



PRECISION TIMING – IMPLEMENTING EPOCH CALCULATIONS WITH R FOR CLINICAL DATA INTEGRITY



Sheik Ibrahim
Navitas Life Sciences



Ram Mohan Konda
Navitas Life Sciences



[Website](#)



[LinkedIn](#)



AGENDA

1

TITLE AND OBJECTIVE

6

PROGRAMMING WORKFLOW

2

WHY AUTOMATION

7

FLOWCHART

3

**OVERVIEW OF THE m_epoch
ALGORITHM**

8

CHALLENGES

4

INPUT PARAMETERS

9

FUTURE DIRECTIONS

5

**VISUAL PRESENTATION WITH
INPUT AND OUTPUT DATASETS**

10

CONCLUSION

TITLE AND OBJECTIVE

Title: Precision Timing: Implementing Epoch Calculations in R for Clinical Data Integrity

Objective: To accurately assign clinical observations to study epochs even when dates are partial, incomplete, or varied in format, ensuring consistency in regulatory-grade clinical datasets.

Key Challenges Addressed

- Partial ISO-8601 Dates (YYYY/YYYY-MM-DD)
- Multi-epoch subject data
- Incorrect Epoch boundaries
- Ordering ambiguity

WHY AUTOMATION?



Clinical Data
Integrity



Reproducibility and
Consistency



Complex
Rules/Logic



Time Saving



Compliance and
Audits



Real-Time
Monitoring

CASE STUDY OVERVIEW

PRECISION TIMING WITH R IN CLINICAL TRIALS

Context and Objective: Clinical trials require accurate epoch classification (e.g. TREATMENT, FOLLOW-UP) to assess drug efficacy and safety. Manual methods are slow, error-prone, and not scalable.

Goal: Build an automated, validated R pipeline to assign epochs using date rules from clinical visit data.

Solution: Develop an R-based workflow using:

- Dplyr, lubridate, stringr for data transformation
- Flowchart-driven logic to map visit dates to epochs
- Handling of partial dates, overlaps, and timestamp inconsistencies

Features: Custom `m_epoch()` function:

- Parses partial dates (year, month-year, full-date)
- Matches records to epochs using start/end windows
- Validates input and handles edge cases

OVERVIEW OF THE m_epoch ALGORITHM



Validate Inputs



Validate Subject
Element Metadata



Process for Partial
Dates



Join in_data and
se_in



Sort based on the
Requirements



Pick the right
needed EPOCH

COMPREHENSIVE LIST OF INPUT PARAMETERS

Input Parameter	Test#	Test	Expected Outcome
in_data	1.01	Supply in_data name for analysis	Analysis performed using the specified in_data
	1.02	No in_data name is supplied	Relevant error message is written to log, and function stops
se_in	2.01	Supply se_in name for analysis	Analysis performed using the specified se_in
	2.02	No se_in name is supplied	Relevant error message is written to log, and function stops
dctvar	3.01	Supply dctvar name for analysis	Analysis performed using the specified dctvar
	3.02	No dctvar name is supplied	Relevant error message is written to log, and function stops
opt	4.01	No opt name is supplied	Default is "TRT_LATE"
	4.02	opt = "EARLY"	If multiple elements for the same date, it picks the earliest element
	4.03	opt = "LATE"	If multiple elements for the same date, it picks the latest element
	4.04	opt = "TRT_EARLY"	If multiple treatment elements for the same date, it picks the earliest treatment element
	4.05	opt = "TRT_LATE"	If multiple treatment elements for the same date, it picks the latest treatment element
partial	5.01	partial = "TRUE"	EPOCH is derived for partial dates
	5.02	partial = "FALSE"	EPOCH is not derived for partial dates
	5.03	No partial is supplied	Relevant error message is written to log, and function stops
debug	6.01	debug = "TRUE"	Intermediate datasets are kept for debugging
	6.02	debug = "FALSE"	Intermediate datasets are removed
	6.03	No debug is supplied	Default is "FALSE"

INPUT DATASETS FOR EPOCH CALCULATION

in_data: Dates may be partial (YYYY or YYYY-MM) and will be standardized.

USUBJID	PRSTDTC	PRSEQ
01	2020-01-01	1
01	2020-01-05	2
01	2020-01-14	3
01	2020-01-18T11:30	4
02	2019-06	1
03	2018	1

SE domain (se_in): SE dataset consists start (SESTDTC) and end (SEENDTC) dates for each element

USUBJID	TAETORD	EPOCH	SESTDTC	SEENDTC
01	1	SCREENING	2019-12-24	2020-01-01
01	2	TREATMENT PERIOD 1	2020-01-01	2020-01-05
01	3	TREATMENT PERIOD 2	2020-01-05	2020-01-15
01	4	FOLLOW-UP	2020-01-16	2020-01-25
02	1	SCREENING	2019-06-01	2019-06-30
02	2	TREATMENT PERIOD 1	2019-06-30	2019-08-21
03	1	SCREENING	2018-01-01	2018-12-31

OUTPUT DATASET AFTER EPOCH CALCULATION

- EPOCH is mapped based on the user's needs using OPT parameter
- m_epoch (in_data = IN_DATA, se_in = SE, dtcvar = "PRSTDTC", opt = "TRT_LATE", partial = TRUE, debug = FALSE)

USUBJID	PRSTDTC	PRSEQ	EPOCH
01	2020-01-01	1	TREATMENT PERIOD 1
01	2020-01-05	2	TREATMENT PERIOD 2
01	2020-01-14	3	TREATMENT PERIOD 2
01	2020-01-18T11:30	4	FOLLOW-UP
02	2019-06	1	TREATMENT PERIOD 1
03	2018	1	SCREENING

USUBJID	TAETORD	EPOCH	SESTDTC	SEENDTC
01	1	SCREENING	2019-12-24	2020-01-01
01	2	TREATMENT PERIOD 1	2020-01-01	2020-01-05
01	3	TREATMENT PERIOD 2	2020-01-05	2020-01-15
01	4	FOLLOW-UP	2020-01-16	2020-01-25
02	1	SCREENING	2019-06-01	2019-06-30
02	2	TREATMENT PERIOD 1	2019-06-30	2019-08-21
03	1	SCREENING	2018-01-01	2018-12-31

SUBJECT PROCEDURE SCHEDULE AND STUDY EPOCH MAPPING

USUBJID	PRSTDTC	PRSEQ	EPOCH (OPT = "EARLY")	EPOCH (OPT = "LATE")	EPOCH (OPT = "TRT_EARLY")	EPOCH (OPT = "TRT_LATE")
1	2020-01-01	1	SCREENING	TREATMENT PERIOD 1	TREATMENT PERIOD 1	TREATMENT PERIOD 1
1	2020-01-05	2	TREATMENT PERIOD 1	TREATMENT PERIOD 2	TREATMENT PERIOD 1	TREATMENT PERIOD 2
1	2020-01-14	3	TREATMENT PERIOD 2	TREATMENT PERIOD 2	TREATMENT PERIOD 2	TREATMENT PERIOD 2
1	2020-01-18T11:30	4	FOLLOW-UP	FOLLOW-UP	FOLLOW-UP	FOLLOW-UP
2	2019-06	1	SCREENING	TREATMENT PERIOD 1	TREATMENT PERIOD 1	TREATMENT PERIOD 1
3	2018	1	SCREENING	SCREENING	SCREENING	SCREENING

EXCEPTIONAL HANDLING



Handling Partial
Dates



Overlapping Epoch
Ranges



Missing Start/End
Dates in Epoch
Dataset



Invalid or
Unexpected Date
Lengths



Invalid 'Opt'
Parameter Values



Zero Observation
or Empty Datasets



Delete Temp
Datasets

INPUT DATASET VALIDATION AND INTEGRITY CHECKS

```
m_epoch <- function(in_data = NULL, se_in = NULL,
                    dtcvar,
                    opt = "trt_early", partial = TRUE, debug = FALSE) {
```

```
# === 1. Check input dataset presence and validity ===
```

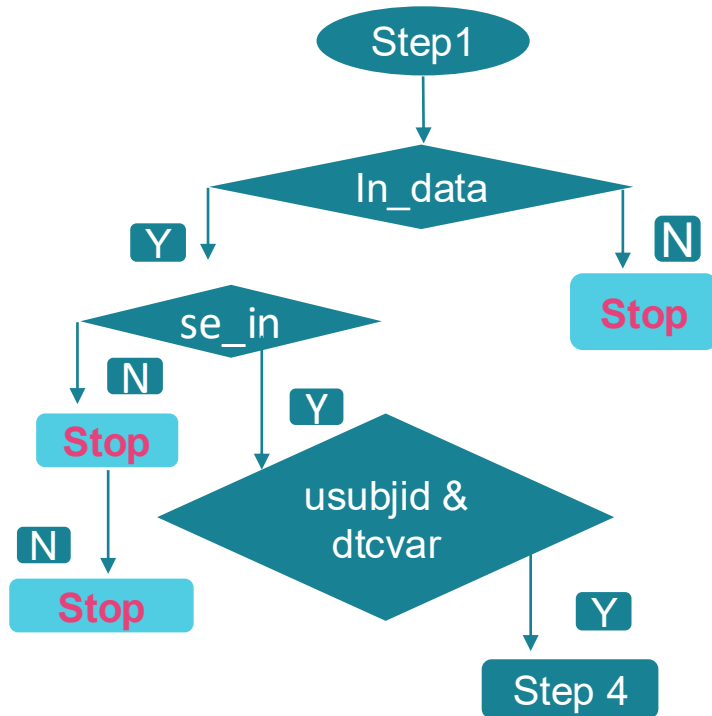
```
if (missing(in_data)) {
  stop("ERROR: No input dataset (in_data) provided.")
}
if (!is.data.frame(in_data)) {
  stop("ERROR: 'in_data' is not a data frame.")
}
```

```
# === 2. Check study epoch dataset (se_in) ===
```

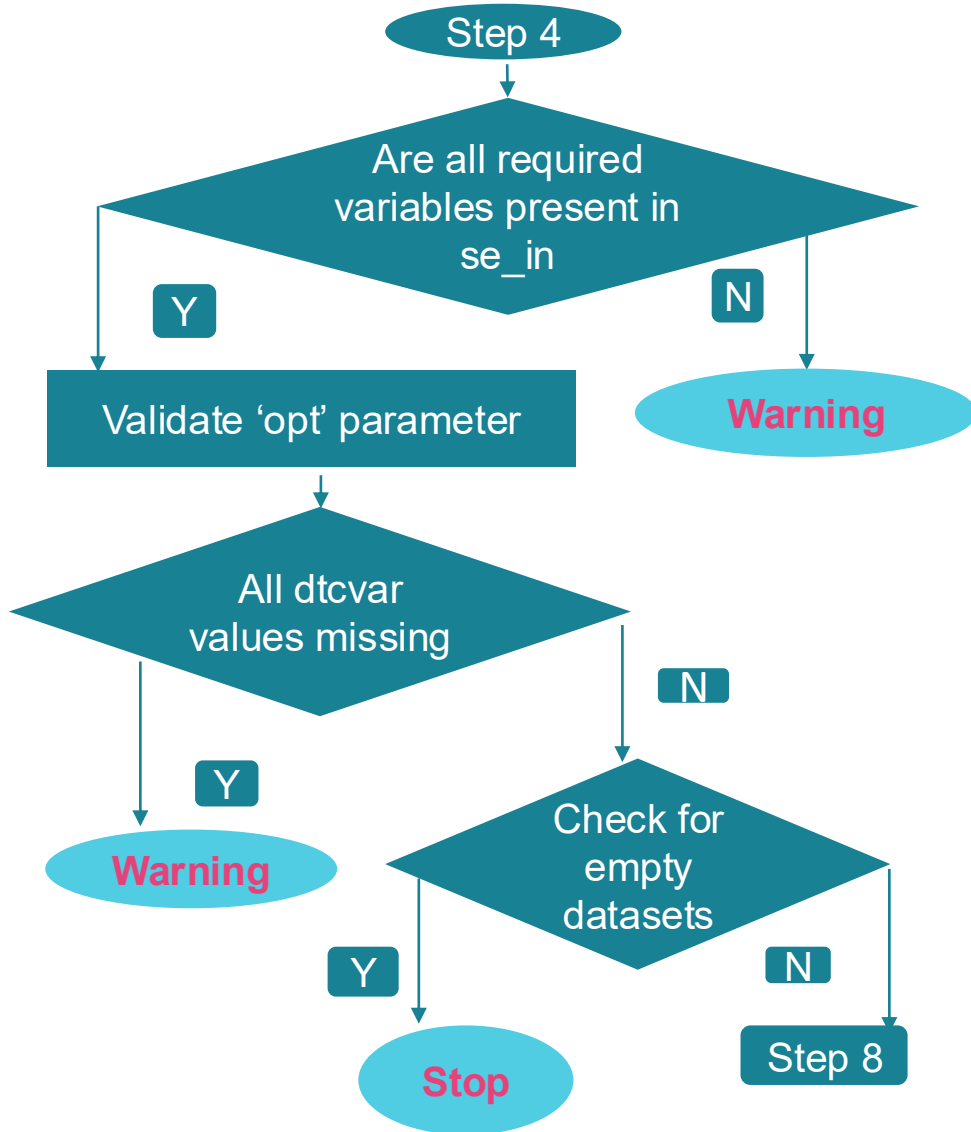
```
if (missing(se_in)) {
  stop("ERROR: No input dataset (se_in) provided.")
}
if (!is.data.frame(se_in)) {
  stop("ERROR: 'se_in' is not a data frame.")
}
```

```
# === 3. Check required variables in in_data ===
```

```
if (!"USUBJID" %in% names(in_data)) {
  stop("ERROR: Input dataset does not contain required variable 'USUBJID'.")
}
if (!dtcvar %in% names(in_data)) {
  stop(paste0("ERROR: Input dataset does not contain the date variable '", dtcvar, "'."))
}
```



PARAMETER AND STRUCTURAL VALIDATION OF STUDY EPOCH DATA



```

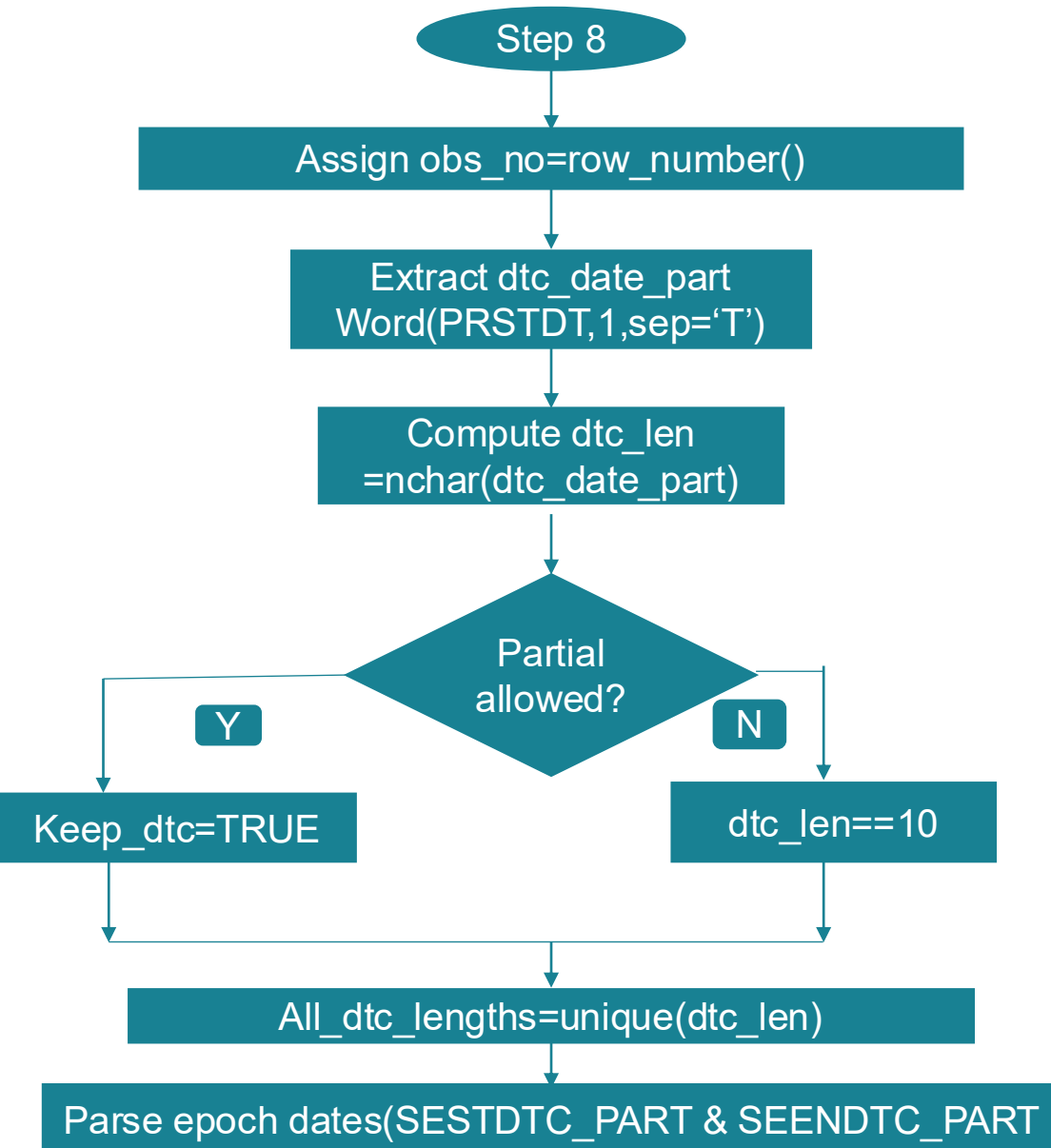
# == 4. Check required variables in se_in ==
required_se_vars <- c("USUBJID", "TAETORD", "EPOCH", "SESTDTC", "SEENDTC")
missing_vars <- setdiff(required_se_vars, names(se_in))
if (length(missing_vars) > 0) {
  warning(paste("WARNING: Study epoch dataset missing variable(s):", paste(missing_vars, collapse = ", ")))
}

# == 5. Validate opt parameter ==
valid_opts <- c("EARLY", "LATE", "TRT_EARLY", "TRT_LATE")
opt <- toupper(opt)
if (!(opt %in% valid_opts)) {
  warning(paste0("warning: 'opt' value '", opt, "' is invalid. Using default 'TRT_LATE'."))
  opt <- "TRT_LATE"
}

# == 6. Check dtcvar for missing values ==
if (all(is.na(in_data[[dtcvar]]))) {
  stop(paste0("ERROR: Date variable '", dtcvar, "' contains only missing values."))
}

# == 7. Check for empty datasets ==
if (nrow(in_data) == 0) stop("ERROR: Input dataset has zero observations.")
if (nrow(se_in) == 0) stop("ERROR: Study epoch dataset has zero observations.")
  
```

INPUT DATA TEMPORAL STANDARDIZATION AND FILTERING



```

# === 8. Prepare date variables in in_data (Extract Date String Only) ===
in_data <- in_data %>%
  mutate(
    obs_no = row_number(),
    # Extract the date part (YYYY-MM-DD or partial) before 'T'
    dtc_date_part = word(as.character(.data[[dtcvar]]), 1, sep = "T"),
    # Store length for filtering later
    dtc_len = nchar(dtc_date_part),
    # Flag to keep only full dates if partial=FALSE
    keep_dtc = if(partial) TRUE else (dtc_len == 10)
  )

```

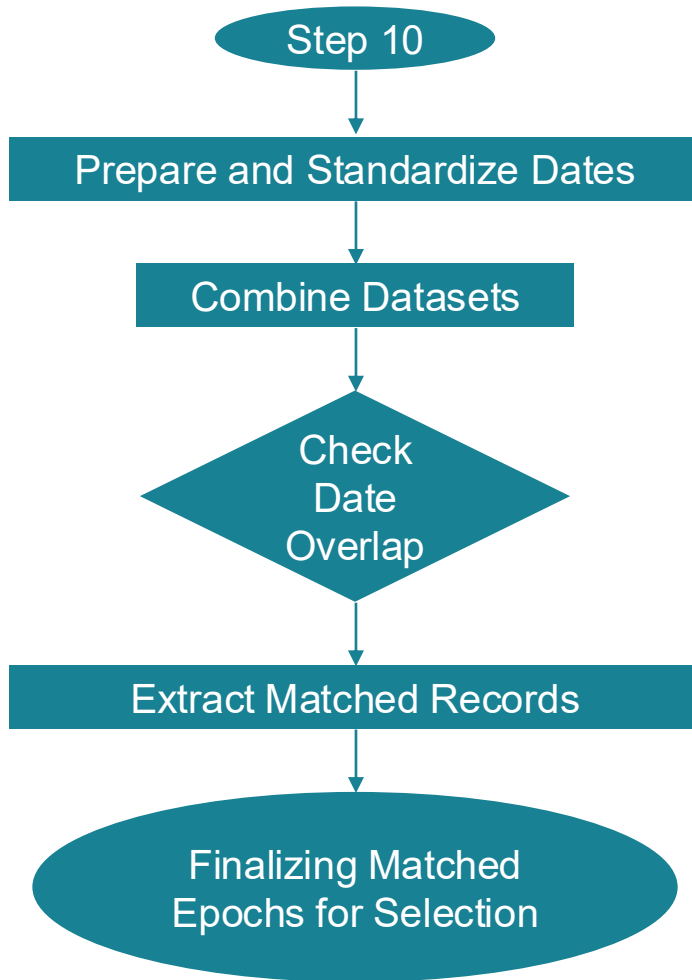
```
all_dtc_lengths <- unique(in_data$dtc_len)
```

```

# === 9. Prepare date variables in se_in (Extract Date String Only) ===
se_in <- se_in %>%
  mutate(
    SESTDTC_PART = word(SESTDTC, 1, sep = "T"),
    SEENDTC_PART = word(SEENDTC, 1, sep = "T")
  )

```

TEMPORAL MATCHING AND PARTIAL DATE RANGE ALIGNMENT



```

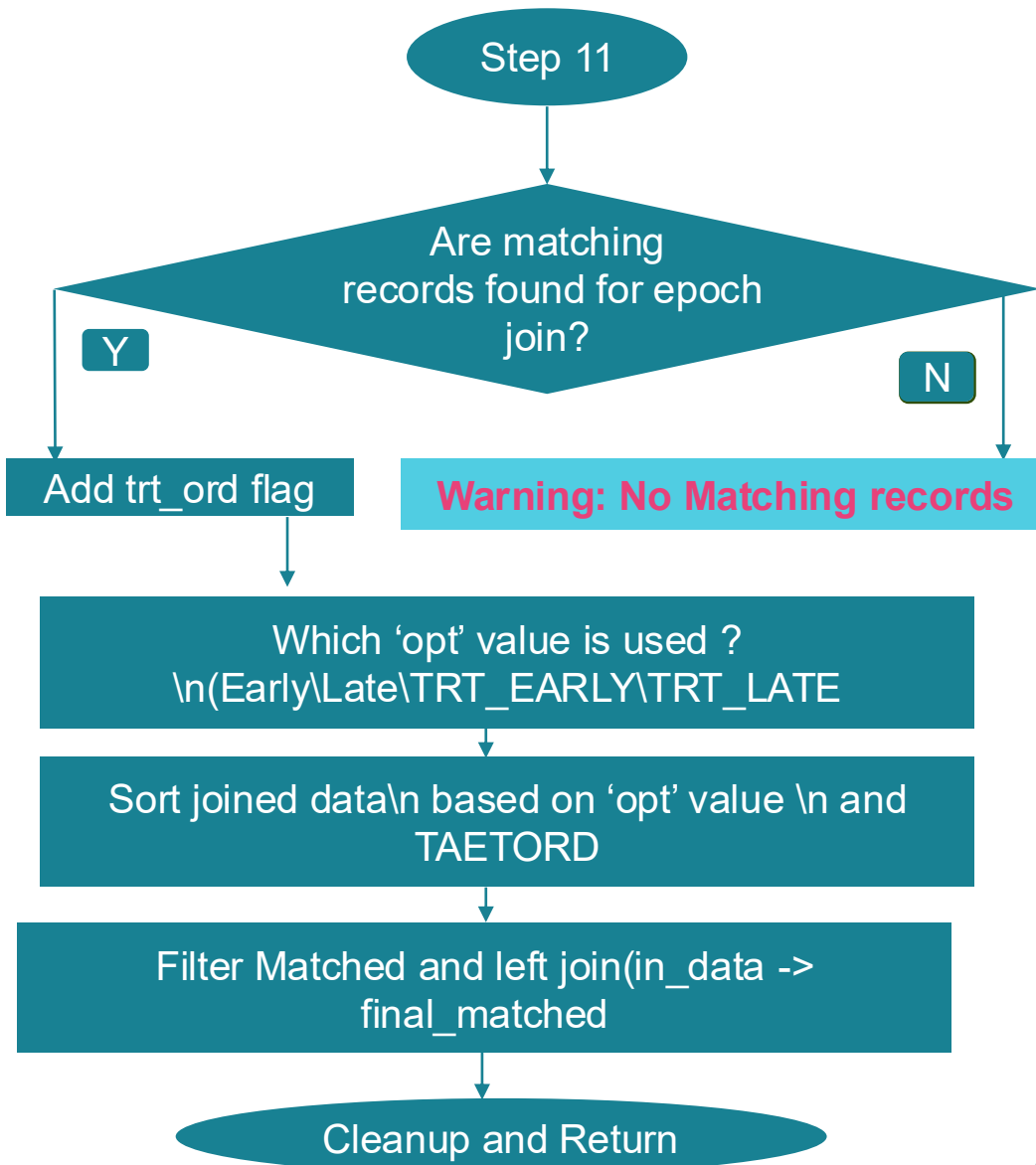
# --- 10. LOGIC FOR PARTIAL MATCHING ** ---
in_data_filtered <- in_data %>%
  filter(keep_dtc)
matched <- in_data_filtered %>%
  left_join(se_in, by = "USUBJID", relationship = "many-to-many") %>%
  rowwise() %>%
  filter(!is.na(dtc_date_part) & !is.na(SESTDTC_PART) & !is.na(SEENDTC_PART) &
    ( (dtc_len == 10 &
      (dtc_date_part >= SESTDTC_PART) & (dtc_date_part <= SEENDTC_PART)) |

      (partial & dtc_len == 7 &
        (dtc_date_part >= str_sub(SESTDTC_PART, 1, 7)) &
        (dtc_date_part <= str_sub(SEENDTC_PART, 1, 7))) |

      (partial & dtc_len == 4 &
        (dtc_date_part >= str_sub(SESTDTC_PART, 1, 4)) &
        (dtc_date_part <= str_sub(SEENDTC_PART, 1, 4)))

    )
  ) %>%
  ungroup() %>%
  select(-starts_with("DTC_"), -starts_with("SEST_"), -starts_with("SEEN_"))
  
```

POST-MATCH RECONCILIATION, TIE-BREAKING, AND FINAL EPOCH ASSIGNMENT



```

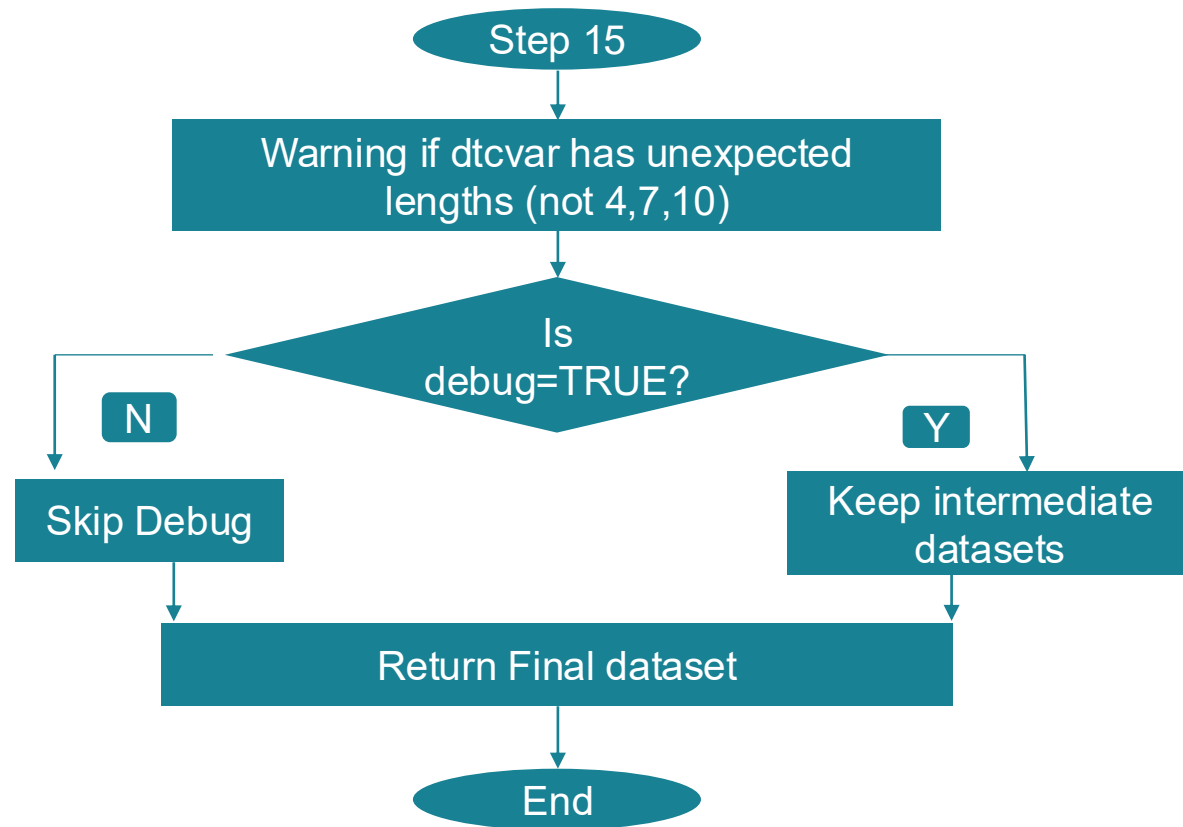
# === 11. Check if join found matches ===
if (nrow(matched) == 0) {
  warning("WARNING: No matching records found in study epoch join.")
}

# === 12. Add trt_ord for sorting (on matched data only) ===
matched <- matched %>% mutate(trt_ord = ifelse( str_detect(toupper(EPOCH), "TREATMENT"), 0, 1))

# === 13. Sort by 'opt' (for tie-breaking) ===
if (opt == "EARLY") {
  matched <- matched %>% arrange(obs_no, TAETORD)
} else if (opt == "LATE") {
  matched <- matched %>% arrange(obs_no, desc(TAETORD))
} else if (opt == "TRT_EARLY") {
  matched <- matched %>% arrange(obs_no, trt_ord, TAETORD)
} else if (opt == "TRT_LATE") {
  matched <- matched %>% arrange(obs_no, trt_ord, desc(TAETORD))
}

# === 14. Keep only first matched row per obs_no AND Merge back ===
final_matched <- matched %>%
  distinct(obs_no, .keep_all = TRUE) %>%
  select(obs_no, EPOCH_NEW = EPOCH)
final <- in_data %>% left_join(final_matched, by = "obs_no") %>%
  # Rename EPOCH_NEW to EPOCH (overwriting any existing EPOCH)
  mutate(EPOCH = EPOCH_NEW) %>%
  # Select original columns + new EPOCH, removing all temp columns
  select(-dtc_date_part, -dtc_len, -obs_no, -keep_dtc, -EPOCH_NEW)
  
```

POST-PROCESSING VALIDATION, DIAGNOSTIC REPORTING, AND FINAL OUTPUT



```

# === 15. Warn about unexpected date length in dtcvar ===
invalid_len <-all_dtc_lengths[!(all_dtc_lengths %in% c(4, 7, 10))]
if (length(invalid_len) > 0) {
  warning("Warning: Some date variable values have unexpected lengths (expected 4, 7, or 10 characters).")
}

# === 16. DEBUG: Push datasets to Global Environment for visibility ===
if (debug) {
  final <- final
  matched <- matched
  final_matched <- final_matched
  in_data_filtered <- in_data_filtered
}

return(final)
}
  
```

```

result <- m_epoch(in_data = in_data, se_in = study_epoch,dtcvar = "PRSTDTC",
                  opt = "TRT_LATE", partial =TRUE, debug = TRUE)
list2env(result, envir = .GlobalEnv)
print(result)
  
```

CHALLENGE



Inconsistent
Timestamp
Formats



Partial and
Missing Dates



Overlapping
Treatment
Periods



Misaligned Visit
Windows



Data Entry
Errors



Time Zone
Variations

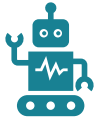


Performance
with Large
Datasets



Variable Naming
Inconsistency

FUTURE DIRECTION



Machine
Learning for
Missing Dates



Interactive
Dashboards
(Shiny, Plotly)



Automated
Error Detection
System



Integration with
CDISC Standards



Parallel
Processing
Optimization



Temporal Data
Analytics for
Efficacy Insights



User-Friendly
GUI Tool

CONCLUSION

-  **High Precision:** Achieved an impressive accuracy rate of over 99%
-  **Full Transparency:** Provides a clear, visible, and easy-to-follow workflow.
-  **Audit-Ready:** Built to be compliant and ready for regulatory inspections.
-  **Simplified Delivery:** Streamlines the analysis of safety and efficacy data.
-  **AI Evolution:** Future plans to integrate **Machine Learning** capabilities.
-  **Real-Time Insights:** Interactive dashboards will soon offer instant quality checks.
-  **Smarter Data Handling:** Better tools are coming to manage missing temporal data.
-  **Future-Proof:** Designed for continuous enhancement and scalability.

POWERING POSSIBILITIES IN DRUG DEVELOPMENT

Navitas is a knowledge-driven, tech-enabled CRO offering bespoke solutions across the drug development lifecycle. From strategic advisory services that navigate complex regulatory landscapes to clinical development ensuring compliant, data-driven operations, and post-marketing solutions enhancing product value with safety surveillance, lifecycle management, and real-world evidence—we integrate deep expertise, cutting-edge technology, and industry insights to drive success.





Driving better outcomes through insights



[Website](#)



[LinkedIn](#)



Sheik Ibrahim

Sheik.ibrahim@navitaslifesciences.com



Ram Mohan Konda

rammohan.konda@navitaslifesciences.com