUE),
    se = sd(outcome_a_imp, na.rm = TRUE) /
           sqrt(sum(!is.na(outcome_a_imp))),
    .groups = 'drop'
  ) |>
  ggplot(aes(x = visit_type, y = m,
             colour = group, group = group)) +
  geom_line() +
  geom_errorbar(aes(ymin = m - se, ymax = m + se),
                width = 0.1) +
  labs(x = 'Visit', y = 'Outcome A (mean +/- SE)')

ggsave(
  here('analysis/figures/trajectory-outcome-a.png'),
  p_traj, width = 6, height = 4, dpi = 300
)
```

Phase 4: Analysis Functions (Items 32-38)

Phase 4 produces tested R functions for summary, modelling, and plotting. The deliverables are `R/summarize_outcomes.R`, `R/model_outcomes.R`, `R/plot_outcomes.R`, and corresponding test files under `inst/tinytest/`.

☐ **Create `R/summarize_outcomes.R` with descriptive statistics functions.** Each function takes a data frame and returns a data frame, never a printed object. Document arguments and return values with `roxygen2`.

```
# R/summarize_outcomes.R

#' Mean and SD of an outcome by group and visit
#'
#' @param dat cleaned data frame
#' @param outcome column name as a string
#' @return tibble with group, visit_type, n, mean, sd
#' @export
summarize_outcome_by_group_visit <- function(dat,
                                             outcome) {
  stopifnot(outcome %in% names(dat))
  dat |>
    dplyr::group_by(group, visit_type) |>
    dplyr::summarise(
      n    = sum(!is.na(.data[[outcome]])),
      mean = mean(.data[[outcome]], na.rm = TRUE),
      sd   = sd(.data[[outcome]], na.rm = TRUE),
      .groups = 'drop'
    )
}
```

☐ **Create `R/model_outcomes.R` with statistical modelling functions.** Wrap each call to `lm()`, `lmer()`, or `glm()` in a function so the model specification is captured in code rather than reproduced inline.

```
# R/model_outcomes.R

#' Fit a linear mixed model for a primary outcome
#'
#' Random intercept for participant; fixed effects of
#' group, visit, their interaction, and baseline
#' covariates.
#'
#' @param dat cleaned data frame
#' @param outcome column name as a string
#' @return fitted lmerMod object
#' @export
fit_primary_lmm <- function(dat, outcome) {
  stopifnot(outcome %in% names(dat))
```

```
  f <- stats::as.formula(
    paste0(
      outcome,
      ' ~ group * visit_type + visit_age + sex + ',
      '(1 | participant_id)'
    )
  )
  lme4::lmer(f, data = dat)
}
```

☐ **Create `R/plot_outcomes.R` with visualisation functions.** Plotting functions return a ggplot; they do not call `ggsave()`. Returning the object lets reports compose panels with `patchwork`.

```
# R/plot_outcomes.R

#' Mean +/- SE trajectory plot for one outcome
#'
#' @param dat cleaned data frame
#' @param outcome column name as a string
#' @return ggplot object
#' @export
plot_trajectory <- function(dat, outcome) {
  assertthat::assert_that(outcome %in% names(dat))
  assertthat::assert_that('group' %in% names(dat))
  assertthat::assert_that('visit_type' %in% names(dat))

  dat |>
    ggplot2::ggplot(
      ggplot2::aes(x = visit_type,
                   y = .data[[outcome]],
                   colour = group, group = group)) +
    ggplot2::stat_summary(fun = mean,
                          geom = 'line') +
    ggplot2::stat_summary(fun.data = mean_se,
                          geom = 'errorbar',
                          width = 0.1)
}
```

☐ **Create `inst/tinytest/test_summarize.R`.** Each test uses a small hand-constructed data frame with known summary statistics.

```
# inst/tinytest/test_summarize.R
library(tinytest)
source(here::here('R/summarize_outcomes.R'))

tiny <- tibble::tribble(
  ~participant_id, ~group, ~visit_type, ~outcome_a_imp,
  1001L, 'Active',  'P1', 5.0,
```

```
  1002L, 'Active',  'P1', 7.0,
  1003L, 'Control', 'P1', 4.0,
  1004L, 'Control', 'P1', 6.0
)

out <- summarize_outcome_by_group_visit(
  tiny, 'outcome_a_imp')
expect_equal(out$mean[out$group == 'Active'],  6.0)
expect_equal(out$mean[out$group == 'Control'], 5.0)
expect_equal(nrow(out), 2L)
```

☐ **Create `inst/tinytest/test_model.R`.** Test modelling functions against simulated data with known parameters. Use `set.seed()` for reproducibility.

```
# inst/tinytest/test_model.R
library(tinytest)
library(lme4)
source(here::here('R/model_outcomes.R'))

set.seed(42)
n <- 200
sim <- tibble::tibble(
  participant_id = rep(1:(n / 2), each = 2),
  group = factor(
    rep(c('Active', 'Control'), each = n / 2)),
  visit_type = factor(
    rep(c('P1', 'V2'), times = n / 2)),
  visit_age = rnorm(n, 50, 10),
  sex = factor(sample(c('M', 'F'), n, replace = TRUE))
)
# True effect of Active vs Control = 2.0
sim$outcome_a_imp <- 5 +
  2 * (sim$group == 'Active') + rnorm(n, 0, 1)

fit <- fit_primary_lmm(sim, 'outcome_a_imp')
est <- lme4::fixef(fit)[['groupActive']]
expect_true(abs(est - 2) < 0.5)
```

☐ **Test edge cases: missing data handling.** The `_Imp` columns should not have NAs, but their unimputed counterparts might. Lock down the documented behaviour with a test.

```
tiny_with_na <- tiny
tiny_with_na$outcome_a_imp[1] <- NA_real_

out <- summarize_outcome_by_group_visit(
  tiny_with_na, 'outcome_a_imp')
# Active row: n drops from 2 to 1, mean from 6 to 7
expect_equal(out$n[out$group == 'Active'],    1L)
expect_equal(out$mean[out$group == 'Active'], 7.0)
```

☐ **Test edge cases: single-group scenarios.** A pre-specified subgroup may turn out empty in the sample. Assert either a graceful empty result or a clear error.

```
tiny_single <- tiny |>
  dplyr::filter(group == 'Active')

out <- summarize_outcome_by_group_visit(
  tiny_single, 'outcome_a_imp')
expect_equal(nrow(out), 1L)
expect_equal(out$group[1], 'Active')
```

Phase 5: Primary Analysis (Items 39-45)

Phase 5 produces the primary-analysis report, including secondary outcomes and sensitivity analyses. The deliverables are `analysis/scripts/03-primary-analysis.R`, saved fitted-model objects in `analysis/data/derived_data/`, and `analysis/report/02-results.Rmd` (or `.qmd`).

☐ **Create `analysis/scripts/03-primary-analysis.R`.** The script orchestrates; `R/` holds the reusable code.

```
# analysis/scripts/03-primary-analysis.R
library(here)
library(broom.mixed)
source(here('R/model_outcomes.R'))

dat <- readRDS(
  here('analysis/data/derived_data/cleaned_data.rds')
)

fit_a <- fit_primary_lmm(dat, 'outcome_a_imp')
fit_b <- fit_primary_lmm(dat, 'outcome_b_imp')
fit_s <- fit_primary_lmm(dat, 'sub_total_imp')
```

☐ **Implement the pre-specified analysis plan.** Follow the SAP line by line. Mark any deviation from the SAP with a code comment that begins `# SAP DEVIATION:` and document the reason.

```
# SAP Section 4.1: primary contrast 'Active vs Control'
# at the final follow-up visit (V4)
primary_contrast_a <- emmeans::emmeans(
  fit_a,
  pairwise ~ group | visit_type
)$contrasts |>
  as.data.frame() |>
  dplyr::filter(visit_type == 'V4')

# SAP DEVIATION: V3 contrast was added post-hoc at
# the request of the DSMB; report as exploratory.
```

☐ **Save model objects to `analysis/data/derived_data/`.** Persist each fitted model with `saveRDS()`; save the tidy and glance outputs as CSV for direct inclusion in tables. The derived data folder becomes the single source of truth for model results.

```
saveRDS(
  fit_a,
  here('analysis/data/derived_data/model_outcome_a_lmm.rds')
)
saveRDS(
  fit_b,
  here('analysis/data/derived_data/model_outcome_b_lmm.rds')
)

readr::write_csv(
  broom.mixed::tidy(fit_a),
  here('analysis/data/derived_data/tidy_outcome_a.csv')
)
readr::write_csv(
  broom.mixed::glance(fit_a),
  here('analysis/data/derived_data/glance_outcome_a.csv')
)
```

☐ **Create `analysis/report/02-results.Rmd` (or `.qmd`).** The results report loads the saved model objects and renders the result tables and figures. It contains no model-fitting code; if it does, the script and report can drift apart.

```
# In the report's R chunk, NOT in the script
fit_a <- readRDS(here::here(
  'analysis/data/derived_data/model_outcome_a_lmm.rds'))
tidy_a <- readr::read_csv(here::here(
  'analysis/data/derived_data/tidy_outcome_a.csv'))

knitr::kable(tidy_a, digits = 3)
```

☐ **Document primary outcome analysis in the report.** Present the primary outcome model first, with a results paragraph that includes the point estimate, confidence interval, and inferential statistic for the pre-specified primary contrast.

A typical results sentence template:

```
The mean Outcome A at V4 was lower in the Active
arm than in the Control arm (estimated difference
-3.2 points, 95% CI -5.1 to -1.3, p = 0.001),
consistent with the pre-specified primary
hypothesis. The fitted model was a linear mixed
model with a random intercept per participant and
fixed effects for group, visit, their interaction,
and baseline covariates (age, sex):

`outcome_a_imp ~ group * visit_type + visit_age +
```

```
sex + (1 | participant_id)`
```

☐ **Document secondary outcomes in the report.** Present each secondary outcome with the same structure as the primary. Acknowledge the multiple-comparison context; either pre-specify a correction or label the analyses as exploratory.

```
# Holm-adjusted p-values across the three primary
# contrasts (Outcome A, Outcome B, Sub_total)
p_unadj <- c(
  outcome_a    = 0.001,
  outcome_b    = 0.04,
  sub_total    = 0.18
)
p_holm <- p.adjust(p_unadj, method = 'holm')
p_holm
```

☐ **Document sensitivity analyses in the report.** For each modelling assumption that could plausibly affect the conclusion, run a sensitivity analysis and document its result alongside the primary.

```
# Sensitivity 1: complete-case using the unimputed
# underlying column (if available from the data
# provider) instead of the _Imp column.
fit_a_cc <- fit_primary_lmm(dat, 'outcome_a_raw')

# Sensitivity 2: alternative covariate set (drop
# baseline age)
f_alt <- as.formula(
  'outcome_a_imp ~ group * visit_type + sex + ',
  '(1 | participant_id)')
fit_a_alt <- lme4::lmer(f_alt, data = dat)

# Compare estimates from primary vs sensitivity
dplyr::bind_rows(
  primary    = broom.mixed::tidy(fit_a),
  complete   = broom.mixed::tidy(fit_a_cc),
  no_age     = broom.mixed::tidy(fit_a_alt),
  .id = 'analysis'
) |>
  dplyr::filter(term == 'groupActive')
```

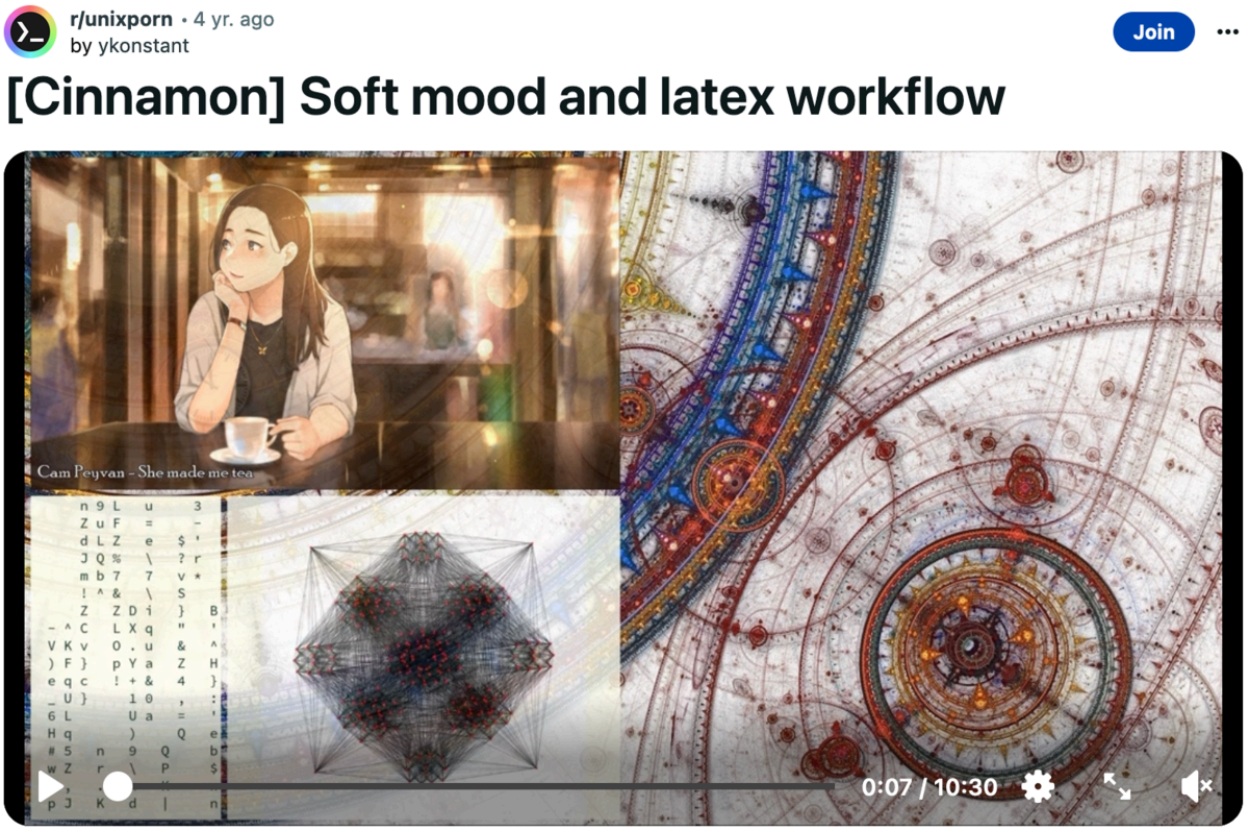

Figure 3: Editor view of the checklist artefact open alongside the working zzcollab project, showing how items are referenced by number from commit messages and TODO notes.

Phase 6: Documentation and Reproducibility (Items 46-55)

Phase 6 closes the loop: DESCRIPTION metadata, a project README, a rebuilt Docker image, and a fresh-container end-to-end reproducibility run.

- ☐ **Update `DESCRIPTION`: add a meaningful Title and Description.** Title is one sentence, fewer than sixty-five characters. Description is one to three sentences naming the study population, primary outcomes, and analytic approach.

```
Package: studya
Title: Study A: Active vs Control on Outcomes A and B
Version: 0.0.0.9000
Authors@R:
    person('First', 'Last', email = 'you@example.com',
           role = c('aut', 'cre'))
Description: Reproducible compendium for Study A, an
    IRB-approved randomised comparison of Active and
    Control on Outcomes A and B across four study
    visits (P1 baseline, V2-V4 follow-ups). Linear
    mixed models with random participant intercepts.
```

☐ **Update `DESCRIPTION`: add or verify Authors.** Use the Authors@R field with `person()` entries; do not silently leave the placeholder 'Your Name'.

```
Authors@R: c(
    person('Anna', 'Analyst',
           email = 'analyst@example.org',
           role = c('aut', 'cre'),
           comment = c(ORCID = '0000-0000-0000-0000')),
    person('Bob', 'Bioscientist',
           email = 'bob@example.org',
           role = 'ctb'))
```

☐ **Verify all package dependencies listed in `DESCRIPTION`.** Any package called but not declared will fail in a clean container. DESCRIPTION lists direct dependencies only; `renv.lock` pins the transitive closure.

```
# Inside the container
deps <- renv::dependencies()
unique(deps$Package) |> sort()
```

```
make check-renv          # validates Imports vs code
```

☐ **Create or update the project `README.md`.** The README is the entry point for anyone reading the repository for the first time. Include a directory tree (`tree -L 2`), the research question, and the reproduction commands.

```
tree -L 2 -I 'renv|_freeze|_site' .
```

☐ **Document reproduction instructions in the README.** Provide a copy-pasteable sequence of shell commands that reproduces the analysis from a clean checkout. Test on a machine that does not already have the project set up.

```
# Reproduce Study A end-to-end
git clone https://github.com/<owner>/studya.git
cd studya
make docker-build

# Inside the container, run the linear pipeline
make r
```

```
source('analysis/scripts/01-clean-data.R')
source('analysis/scripts/02-eda.R')
source('analysis/scripts/03-primary-analysis.R')
quarto::quarto_render('analysis/report/01-eda.qmd')
quarto::quarto_render('analysis/report/02-results.qmd')
```

☐ **Rebuild Docker image: `make docker-build`.** Tag with the current git SHA so future readers can identify the exact build that produced the published results.

```
make docker-build
GIT_SHA=$(git rev-parse --short HEAD)
```

```
docker tag studya:latest studya:${GIT_SHA}
docker images studya
```

☐ **Verify reproducibility: run all scripts in a fresh container.** The end-to-end run is the canonical reproducibility test.

```
GIT_SHA=$(git rev-parse --short HEAD)

docker run --rm -v "$(pwd):/project" -w /project \
  studya:${GIT_SHA} \
  Rscript analysis/scripts/01-clean-data.R

docker run --rm -v "$(pwd):/project" -w /project \
  studya:${GIT_SHA} \
  Rscript analysis/scripts/02-eda.R

docker run --rm -v "$(pwd):/project" -w /project \
  studya:${GIT_SHA} \
  Rscript analysis/scripts/03-primary-analysis.R

docker run --rm -v "$(pwd):/project" -w /project \
  studya:${GIT_SHA} \
  quarto render analysis/report/02-results.qmd
```

☐ **Confirm all outputs match expected results.** Bit-for-bit reproducibility is a strong signal; numerically identical floating-point results without timestamp or render-metadata noise is the realistic target.

```
# Hash the derived data and figures from the fresh
# run, compare against the previously committed
# versions
git stash
shasum -a 256 analysis/data/derived_data/*.rds \
              analysis/figures/*.png > /tmp/fresh.sha256
git stash pop
shasum -a 256 analysis/data/derived_data/*.rds \
              analysis/figures/*.png > /tmp/committed.sha256
diff /tmp/fresh.sha256 /tmp/committed.sha256
```

☐ **Run `make test` and verify all tests pass.** A failing test at the publication stage is a stop-the-line event.

```
make test
# Equivalent inside R:
# tinytest::run_test_dir('inst/tinytest')
# testthat::test_local()
```

☐ **Final review: all Five Pillars complete and consistent.** Walk through each pillar one last time and confirm that the artefacts on disk tell a single, consistent story.

```
# Pillar 1: Dockerfile R version
grep '^FROM' Dockerfile

# Pillar 2: renv.lock R version
jq '.R.Version' renv.lock

# Pillar 3: .Rprofile activates renv
grep -E 'renv/activate' .Rprofile

# Pillar 4: source code is sourceable
Rscript -e 'devtools::load_all()'

# Pillar 5: raw data is read-only and documented
ls -la analysis/data/raw_data/
test -f analysis/data/README.md && \
  echo 'data README present'
test -f docs/data-dictionary.md && \
  echo 'data dictionary present'
```

The grep, jq, and ls outputs together must agree: same R version in the Dockerfile and renv.lock, renv activation in .Rprofile, sourceable code, and read-only documented raw data.

Daily Workflow

The checklist is not consulted line by line every day. It is consulted at three moments:

1. **Project kick-off.** Read the entire checklist before doing anything else. Skim, do not implement; the goal is to know what is coming so the early decisions (directory layout, factor coding, file naming) do not need to be unwound later.
2. **Phase boundaries.** At the end of each phase, work through the checklist items for that phase explicitly. Tick each item off in a private copy (`docs/analysis-checklist-<project>.md` is a reasonable convention) so the project's progress is auditable.
3. **Reproducibility gate.** Before circulating any externally facing report, run Phase 6 in full. This is the gate that catches the inconsistency between the author's laptop and a fresh container.

The numbered structure is also useful in commit messages ('Item 17: add plausibility-bound assertions for outcomes A and B') and in TODO notes ('Item 41: still need to save fitted models as RDS').

Command	Action
`quarto render docs/analysis-checklist.qmd`	Render the standalone checklist
`make r`	Enter the Docker container
`make test`	Run the tinytest and testthat suites
`make check-renv`	Validate package dependencies
`make docker-build`	Rebuild the image after package changes

Things to Watch Out For

1. **The checklist is a tool, not a contract.** Items are meant to be skipped consciously, not silently. If the project has no longitudinal structure, Item 16 (visit variables) is not applicable; mark it 'N/A' with a one-line reason rather than removing it from the file.
2. **Phase 1 takes longer than expected.** First-time users of zzcollab routinely underestimate Item 6 (data provenance README) and Item 7 (data dictionary). Both feel like overhead and pay off only weeks later, when they are the difference between a clean Phase 2 and a month of detective work. The `_Imp` ambiguity in the example dataset is a concrete instance: resolving it in Item 7 takes ten minutes; discovering it in Phase 5 takes days.
3. **The integrity tests are not unit tests.** Items 11-17 assert contracts about the raw data, not behaviours of functions. They live in `inst/tinytest/`, not `tests/testthat/`, by convention. Mixing the two leads to tests that pass against frozen fixtures but fail against live data.
4. **Item 41 (save model objects) is easy to forget.** Without saved RDS objects, the report in Phase 5 re-fits models on every render, which couples the report to a particular package version and can produce different results across renders. Save once, load thereafter.
5. **Item 52 (fresh-container rerun) catches errors that nothing else does.** A pipeline that runs on the author's laptop with a stale R session can fail in a fresh container because of a missing `library()` call, a hardcoded path, or a package not declared in DESCRIPTION. Run Item 52 weekly during active analysis, not just at the end.
6. **Item 55 is not a formality.** The Five Pillars review has caught real inconsistencies (mismatched R versions, undeclared dependencies, raw data committed under the wrong path) on every project that used it. Treat it as a separate review pass, with at least one fresh pair of eyes.

When the Updated Extract Arrives

The email mentioned that 'an updated extract may follow if a few late withdrawals come in'. This is the moment the Phase 2 tests earn their keep. The procedure is four commands:

```
# 1. Drop the new file in and lock it down.
mv ~/Downloads/studyA_data_2026Q2.csv \
   analysis/data/raw_data/
chmod 444 analysis/data/raw_data/studyA_data_2026Q2.csv

# 2. Update the data README with the new receipt date
#    and the provider's notes on what changed.
$EDITOR analysis/data/README.md

# 3. Update load_raw_data() to point at the new file
#    (or accept a filename argument), then run the
#    integrity tests in the container.
make r
```

```
# 4. Inside R: run integrity tests, then re-source
#    the pipeline if they pass.
tinytest::run_test_file(
  'inst/tinytest/test_data_integrity.R')
# All passed -> safe to re-run the pipeline.

source('analysis/scripts/01-clean-data.R')
source('analysis/scripts/02-eda.R')
source('analysis/scripts/03-primary-analysis.R')

quarto::quarto_render('analysis/report/01-eda.qmd')
quarto::quarto_render('analysis/report/02-results.qmd')
```

If Items 11-17 pass silently, the contract held and the data are just smaller or larger. If any of them fails loudly, the contract changed; investigate before proceeding. A pipeline without integrity tests requires the analyst to re-derive every assumption about the new file by hand; a pipeline with the Phase 2 tests in place reduces the update to a build-and-render cycle.

Figure 4: A finished compendium archived alongside the rendered report, with the standalone checklist filed in docs/.

What Did We Learn?

Lessons Learnt

Conceptual Understanding:

- A checklist is a tool for surfacing the steps that are easy to skip, not for enforcing a particular workflow. The same checklist will produce slightly different artefacts in two projects; that is the intended outcome.
- The phase boundaries are useful precisely because they are arbitrary. Anything that lets a project pause and audit itself before moving on is worth the friction.
- The numbered structure is more valuable than the prose. The annotation paragraphs are scaffolding for first-time use; experienced users only consult the numbers.
- Reproducibility is a property of the workspace as a whole, not of any single artefact. The Five Pillars review at Item 55 is the only point that audits the workspace rather than its parts.

Technical Skills:

- Quarto is a better authoring surface for the checklist than LaTeX. The same source renders to HTML for inline reading and PDF for printing, and the document carries a YAML header that the analyst can edit per project.
- A standalone `docs/analysis-checklist.qmd` lives outside the analysis pipeline, so changes to the checklist do not retrigger expensive analysis re-renders.
- `tinytest` in `inst/tinytest/` is well suited to the data-integrity tests in Items 11-17. Its lightweight API is appropriate for assertions about a single loaded data frame.
- Saving fitted model objects as RDS in `analysis/data/derived_data/` decouples Phase 5 from the report rendering and makes report regeneration cheap.

Gotchas and Pitfalls:

- Forgetting to set raw data files read-only (Item 5) produces silent corruption that is impossible to detect without a checksum.
- Conflating the data dictionary (a contract) with the cleaned data documentation (a description) blurs the audit trail. Keep them separate.
- Running tinytest from the host instead of from inside the container exposes the tests to whatever R version the host happens to have, defeating the purpose of the zzcollab environment.
- A passing `make test` does not certify reproducibility; only Item 52 (fresh-container rerun) does. Several projects in the wild conflate the two.

Limitations

- **Six phases is one ordering, not the ordering.** A simulation study or a methods paper will want to reorder Phases 3 and 4. The checklist accommodates this by numbering items, not by enforcing serial execution.
- **The checklist assumes a tabular outcome.** Imaging pipelines, NLP corpora, and high-throughput genomics will need different Phase 2 contents.

- **No version-control discipline.** The checklist does not prescribe a git branching model, commit message format, or PR review workflow. Those belong in a separate document.
- **No statistical analysis plan template.** Item 40 says 'follow the SAP'; it does not provide one. Drafting the SAP is the analyst's responsibility.
- **No coverage of stakeholder-facing artefacts.** The checklist ends at Item 55 (reproducible compendium). It says nothing about manuscript drafting, conference posters, or oral presentations.

Opportunities for Improvement

Several avenues would be worth exploring further:

1. Add a Phase 0 with project-charter items (research question, target population, success criteria) that precede the workspace creation in Item 1.
2. Add a 'completion column' to the rendered checklist with a date and analyst initials, so the artefact doubles as a sign-off log.
3. Extend Phase 6 with archival items (DOI assignment, Zenodo deposit, OSF preregistration) for projects that require external archival of the compendium.
4. Provide a tinytest harness that takes the checklist as input and asserts each item's existence (`tests/integration/test-checklist-coverage.R`).
5. Translate the checklist into a Quarto project template that initialises an empty `docs/analysis-checklist.qmd` alongside the boilerplate `Dockerfile`, `Makefile`, `DESCRIPTION` produced by `zzc analysis`.

Wrapping Up

A 55-item checklist is enough structure to take a single CSV in the inbox to a publishable, reproducible compendium, and short enough to read end to end before doing anything. The six phases correspond to deliverables that map directly onto the zzcollab workspace: a buildable container, a validated dataset, an EDA report, tested functions, a primary-analysis report, and an archival compendium.

The most valuable lesson from running the checklist against the example dataset (a single CSV, no codebook, an updated extract on the way) was that Phase 1 and Phase 2 absorb most of the apparent overhead and produce most of the durable value. By the time Phase 3 begins, the analysis runs against a typed, tested, documented dataset, and the rest of the project is a series of auditable transformations on it.

In conclusion, four points merit emphasis. First, a six-phase, 55-item checklist drives a zzcollab analysis from a CSV in the inbox to a published, reproducible report. Second, phase boundaries are the natural audit points; Phase 6 is the only phase that audits the workspace as a whole. Third, the Phase 2 integrity tests pay off when the updated extract arrives. Fourth, the numbered structure is the durable feature; the prose is scaffolding for first-time users.

See Also

Related posts:

- Setup Post Template (worked example: AWS CLI provisioning): the post-47 template that this workflow post adapts.
- Palmer Penguins, Part 1: EDA and Simple Regression: a worked zzcollab analysis that exercises Phases 1 through 5 on a small public dataset.

Key resources:

- zzcollab framework: the Docker-based research compendium framework that this checklist is written against.
- rrtools: an R package for writing reproducible research compendia: the rrtools convention that the zzcollab layout extends.
- The Turing Way: Guide for Reproducible Research: a broader treatment of reproducibility practice.
- Quarto documentation: authoring reference for the standalone checklist artefact.

Reproducibility

Tested configuration:

Component	Version
Operating system	macOS 15.4
zzcollab	2.4.0
R	4.5.1
Quarto	1.5.x
renv	1.0.x
Docker Desktop	4.x
Last verified	2026-04-29

Configuration files:

- `docs/analysis-checklist.qmd`: the standalone generic checklist, archived alongside this post for offline reference.
- `Dockerfile`, `renv.lock`, `.Rprofile`: the Five Pillars computational environment shared with every zzcollab workspace.
- `Makefile`: standard zzcollab targets (`r`, `test`, `check-renv`, `docker-build`).

To reproduce end-to-end:

```
git clone <this-post-repo>
cd zzcollab-analysis-checklist
make docker-build
quarto render index.qmd --to pdf
```

Appendix A: Printable Reference Card

The annotated walkthrough above is the durable teaching content. The compact table below is the operational artefact, intended to be printed as a single sheet and ticked off in pen as a project progresses. The PDF render ships a fully-gridded version with cell borders; the HTML render uses the standard browser table.

#	Item	Done	Date completed	Notes
Phase 1: Project Setup				
1	Dockerfile exists and builds			
2	renv.lock lists required packages			
3	.Rprofile activates renv			
4	Source code directories exist			
5	Raw data is read-only in raw_data/			
6	analysis/data/README.md documents provenance			
7	Data dictionary written			
8	Analysis packages installed and snapshotted			
Phase 2: Data Ingestion and Validation				
9	R/load_data.R defines load_raw_data()			
10	inst/tinytest/test_data_integrity.R created			
11	Test: expected number of columns			
12	Test: expected column names			
13	Test: ID columns no NAs, expected format			
14	Test: pre-specified categorical levels match			
15	Test: demographic categorical levels match			
16	Test: visit variables in pre-specified set			
17	Test: numeric ranges plausible			
18	analysis/scripts/01-clean-data.R written			
19	janitor::clean_names() applied			
20	Categorical variables converted to factors			
21	Missing values handled and documented			
22	Cleaned data saved to derived_data/			
Phase 3: Exploratory Analysis				
23	analysis/scripts/02-eda.R written			
24	Summary statistics by group computed			
25	Baseline balance checked			
26	Outcome distributions visualised			
27	Outliers and data-quality flags documented			
28	analysis/report/01-eda.qmd written			
29	Sample size and missingness documented			
30	Table 1 included			
31	Trajectory plots included			
Phase 4: Analysis Functions				
32	R/summarize_outcomes.R written			
33	R/model_outcomes.R written			
34	R/plot_outcomes.R written			
35	inst/tinytest/test_summarize.R passes			
36	inst/tinytest/test_model.R passes			
37	Missing-data edge cases tested			
38	Single-group edge cases tested			
Phase 5: Primary Analysis				
39	analysis/scripts/03-primary-analysis.R written			
40	Pre-specified analysis plan implemented			
41	Fitted model objects saved to derived_data/			
42	analysis/report/02-results.qmd written			
43	Primary outcome documented			
44	Secondary outcomes documented			
45	Sensitivity analyses documented			
Phase 6: Documentation and Reproducibility				
46	DESCRIPTION Title and Description updated			
47	DESCRIPTION Authors set			

#	Item	Done	Date completed	Notes
48	DESCRIPTION Imports verified			
49	Project README.md created or updated			
50	Reproduction instructions documented			
51	Docker image rebuilt and tagged with git SHA			
52	Fresh-container end-to-end run passes			
53	Outputs match committed versions			
54	make test passes in fresh container			
55	Five Pillars consistency review complete			

A two-up print on letter or A4 stationery yields a single double-sided card. Items marked ‘N/A’ for the project at hand should carry a brief reason in the Notes column rather than a tick.

Let's Connect

Have questions, suggestions, or spot an error? Let me know.

- **GitHub:** rgt47
- **Twitter/X:** @rgt47
- **LinkedIn:** Ronald Glenn Thomas
- **Email:** Contact form

I would enjoy hearing from you if:

- You spot an error or a stronger ordering for any phase in the checklist.
- You have suggestions for items that should be added (or removed).
- You have used a similar checklist in a different research domain and want to compare notes.
- You just want to say hello and connect.

Rendered on 2026-04-30 at 08:06 PDT. Source: ~/Dropbox/prj/qblog/posts/61-zzcollab-analysis-checklist/zzcollab-analysis-checklist/analysis/report/index.qmd

Related posts in this cluster

This post is part of the *ZZCOLLAB Reproducible Compendia* series. Recommended reading order:

1. Post 01: Reproducible Blog Posts with ZZCOLLAB
2. Post 02: Constructing a reproducible blog post using zzcollab tools
3. Post 03: From Markdown to Blog Post: A ZZCOLLAB workflow
4. Post 04: Sharing R Code via Docker: R Markdown Reports
5. **Post 05: A 55-Item Initiation Checklist for zzcollab Data Analyses** (this post)
6. Post 06: Seven Required Elements for a zzc Manuscript report.Rmd
7. Post 07: A tiered CI strategy for zzcollab research compendia
8. Post 08: GitHub Actions workflows for zzcollab research compendia