Intelligent R-Driven Data Hygiene: Unified Automation for Detecting Hidden Characters and Optimizing Character Lengths for Efficient SAS Workflows

Ram Mohan Konda

Navitas Life Sciences

18 APRIL 2026

Agenda

- Introduction & Unified R–SAS Workflow Overview
- Why Automation in Clinical Programming?
- Case Study 1: Intelligent Risk Detection (Dry Run 1)
- Case Study 2: Impact of Automation (Dry Run 2)
- Automation Architecture: Special Character Detection
- Special Character Detection: Processing, Reporting & Exception Handling
- Automation Architecture: Character Length Optimization
- Character Length Optimization: Processing, Reporting & Validation
- Key Results, Reusability, Challenges & Future Directions

Overview, Objectives, and Unified R-SAS Workflow

connect·share·advance

Overview

📁 Automated R framework integrated with SAS to detect special characters and optimize character lengths in clinical datasets to improve data quality and performance.

Objectives

🔍 Detect hidden special characters
✅ Improve data quality
📏 Optimize variable lengths
⚡ Reduce dataset size and processing time

Unified R-SAS Workflow

🤝 **Hybrid Approach:** Combines R's memory efficiency with SAS's regulatory-standard data management.
📥 **Direct Ingestion:** R reads .sas7bdat files directly via haven for "in-place" auditing.
⚡ **High-Speed Audit:** Instantly calculates "True Max Length" and detects hidden characters across the entire library.

Why Automation?

connect·share·advance

- Faster Processing
- Consistency
- Error Reduction
- Improved Data Quality
- Time Efficiency
- Optimized Storage
- Better Monitoring
- Regulatory Readiness
- Scalability

Case Study – Dry Run 1
Intelligent Risk Detection

connect·share·advance

Visual Flow : SAS Datasets → R Hygiene Engine → Risks Identified

Input
- 32 Phase III datasets
- Pre-XPT audit

R Hygiene Engine
- Automated R scan
- Hidden character detection
- True length profiling

Findings
- 12 hidden character issues
- 360 length violations
- Submission risk

Key Message
- Submission-blocking issues detected before XPT generation

Case Study – Dry Run 2
The Impact of Automation

connect·share·advance

Before vs After Automation

Manual Baseline	Automated R–SAS Engine
~180 mins review time	< 60 seconds (batch)
Risk of garbled text	0% hidden characters
1.2 GB storage	~840 MB (30% saved)
Manual reconciliation	Fully audit-ready

Key Impact

- **180× Faster Execution**
- **360 → 0 Length Violations**
- **30% Storage Reduction**
- **100% Audit-Ready Outputs**

Outcome

- **All risks resolved with zero data loss using automated “True Max Length” mapping**

Macro Architecture Overview - Special Character Detection

connect·share·advance

Read SAS Datasets into R

Identify Character Variables

Extract Unique Values

Split Values into Characters

Evaluate ASCII/Unicode Codes

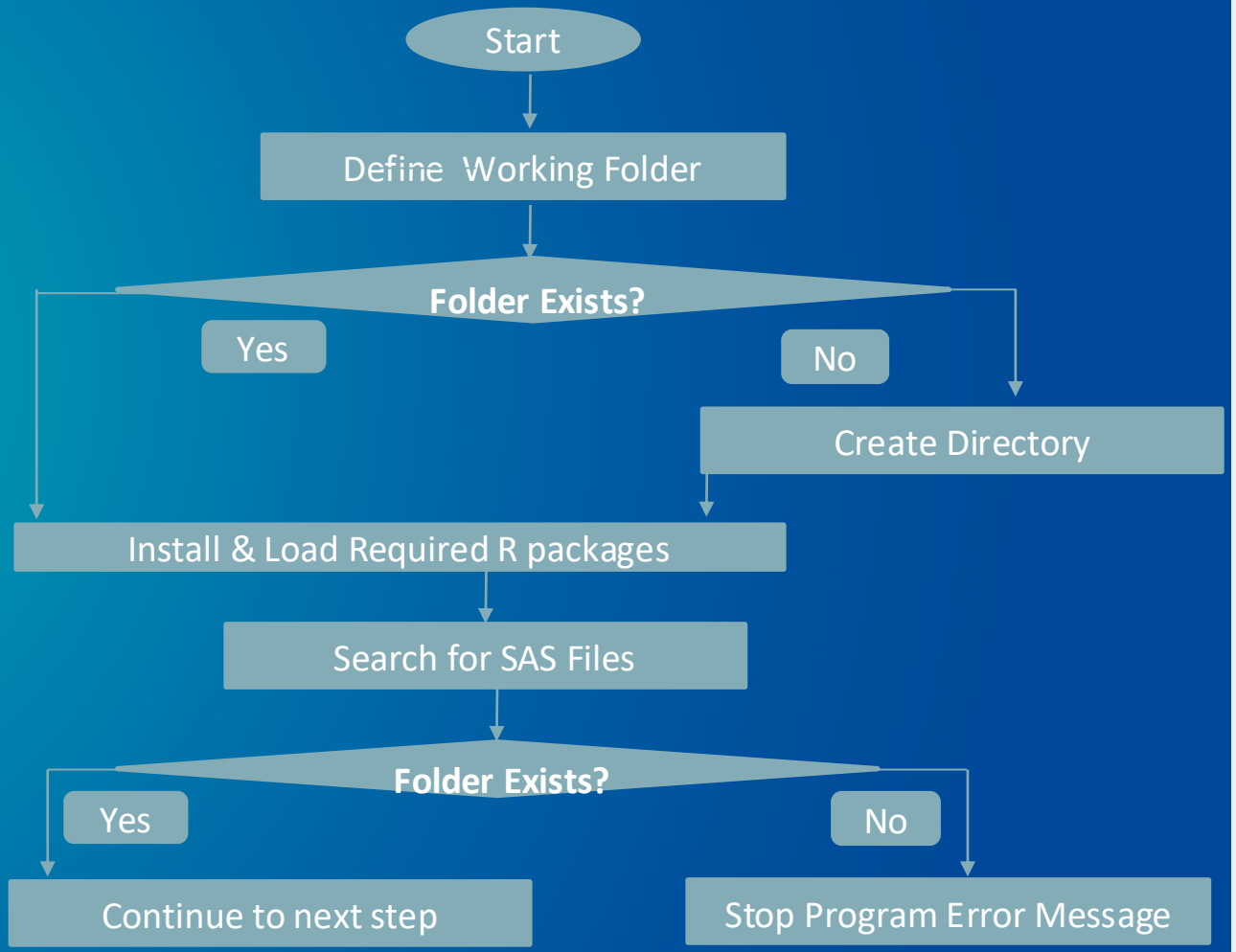
Detect Special/Non-Printable Characters

Capture Character Positions

Create Dataset-Level Results

Export Special Character Report

Environment Setup and SAS Dataset Import

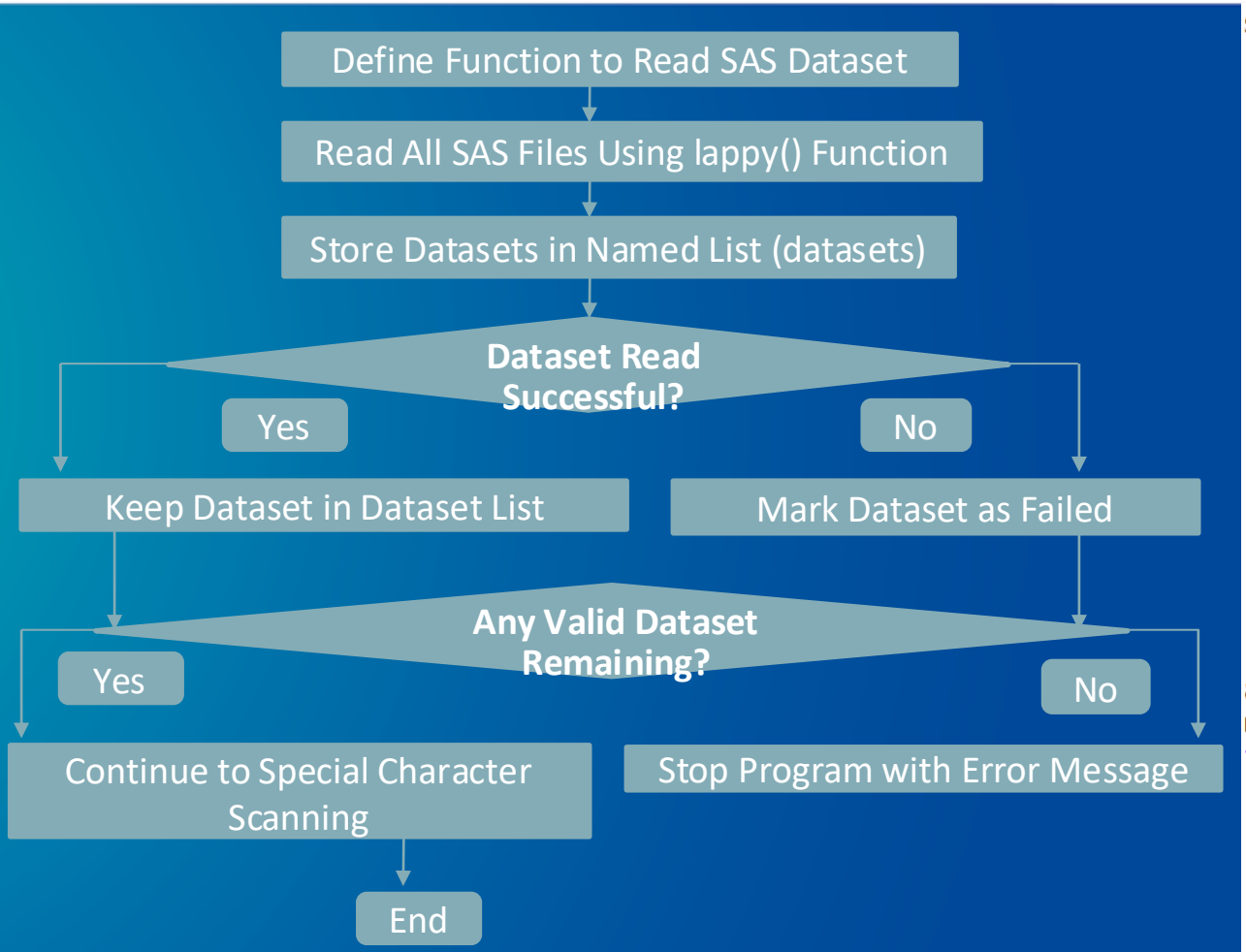



```r
# ---- Setup ----
ss <- "C:/Users/rammohan.konda/OneDrive - Navitas Life Sciences Pvt. Ltd/R-training"
if (!dir.exists(ss)) dir.create(ss, recursive = TRUE)
pkgs <- c("haven","dplyr","tidyr","stringi","readr","tools","purrr")
inst <- pkgs[!sapply(pkgs, requireNamespace, quietly = TRUE)]
if(length(inst)) install.packages(inst, dependencies = TRUE)
invisible(lapply(pkgs, library, character.only = TRUE))

# ---- Read SAS datasets ----
sas_files <- list.files(ss, pattern="\\.sas7bdat$", ignore.case=TRUE, full.names=TRUE)
if(!length(sas_files)) stop("No .sas7bdat files found in: ", ss)
read_one <- function(p){
  nm <- tools::file_path_sans_ext(basename(p))
  cat("Reading:", nm, "\n")
  tryCatch(    haven::read_sas(p, encoding="UTF-8"),
               error=function(e){message(" ! Failed: ", e$message); NULL}  )}
datasets <- setNames(lapply(sas_files, read_one),
                     tools::file_path_sans_ext(basename(sas_files)))
datasets <- datasets[!vapply(datasets, is.null, logical(1))]
if(!length(datasets)) stop("None of the .sas7bdat files could be read successfully.")
```

Special Character Detection and Report Generation

connect·share·advance




```
scan_specials <- function(df, dsn){
  char_cols <- names(df)[vapply(df, is.character, logical(1))]
  if(!length(char_cols) || nrow(df)==0) return(NULL)
  vals_df <- df |> dplyr::select(dplyr::all_of(char_cols)) |>
    tidyr::pivot_longer(dplyr::everything(),names_to="variable", values_to="var_value") |>
    dplyr::filter(!is.na(var_value)) |>
    dplyr::count(variable, var_value, name="value_ct") |>
    dplyr::mutate(varlen = nchar(var_value, type="chars", allowNA=TRUE))
  if(!nrow(vals_df)) return(NULL)
  vals_df |> dplyr::mutate(chars = stringi::stri_split_boundaries(var_value, type="character")) |>
    tidyr::unnest(chars) |> dplyr::group_by(variable, var_value, value_ct, varlen) |>
    dplyr::mutate(position=dplyr::row_number()) |>dplyr::ungroup() |>
    dplyr::mutate(codepoint = vapply(chars, function(x)
      ifelse(nchar(x)>0, utf8ToInt(x)[1], NA_integer_), integer(1)),
      spec_flag = dplyr::case_when(codepoint < 32 & !codepoint %in% c(9,10,13) ~ "[NPC]",
                                   codepoint == 127 ~ "[DEL]",codepoint > 127 ~ chars,TRUE ~ NA_character_
    dplyr::filter(!is.na(spec_flag)) |>dplyr::arrange(variable, var_value, position) |>
    dplyr::mutate(dataset = dsn) |> dplyr::relocate(dataset, .before = variable) |>
    dplyr::transmute(dataset, variable, var_value, value_ct, varlen,
                     position, char=chars, codepoint, spec_flag)}
all_reports <- purrr::imap(datasets, scan_specials)
non_empty <- purrr::keep(all_reports, ~ !is.null(.x) && nrow(.x)>0)
if(length(non_empty)){
  all_df <- dplyr::bind_rows(non_empty)
  out_all <- file.path(ss, paste0("ALL_special_char_report_",format(Sys.time(),"%Y%m%d_%H%M%S"), ".csv") )
  readr::write_csv(all_df, out_all)
  cat("\nCombined report saved to:", out_all,
      "(", nrow(all_df), "rows)\n") }else{
        cat("\nNo special characters found in any dataset.\n")}
cat("\nSummary:\n")
purrr::walk2(names(datasets), all_reports, ~ cat(sprintf("  - %s: %s\n",.x,
if(is.null(.y) || !nrow(.y)) "OK (no specials)"  else paste0(nrow(.y)," flagged characters"))))
```

Exceptional Handling - Special Character Detection

connect·share·advance

Handle Missing/Empty Datasets

Skip Failed Dataset Loads

Ignore Non-Character Variables

Managed NULL/NA Values

Handle Encoding Issues

Avoid Duplicate Entries

Process Large Datasets Efficiently

Notify when no Special Characters Found

Log Errors and Summary

Validation Summary - Special Character Detection

connect·share·advance

Validation Step		Description	Result	
	Dataset Loading	Verify SAS datasets are read into R		All datasets loaded successfully
	Variable Identification	Identify character variables		Variables detected correctly
	Special Character Scan	Scan for hidden/non-ASCII characters		Special characters detected
	Missing Value Check	Handle NULL/NA values during scan		Missing values ignored
	Character Position Check	Extract character position and code point		Accurate detection
	Duplicate Value Handling	Avoid repeated value processing		Duplicates controlled
	Error Handling	Handle dataset read errors		Program continued execution
	Report Generation	Generate special character report		CSV report created
	Result Consistency	Validate results across datasets		Consistent outputs

Macro Architecture Overview – Character Length Optimization

connect·share·advance

Read SAS Datasets into R

Identify Character Variables

Trim Leading and Trailing Spaces

Compute Max String Length

Compare Observed vs. Defined Length

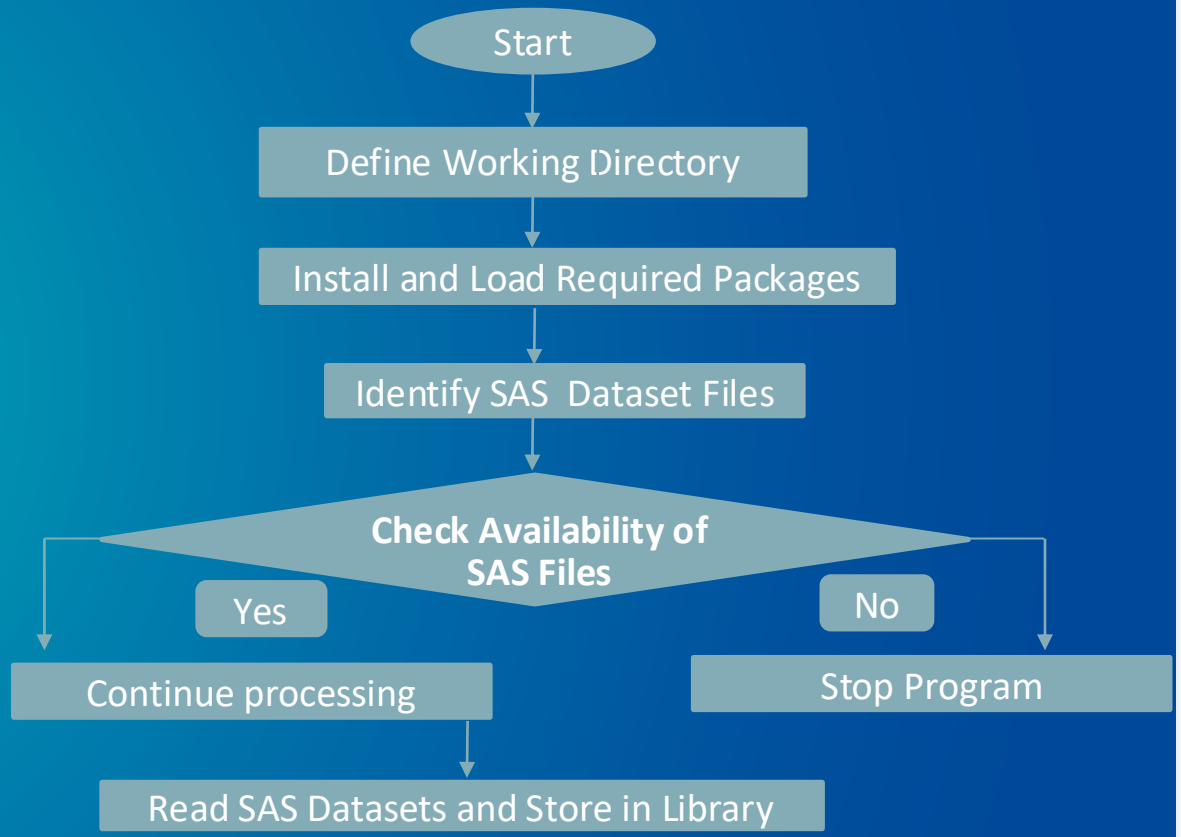
Derive Optimal Length

Update Variable-Length Metadata

Generate Length-Change Log

Export Optimization Report

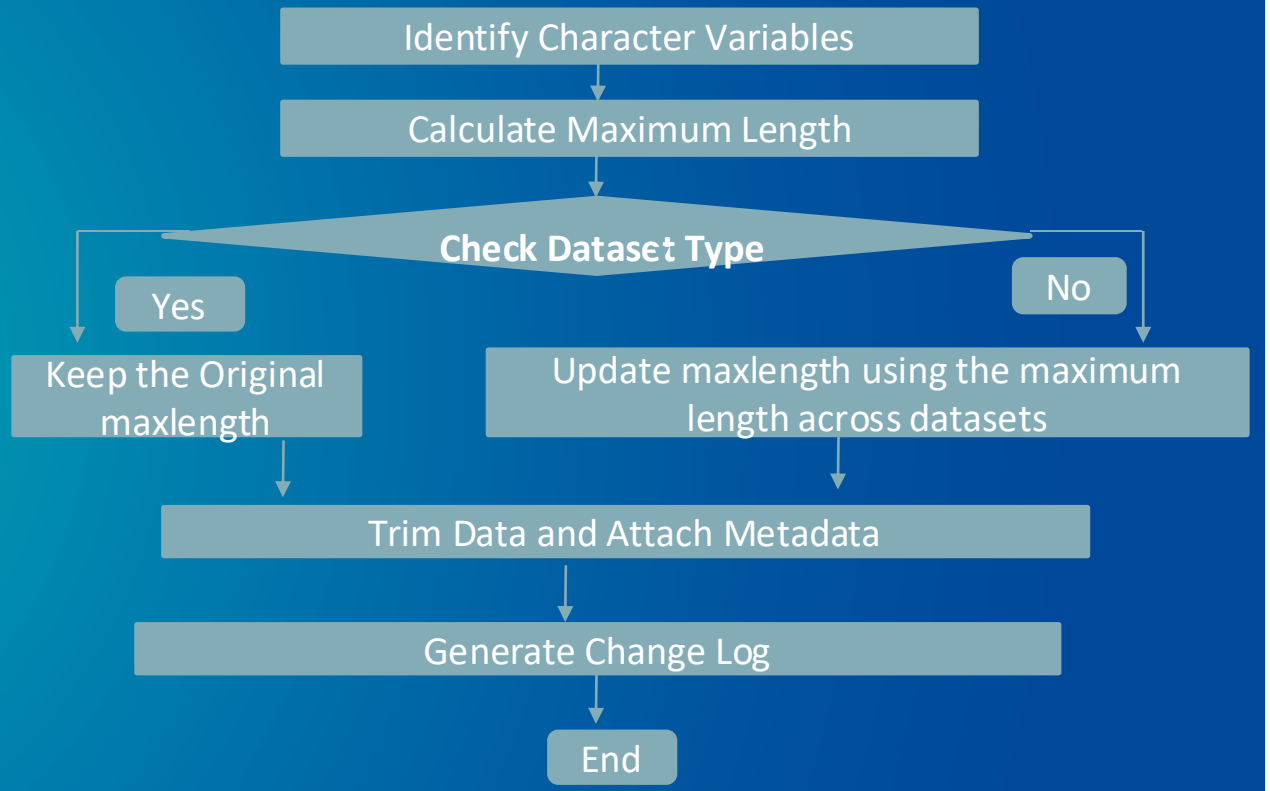
Environment Setup and SAS Dataset Processing




```r
# ---- 0) Setup ----
ss <- "C:/Users/rammohan.konda/OneDrive - Navitas Life Sciences Pvt. Ltd/R-training"
if (!dir.exists(ss)) dir.create(ss, recursive = TRUE)
need <- c("haven","dplyr","stringi","readr","purrr","tools","tibble")
to_install <- need[!sapply(need, requireNamespace, quietly = TRUE)]
if (length(to_install)) install.packages(to_install, dependencies = TRUE)
invisible(lapply(need, library, character.only = TRUE))
# ---- CONFIG ----
excluded_memnames <- character(0)
WRITE_CHANGE_LOG_CSV <- TRUE
CHANGE_LOG_CSV_PATH  <- file.path(ss, "char_length_change_log.csv")
# ---- helpers ----
repair_names <- function(df) {
  nm <- names(df); nm[!nzchar(nm) | is.na(nm)] <- "X"
  names(df) <- make.names(nm, unique = TRUE); df
}
is_char <- function(df) {
  nm <- names(df); nm[vapply(df, is.character, logical(1))] |> (\(x) x[nzchar(x)])()
}
max_byte_len_trim <- function(x) {
  x <- x[!is.na(x)]
  if (!length(x)) 0L else max(nchar(stri_trim_both(x), type = "bytes"))
}
# ---- 1) Read .sas7bdat ----
sas_files <- list.files(ss, "\\.sas7bdat$", ignore.case = TRUE, full.names = TRUE)
stopifnot(length(sas_files) > 0)
lib <- setNames(lapply(sas_files, \(p){
  nm <- file_path_sans_ext(basename(p)); cat("Reading:", nm, "\n")
  df <- tryCatch(read_sas(p), error = \(e){message("  ! Failed: ", e$message); NULL})
  if (is.null(df)) return(NULL); repair_names(df)
}), file_path_sans_ext(basename(sas_files)))
lib <- lib[!vapply(lib, is.null, logical(1))]
stopifnot(length(lib) > 0)
```

Character Length Analysis and Reporting

connect·share·advance




```r
uc <- purrr::imap_dfr(lib, \(df, mem){
  if (toupper(mem) %in% toupper(excluded_memnames)) return(NULL)
  cc <- is_char(df); if (!length(cc)) return(NULL)
  tibble::tibble(memname = mem, name = cc,
    maxlength = vapply(df[cc], max_byte_len_trim, integer(1)) ) }) |>
  dplyr::mutate(maxlength = pmax.int(1L, as.integer(dplyr::coalesce(maxlength, 1L))))
is_supp <- grepl("SUPP", toupper(uc$memname))
update_code_max <- dplyr::bind_rows(
  uc[ is_supp, ] |> dplyr::mutate(newmax = maxlength),
  uc[!is_supp, ] |> dplyr::group_by(name) |> dplyr::mutate(newmax = max(maxlength)) |> dplyr::ungroup()
) |>
  dplyr::mutate(newmax = pmax.int(1L, as.integer(newmax))) |>
  dplyr::arrange(memname, name)
meta_by_ds <- update_code_max |>
  dplyr::group_by(memname) |>
  dplyr::reframe(lengths = list(stats::setNames(newmax, name)))
shrunk_lib <- purrr::imap(lib, \(df, mem){
  if (toupper(mem) %in% toupper(excluded_memnames)) return(df)
  cc <- is_char(df)
  if (length(cc)) df[cc] <- lapply(df[cc], stringi::stri_trim_both)
  idx <- match(mem, meta_by_ds$memname)
  if (!is.na(idx)) {
    lens <- meta_by_ds$lengths[[idx]]
    attr(df, "sas_char_lengths") <- lens }
  df })
change_log <- update_code_max |>
  dplyr::select(memname, name, maxlength, newmax)
cat("\n--- CHANGE LOG (first 200 rows) ---\n"); print(utils::head(change_log, 200))
if (WRITE_CHANGE_LOG_CSV) {
  readr::write_csv(change_log, CHANGE_LOG_CSV_PATH)
  message("Change log written to: ", CHANGE_LOG_CSV_PATH) }
cat("\nAll steps completed.\n")
```

Exceptional Handling – Character Length Optimization

connect·share·advance

- Missing Dataset Handling
- Read Error Handling
- Character Variable Filtering
- Missing Value Handling
- Whitespace Handling
- Minimum Length Protection
- No Character Variable Handling
- Length Validation
- Change Log Recording

Validation Summary - Character Length Optimization

connect·share·advance

Validation Step		Description	Result	
	Dataset Loading	Verify SAS datasets are read into R		All datasets loaded successfully
	Variable Identification	Detect character variables		Variables identified correctly
	White Space Handling	Trim leading and trailing spaces		Spaces removed
	Length Calculation	Compute maximum string length		Correct maximum length calculated
	Length Comparison	Compare observed vs. defined length		Differences identified
	Optimization Check	Ensure no data truncation		Data integrity maintained
	Metadata Assignment	Assign optimized variable length		Metadata updated
	Change Log Generation	Record length changes		CSV change log generated
	Result Verification	Validate consistency of results		Accurate outputs

Detecting Hidden Characters

Dataset	Variable	Var_Value	Value_CT	Varlen	Postion	Char	Codepoint	Spec_Flag
df1	address	12 Main StÂ Apt 3	1	16	11	Â	160	Â
df1	address	22 Cedar Dr	1	12	12		1	[NPC]

Optimizing Character Lengths for Efficient SAS Workflows

Memname	Name	Maxlength	Newmax
demo	ETHNIC	22	22
demo	RACE	5	5
demo	SUBJID	8	8
ds	DOMAIN	2	2
ds	DSCAT	18	18
ds	DSDECOD	25	25
ds	DSSCAT	16	16
ds	DSTERM	25	25
ds	Epoch	9	9
ds	STUDYID	4	8
eg	VISIT	8	19
dv	DOMAIN	2	2
dv	DVREFID	20	20
dv	DVTERM	196	196
dv	STUDYID	8	8
vs	VISIT	19	19

Key Results, Reusability, and Regulatory Alignment

connect·share·advance

Key Results

- Improved data quality by detecting hidden special characters
- Reduced dataset size and improved processing performance
- Generated automated validation reports

Reusability

- Framework reusable across multiple clinical studies
- Easy integration with existing SAS/R workflows
- Works with multiple datasets automatically

Regulatory Alignment

- Provides audit-ready reports and logs
- Supports data validation and traceability
- Aligns with clinical data quality and compliance practices

Performance Impact (Manual vs. Automated)

connect·share·advance

Parameter/Task	Manual Process	Automated Process
Character Length Handling	Defined manually; often over-allocated	Optimized based on actual maximum length
Special Character Detection	Requires manual review of variables	Automatically scans all datasets and variables
Processing Time	Slow and time-consuming	Faster processing with automated scripts
Error Risk	Higher chance of human error	Reduced errors through automated checks
Data Validation	Multiple manual validation steps	Automated validation reports generated
Dataset Size	Often larger due to inefficient lengths	Reduced dataset size after optimization
Reproducibility	Difficult to maintain consistency	Consistent results across studies
Documentation	Manual documentation required	Auto-generated logs and reports
Scalability	Hard to scale for many datasets	Easily processes multiple dataset automatically

Challenges

connect·share·advance

Hidden Unicode/Special Characters

Large Dataset Processing Time

High Memory Consumption

Inconsistent Dataset Structures

Risk of Data Truncation

Limited Data Quality Checks

Future Directions

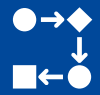
connect·share·advance

Enhanced Character Detection Engine

Chunk-Based Processing

Automated Validation Layer

Parallel Processing



Metadata-Driven Framework

Advanced Data Quality Framework

Conclusion

- The automated R framework improves clinical data quality and validation efficiency.
- It enables faster detection of hidden special characters in SAS datasets.
- Character length optimization helps reduce dataset size and improve performance.
- Automated scanning minimizes manual review and human errors.
- The framework can process multiple datasets efficiently.
- The solution is reusable across different clinical studies.
- Automated reports support better documentation and traceability.
- The workflow supports regulatory compliance and validation standards.
- Overall, automation enhances efficiency, scalability, and reliability in clinical programming.

connect·share·advance

THANK YOU!

phuse

The Global Healthcare Data Science Community

Contact Channels

office@phuse.global

phuse.global

Social Media

/company/phuse

/phusetube

/phuse_global

/phusebook