R for Clinical Study Reports and Submission

Yilong Zhang Nan Xiao Keaven Anderson
Yalin Zhu

Table of contents

Welcome

Preface

Folder structure

In the development of clinical trials, it is necessary to create and manage source code for generating and delivering Study Data Tabulation Model (SDTM), Analysis Dataset Model (ADaM) datasets, as well as tables, listings, and figures (TLFs). This is particularly evident in Phase 3 trials, where numerous TLFs are needed for submission. To effectively handle the large number of programs involved in such endeavors, it is essential to establish a consistent and well-defined folder structure for managing the analysis and reporting (A&R) project of a clinical trial.

To streamline the organization of source code and documentation for a clinical trial A&R project, we suggest employing the R package folder structure. This folder structure is extensively utilized within the R community and is well-defined, often found in repositories like CRAN. By adopting this structure, you can benefit from a standardized and widely accepted framework for managing your A&R-related materials in an efficient and accessible manner.

Using the R package folder structure provides a consistent approach that simplifies communication among developers, both within and across organizations.

- For newcomers to R development, creating R packages is an essential step when sharing their work with others. The R community offers a widely adopted folder structure accompanied by excellent tutorials and free tools.

- For an experienced R developer, there is a minimal learning curve.
- For an organization, adopting the R package folder structure simplifies the development of processes, tools, templates, and training. It enables the use of a unified folder structure for building and maintaining standardized tool and analysis projects.

The workflow around an R package can also improve the traceability and reproducibility of an analysis project (Marwick, Boettiger, and Mullen 2018).

We will revisit the folder structure topic when discussing project management for a clinical trial project.

Additionally, the R package folder structure is also recommended for developing Shiny apps, as discussed in Chapter 20 of the Mastering Shiny book and the Engineering Production-Grade Shiny Apps book.

In this book

This book is designed for intermediate-level readers who possess knowledge in both R programming and clinical development. Each part of the book makes certain assumptions about the readers' background:

- Part 1, titled "Delivering TLFs in CSR", provides general information and examples on creating tables, listings, and figures. It assumes that readers are individual contributors to a clinical project with prior experience in R programming. Familiarity with data manipulation in R is expected. Some recommended references for this part include Hands-On Programming with R, R for Data Science, and Data Manipulation with R.
- Part 2, titled "Clinical trial project", provides general information and examples on managing a clinical trial A&R project. It assumes that readers are project leads who have experience in R package

development. Recommended references for this part include R Packages and the tidyverse style guide.

- Part 3, titled "eCTD submission package", provides general information on preparing submission packages related to the CSR in the electronic Common Technical Document (eCTD) format. It assumes that readers are project leads of clinical projects who possess experience in R package development and submission.

Philosophy

We share the same philosophy described in the introduction of the R Packages book (Wickham and Bryan 2023), which we quote below:

- "Anything that can be automated, should be automated."
- "Do as little as possible by hand. Do as much as possible with functions."

Authors and contributors

This document is a collaborative effort maintained by a community. As you read through it, you also have the opportunity to contribute and enhance its quality. Your input and involvement play a vital role in shaping the excellence of this document.

- Authors: made significant contributions to at least one chapter, constituting the majority of the content.

  Yilong Zhang, Nan Xiao, Keaven Anderson, Yalin Zhu

- Contributors: contributed at least one commit to the source code.

  We are grateful for all the improvements brought by these contributors (in chronological order): Yujie Zhao (@LittleBeannie), Aiming Yang, Steven Haesendonckx (@SHAESEN2), Howard Baek (@howardbaek), Xiaoxia Han (@echohan), Jie Wang (@ifendo).

Part I.

Delivering TLFs in CSR

1. Overview

1.1. Background

Submitting clinical trial results to regulatory agencies is a crucial aspect of clinical development. The Electronic Common Technical Document (eCTD) has emerged as the global standard format for regulatory submissions. For instance, the United States Food and Drug Administration (US FDA) mandates the use of eCTD for new drug applications and biologics license applications.

A CSR provides comprehensive information about the methods and results of an individual clinical study. To support the statistical analysis, numerous tables, listings, and figures are included within the main text and appendices. As part of the CDISC pilot project, an example CSR is also available for reference. If you seek additional examples of CSR, you can visit the clinical data website of the European Medicines Agency (EMA) clinical data website.

The creation of CSR is a collaborative effort that involves various professionals such as clinicians, medical writers, statisticians, statistical programmers. In this context, we will focus on the specific deliverables provided by statisticians and statistical programmers.

Within an organization, these professionals typically collaborate to define, develop, validate, and deliver the necessary tables, listings, and figures (TLFs) for a CSR. These TLFs serve to summarize the efficacy and/or safety of the pharmaceutical product under study. In the pharmaceutical industry, Microsoft Word is widely utilized for CSR preparation. As a

result, the deliverables from statisticians and statistical programmers are commonly provided in formats such as `.rtf`, `.doc`, `.docx` to align with industry standards and requirements.

Our focus is to demonstrate the process of generating TLFs in RTF format, which is commonly employed in CSRs. The examples provided in this chapter adhere to the ICH E3 guidance and the FDA's PDF Specifications.

> **Note**
>
> FDA's PDF specification is a general reference. Each organization can define more specific TLF format requirements that can be different from the examples in this book. The FDA's PDF specification serves as a general reference for formatting requirements. Each organization has the flexibility to define its own specific requirements for TLFs. These specific format requirements may differ from the examples provided in this book. It is advisable to consult and adhere to the guidelines and specifications set by your respective organization when preparing TLFs for submission.

By following the ICH E3 guidance, most of TLFs in a CSR are located at

- Section 10: Study participants
- Section 11: Efficacy evaluation
- Section 12: Safety evaluation
- Section 14: Tables, listings, and figures referrals but not included in the text
- Section 16: Appendices

1.2. Datasets

The dataset structure follows CDISC Analysis Data Model (ADaM).

In this project, we used publicly available CDISC pilot study data, which is accessible through the CDISC GitHub repository.

To streamline the process, we have downloaded all the datasets from the repository and stored them in the `data-adam/` folder within this project. Additionally, we converted these datasets from the `.xpt` format to the `.sas7bdat` format for ease of use and compatibility. The dataset structure adheres to the CDISC Analysis Data Model (ADaM) standard.

1.3. Tools

To exemplify the generation of TLFs in RTF format, we rely on the functionality provided by two R packages:

- tidyverse: preparation of datasets in a format suitable for reporting purposes. The tidyverse package offers a comprehensive suite of tools and functions for data manipulation and transformation, ensuring that the data is structured appropriately.
- r2rtf: creation RTF files. The r2rtf package offers functions specifically designed for generating RTF files, allowing us to produce TLFs in the desired format.

> **Note**
>
> There are indeed several other R packages available that can assist in creating TLFs in ASCII, RTF, and Word formats such as rtables, huxtable, pharmaRTF, gt, officer, and flextable. However, in this particular context, we will concentrate on demonstrating the concept using the r2rtf package. It is highly recommended for readers to

explore and experiment with various R packages to identify the most suitable tools that align with their specific needs and objectives.

1.3.1. tidyverse

The tidyverse is a comprehensive collection of R packages that aim to simplify the workflow of manipulating, visualizing, and analyzing data in R. These packages adhere to the principles outlined in the the tidy tools manifesto and offer user-friendly interfaces for interactive data analysis.

The creators of the tidyverse, Posit, have provided exceptional cheatsheets and tutorials that serve as valuable resources for learning and mastering the functionalities of these packages.

Furthermore, there are several books available that serve as introductions to the tidyverse. For example:

- The tidyverse cookbook
- R for Data Science

Note

In this book, we assume that the reader already has experience in using the tidyverse. This prior knowledge and familiarity with the tidyverse tools enable a more efficient and focused exploration of the concepts presented throughout the book.

1.3.2. r2rtf

r2rtf is an R package specifically designed to create production-ready tables and figures in RTF format.

- Provide simple "verb" functions that correspond to each component of a table, to help you translate a data frame to a table in an RTF file.
- Enable pipes (`|>`).
- Focus on the **table format** only. Data manipulation and analysis tasks can be handled by other R packages like the tidyverse.

Before generating an RTF table using r2rtf, there are a few steps to follow:

- Determine the desired layout of the table.
- Break down the layout into smaller tasks, which can be programmed.
- Execute the program to generate the table.

We provide a brief introduction of r2rtf and show how to transfer data frames into table, listing, and figures (TLFs).

We provide a concise introduction to r2rtf and demonstrate how to convert data frames into TLFs. For more comprehensive examples and additional features, we encourage readers to explore the r2rtf package website.

To illustrate the basic usage of the r2rtf package, we will work with the "r2rtf_adae" dataset, available within the r2rtf package. This dataset contains information on adverse events (AEs) from a clinical trial, which will serve as a practical example for generating RTF tables using r2rtf.

To begin, let's load the required packages:

```
library(tidyverse) # Manipulate data
library(r2rtf) # Reporting in RTF format
```

In this example, we will focus on three variables from the `r2rtf_adae` dataset:

- `USUBJID`: unique subject identifier.
- `TRTA`: actual treatment group.

- `AEDECOD`: dictionary-derived derm.

> **Note**
>
> Additional information about these variables can be found on the help page of the dataset, which can be accessed by using the command `?r2rtf_adae` in R.

```
r2rtf_adae |> select(USUBJID, TRTA, AEDECOD)
#>       USUBJID    TRTA                    AEDECOD
#> 1 01-701-1015 Placebo APPLICATION SITE ERYTHEMA
#> 2 01-701-1015 Placebo APPLICATION SITE PRURITUS
#> 3 01-701-1015 Placebo                 DIARRHOEA
#> 4 01-701-1023 Placebo                  ERYTHEMA
#> #   1187 more rows
```

To manipulate the data and create a data frame containing the necessary information for the RTF table, we can use the dplyr and tidyr packages within the tidyverse.

```
tbl <- r2rtf_adae %>%
  count(TRTA, AEDECOD) %>%
  pivot_wider(names_from = TRTA, values_from = n, values_fill = 0)

tbl
#> # A tibble: 242 x 4
#>   AEDECOD        Placebo `Xanomeline High Dose` `Xanomeline Low Dose`
#>   <chr>            <int>                  <int>                 <int>
#> 1 ABDOMINAL PAIN       1                      2                     3
#> 2 AGITATION            2                      1                     2
#> 3 ALOPECIA             1                      0                     0
#> 4 ANXIETY              2                      0                     4
#> # i 238 more rows
```

Having prepared the dataset `tbl`, we can now proceed with constructing the final RTF table using the r2rtf package. The r2rtf package has various functions, each designed for a specific type of table layout. Some commonly used verbs include:

- `rtf_page()`: RTF page information
- `rtf_title()`: RTF title information
- `rtf_colheader()`: RTF column header information
- `rtf_body()`: RTF table body information
- `rtf_footnote()`: RTF footnote information
- `rtf_source()`: RTF data source information

Functions provided by the r2rtf package are designed to work seamlessly with the pipe operator (`|>`). This allows for a more concise and readable code structure, enhancing the efficiency of table creation in RTF format. A full list of functions in the r2rtf package can be found in the package reference page.

Here is a minimal example that demonstrates how to combine functions using pipes to create an RTF table.

- `rtf_body()` is used to define table body layout.
- `rtf_encode()` transfers table layout information into RTF syntax.
- `write_rtf()` save RTF encoding into a file with file extension `.rtf`.

```
tbl |>
  head() |>
  rtf_body() |>
  rtf_encode() |>
  write_rtf("tlf/intro-ae1.rtf")
```

AEDECOD	Placebo	Xanomeline High Dose	Xanomeline Low Dose
ABDOMINAL PAIN	1	2	3
AGITATION	2	1	2
ALOPECIA	1	0	0
ANXIETY	2	0	4
APPLICATION SITE DERMATITIS	9	12	15
APPLICATION SITE ERYTHEMA	3	23	20

In the previous example, we may want to add more column space to the first column. We can achieve the goal by updating the `col_rel_width argument` in the `rtf_body()` function.

In the example below, the `col_rel_width` argument expects a vector with the same length as the number of columns in the table `tbl`. This vector defines the relative width of each column within a predetermined total column width. Here, the relative width is defined as `3:2:2:2` that allow us to allocate more space to specific columns.

> **Note**
>
> Only the ratio of the `col_rel_width` values is considered. Therefore, using `col_rel_width = c(6, 4, 4, 4)` or `col_rel_width =`

```
tbl |>
  head() |>
  rtf_body(col_rel_width = c(3, 2, 2, 2)) |>
  rtf_encode() |>
  write_rtf("tlf/intro-ae2.rtf")
```

AEDECOD	Placebo	Xanomeline High Dose	Xanomeline Low Dose
ABDOMINAL PAIN	1	2	3
AGITATION	2	1	2
ALOPECIA	1	0	0
ANXIETY	2	0	4
APPLICATION SITE DERMATITIS	9	12	15
APPLICATION SITE ERYTHEMA	3	23	20

In the previous example, we encountered a misalignment issue with the column header. To address this, we can use the `rtf_colheader()` function to adjust column header width and provide more informative column headers.

Within the `rtf_colheader()` function, the `colheader` argument is used to specify the content of the column header. The columns are separated using the `|` symbol. In the following example, we define the column header as `"Adverse Events | Placebo | Xanomeline High Dose | Xanomeline Low Dose"`, representing the four columns in the table:

```
tbl |>
  head() |>
  rtf_colheader(
    colheader = "Adverse Events | Placebo | Xanomeline High Dose | Xanomeline Low Dose",
    col_rel_width = c(3, 2, 2, 2)
  ) |>
  rtf_body(col_rel_width = c(3, 2, 2, 2)) |>
  rtf_encode() %>%
  write_rtf("tlf/intro-ae3.rtf")
```

Adverse Events	Placebo	Xanomeline High Dose	Xanomeline Low Dose
ABDOMINAL PAIN	1	2	3
AGITATION	2	1	2
ALOPECIA	1	0	0
ANXIETY	2	0	4
APPLICATION SITE DERMATITIS	9	12	15
APPLICATION SITE ERYTHEMA	3	23	20

In `rtf_body()` and `rtf_colheader()`, the `text_justification` argument is used to align text within the generated RTF table. The default value is `"c"`, representing center justification. However, you can customize the text justification by column using a character vector with a length equal to the number of displayed columns. Here is a table displaying the possible inputs for the `text_justification` argument:

type	name
l	left
c	center
r	right
d	decimal
j	justified

Below is an example to make the first column left-aligned and the rest columns center-aligned.

```
tbl |>
  head() |>
  rtf_body(text_justification = c("l", "c", "c", "c")) |>
  rtf_encode() |>
  write_rtf("tlf/intro-ae5.rtf")
```

AEDECOD	Placebo	Xanomeline High Dose	Xanomeline Low Dose
ABDOMINAL PAIN	1	2	3
AGITATION	2	1	2
ALOPECIA	1	0	0
ANXIETY	2	0	4
APPLICATION SITE DERMATITIS	9	12	15
APPLICATION SITE ERYTHEMA	3	23	20

The `border_left`, `border_right`, `border_top`, and `border_bottom` arguments in the `rtf_body()` and `rtf_colheader()` functions are used to control the cell borders in the RTF table. If we want to remove the top border of `"Adverse Events"` in the header, we can change the default value `"single"` to `""` in the `border_top` argument. Below is an example to demonstrate the possibility of adding multiple column headers with proper border lines.

```
tbl |>
  head() |>
  rtf_colheader(
    colheader = " | Treatment",
    col_rel_width = c(3, 6)
  ) |>
  rtf_colheader(
    colheader = "Adverse Events | Placebo | Xanomeline High Dose | Xanomeline Low Dose",
    border_top = c("", "single", "single", "single"),
    col_rel_width = c(3, 2, 2, 2)
  ) |>
  rtf_body(col_rel_width = c(3, 2, 2, 2)) %>%
  rtf_encode() |>
  write_rtf("tlf/intro-ae7.rtf")
```

Adverse Events	Treatment		
	Placebo	Xanomeline High Dose	Xanomeline Low Dose
ABDOMINAL PAIN	1	2	3
AGITATION	2	1	2
ALOPECIA	1	0	0
ANXIETY	2	0	4
APPLICATION SITE DERMATITIS	9	12	15
APPLICATION SITE ERYTHEMA	3	23	20

The r2rtf R package website provides additional examples that demonstrate how to customize various aspects of the generated RTF tables. These examples cover topics such as customizing the title, subtitle, footnote, data source, and handling special characters within the table content.

In the upcoming chapters of this book, we will introduce and explore these features as they become relevant to the specific use cases and scenarios discussed. By following along with the chapters, readers will gradually learn how to leverage these features to customize and enhance their RTF tables in real examples.

2. Disposition

Following ICH E3 guidance, a summary table needs to be provided to include all participants who entered the study in Section 10.1, Disposition of Participants.

The disposition of participants table reports the numbers of participants who were randomized, and who entered and completed each phase of the study. In addition, the reasons for all post-randomization discontinuations, grouped by treatment and by major reason (lost to follow-up, adverse event, poor compliance, etc.) are reported.

```
library(haven) # Read SAS data
library(dplyr) # Manipulate data
library(tidyr) # Manipulate data
library(r2rtf) # Reporting in RTF format
```

In this chapter, we show how to create a typical disposition table.

2. Disposition

Disposition of Participants

	Placebo		Xanomeline Low Dose		Xanomeline High Dose	
	n	(%)	n	(%)	n	(%)
Participants in population	86		84		84	
Completed	58	67.4	25	29.8	27	32.1
Discontinued	28	32.6	59	70.2	57	67.9
Adverse Event	8	9.3	44	52.4	40	47.6
Death	2	2.3	1	1.2	0	0.0
I/E Not Met	1	1.2	0	0.0	2	2.4
Lack of Efficacy	3	3.5	0	0.0	1	1.2
Lost to Follow-up	1	1.2	1	1.2	0	0.0
Physician Decision	1	1.2	0	0.0	2	2.4
Protocol Violation	1	1.2	1	1.2	1	1.2
Sponsor Decision	2	2.3	2	2.4	3	3.6
Withdrew Consent	9	10.5	10	11.9	8	9.5

The first step is to read in the relevant datasets into R. For a disposition table, all the required information is saved in a Subject-level Analysis Dataset (ADSL). This dataset is provided in `sas7bdat` format, which is a SAS data format currently used in many clinical trial analysis and reporting. The `haven` package is able to read the dataset, while maintaining its attributes (e.g., variable labels).

```
adsl <- read_sas("data-adam/adsl.sas7bdat")
```

The following variables are used in the preparation of a simplified disposition of participants table:

- USUBJID: unique subject identifier
- TRT01P: planned treatment
- TRT01PN: planned treatment numeric encoding
- DISCONFL: discontinued from study flag
- DCREASCD: discontinued from study reason coded

```
adsl %>% select(USUBJID, TRT01P, TRT01PN, DISCONFL, DCREASCD)
#> # A tibble: 254 x 5
#>   USUBJID     TRT01P               TRT01PN DISCONFL DCREASCD
#>   <chr>       <chr>                  <dbl> <chr>    <chr>
#> 1 01-701-1015 Placebo                    0 ""       Completed
#> 2 01-701-1023 Placebo                    0 "Y"      Adverse Event
#> 3 01-701-1028 Xanomeline High Dose      81 ""       Completed
#> 4 01-701-1033 Xanomeline Low Dose       54 "Y"      Sponsor Decision
#> # i 250 more rows
```

In the code below, we calculate the number of participants in the analysis population by treatment arms.

```
n_rand <- adsl %>%
  group_by(TRT01PN) %>%
  summarize(n = n()) %>%
  pivot_wider(
    names_from = TRT01PN,
    names_prefix = "n_",
    values_from = n
  ) %>%
  mutate(row = "Participants in population")

n_rand
#> # A tibble: 1 x 4
#>     n_0  n_54  n_81 row
#>   <int> <int> <int> <chr>
#> 1    86    84    84 Participants in population
```

```
n_disc <- adsl %>%
  group_by(TRT01PN) %>%
  summarize(
    n = sum(DISCONFL == "Y"),
    pct = formatC(n / n() * 100,
      digits = 1, format = "f", width = 5
    )
  ) %>%
  pivot_wider(
    names_from = TRT01PN,
    values_from = c(n, pct)
  ) %>%
  mutate(row = "Discontinued")

n_disc
#> # A tibble: 1 x 7
#>     n_0  n_54  n_81 pct_0   pct_54  pct_81  row
```

```
#>   <int> <int> <int> <chr>   <chr>   <chr>   <chr>
#> 1    28    59    57 " 32.6" " 70.2" " 67.9" Discontinued
```

In the code below, we calculate the number and percentage of participants who completed/discontinued the study for different reasons by treatment arms.

```
n_reason <- adsl %>%
  group_by(TRT01PN) %>%
  mutate(n_total = n()) %>%
  group_by(TRT01PN, DCREASCD) %>%
  summarize(
    n = n(),
    pct = formatC(n / unique(n_total) * 100,
      digits = 1, format = "f", width = 5
    )
  ) %>%
  pivot_wider(
    id_cols = DCREASCD,
    names_from = TRT01PN,
    values_from = c(n, pct),
    values_fill = list(n = 0, pct = "  0.0")
  ) %>%
  rename(row = DCREASCD)

n_reason
#> # A tibble: 10 x 7
#>   row             n_0  n_54  n_81 pct_0   pct_54  pct_81
#>   <chr>         <int> <int> <int> <chr>   <chr>   <chr>
#> 1 Adverse Event     8    44    40 "  9.3" " 52.4" " 47.6"
#> 2 Completed        58    25    27 " 67.4" " 29.8" " 32.1"
#> 3 Death             2     1     0 "  2.3" "  1.2" "  0.0"
#> 4 I/E Not Met       1     0     2 "  1.2" "  0.0" "  2.4"
```

```
#> # i 6 more rows
```

In the code below, we calculate the number and percentage of participants who complete the study by treatment arms. We split `n_reason` because we want to customize the row order of the table.

```
n_complete <- n_reason %>%
  filter(row == "Completed")

n_complete
#> # A tibble: 1 x 7
#>   row         n_0  n_54  n_81 pct_0   pct_54  pct_81
#>   <chr>     <int> <int> <int> <chr>   <chr>   <chr>
#> 1 Completed    58    25    27 " 67.4" " 29.8" " 32.1"
```

In the code below, we calculate the numbers and percentages of participants who discontinued the study for different reasons by treatment arms. For display purpose, `paste0("    ", row)` is used to add leading spaces to produce indentation in the final report.

```
n_reason <- n_reason %>%
  filter(row != "Completed") %>%
  mutate(row = paste0("    ", row))

n_reason
#> # A tibble: 9 x 7
#>   row                     n_0  n_54  n_81 pct_0   pct_54  pct_81
#>   <chr>                 <int> <int> <int> <chr>   <chr>   <chr>
#> 1 "    Adverse Event"       8    44    40 "  9.3" " 52.4" " 47.6"
#> 2 "    Death"               2     1     0 "  2.3" "  1.2" "  0.0"
#> 3 "    I/E Not Met"         1     0     2 "  1.2" "  0.0" "  2.4"
#> 4 "    Lack of Efficacy"    3     0     1 "  3.5" "  0.0" "  1.2"
#> # i 5 more rows
```

Now we combine individual rows into one table for reporting purpose. `tbl_disp` is used as input for r2rtf to create final report.

```
tbl_disp <- bind_rows(n_rand, n_complete, n_disc, n_reason) %>%
  select(row, ends_with(c("_0", "_54", "_81")))

tbl_disp
#> # A tibble: 12 x 7
#>   row                          n_0 pct_0    n_54 pct_54   n_81 pct_81
#>   <chr>                      <int> <chr>   <int> <chr>   <int> <chr>
#> 1 "Participants in population"    86  <NA>      84  <NA>      84  <NA>
#> 2 "Completed"                     58 " 67.4"    25 " 29.8"    27 " 32.1"
#> 3 "Discontinued"                  28 " 32.6"    59 " 70.2"    57 " 67.9"
#> 4 "    Adverse Event"              8 "  9.3"    44 " 52.4"    40 " 47.6"
#> # i 8 more rows
```

In the below code, formatting of the final table is defined. Items that were not discussed in the previous sections, are highlighted below.

The `rtf_title` defines table title. We can provide a vector for the title argument. Each value is a separate line. The format can also be controlled by providing a vector input in text format.

```
tbl_disp %>%
  # Table title
  rtf_title("Disposition of Participants") %>%
  # First row of column header
  rtf_colheader(" | Placebo | Xanomeline Low Dose| Xanomeline High Dose",
    col_rel_width = c(3, rep(2, 3))
  ) %>%
  # Second row of column header
  rtf_colheader(" | n | (%) | n | (%) | n | (%)",
    col_rel_width = c(3, rep(c(0.7, 1.3), 3)),
    border_top = c("", rep("single", 6)),
```

```
  border_left = c("single", rep(c("single", ""), 3))
) %>%
# Table body
rtf_body(
  col_rel_width = c(3, rep(c(0.7, 1.3), 3)),
  text_justification = c("l", rep("c", 6)),
  border_left = c("single", rep(c("single", ""), 3))
) %>%
# Encoding RTF syntax
rtf_encode() %>%
# Save to a file
write_rtf("tlf/tbl_disp.rtf")
```

Disposition of Participants

	Placebo		Xanomeline Low Dose		Xanomeline High Dose	
	n	(%)	n	(%)	n	(%)
Participants in population	86		84		84	
Completed	58	67.4	25	29.8	27	32.1
Discontinued	28	32.6	59	70.2	57	67.9
Adverse Event	8	9.3	44	52.4	40	47.6
Death	2	2.3	1	1.2	0	0.0
I/E Not Met	1	1.2	0	0.0	2	2.4
Lack of Efficacy	3	3.5	0	0.0	1	1.2
Lost to Follow-up	1	1.2	1	1.2	0	0.0
Physician Decision	1	1.2	0	0.0	2	2.4
Protocol Violation	1	1.2	1	1.2	1	1.2
Sponsor Decision	2	2.3	2	2.4	3	3.6
Withdrew Consent	9	10.5	10	11.9	8	9.5

2. Disposition

The procedure to generate a disposition table can be summarized as follows:

- Step 1: Read subject level data (i.e., `adsl`) into R.
- Step 2: Count participants in the analysis population and name the dataset `n_rand`.
- Step 3: Calculate the number and percentage of participants who discontinued the study by treatment arm, and name the dataset `n_disc`.
- Step 4: Calculate the numbers and percentages of participants who discontinued the study for different reasons by treatment arm, and name the dataset `n_reason`.
- Step 5: Calculate the number and percentage of participants who completed the study by treatment arm, and name the dataset `n_complete`.
- Step 6: Bind `n_rand`, `n_disc`, `n_reason`, and `n_complete` by row.
- Step 7: Write the final table to RTF

3. Analysis population

Following ICH E3 guidance, we need to summarize the number of participants included in each efficacy analysis in Section 11.1, Data Sets Analysed.

```
library(haven) # Read SAS data
library(dplyr) # Manipulate data
library(tidyr) # Manipulate data
library(r2rtf) # Reporting in RTF format
```

In this chapter, we illustrate how to create a summary table for the analysis population for a study.

3. Analysis population

Summary of Analysis Sets
(All Participants Randomized)

	Placebo n (%)	Xanomeline line Low Dose n (%)	Xanomeline line High Dose n (%)
Participants in Population	86	84	84
Participants included in ITT population	86 (100.0)	84 (100.0)	84 (100.0)
Participants included in efficacy population	79 (91.9)	81 (96.4)	74 (88.1)
Participants included in safety population	86 (100.0)	84 (100.0)	84 (100.0)

The first step is to read relevant datasets into R. For the analysis population table, all the required information is saved in the ADSL dataset. We can use the `haven` package to read the dataset.

```
adsl <- read_sas("data-adam/adsl.sas7bdat")
```

We illustrate how to prepare a report data for a simplified analysis population table using variables below:

- USUBJID: unique subject identifier
- ITTFL: intent-to-treat population flag
- EFFFL: efficacy population flag
- SAFFL: safety population flag

```
adsl %>% select(USUBJID, ITTFL, EFFFL, SAFFL)
#> # A tibble: 254 x 4
#>   USUBJID     ITTFL EFFFL SAFFL
#>   <chr>       <chr> <chr> <chr>
#> 1 01-701-1015 Y     Y     Y
#> 2 01-701-1023 Y     Y     Y
#> 3 01-701-1028 Y     Y     Y
#> 4 01-701-1033 Y     Y     Y
#> # i 250 more rows
```

3.1. Helper functions

Before we write the analysis code, let's discuss the possibility of reusing R code by writing helper functions.

As discussed in R for data science, "You should consider writing a function whenever you've copied and pasted a block of code more than twice".

In Chapter 2, there are a few repeating steps to:

3. Analysis population

- Format the percentages using the `formatC()` function.
- Calculate the numbers and percentages by treatment arm.

We create two ad-hoc functions and use them to create the tables in the rest of this book.

To format numbers and percentages, we create a function called `fmt_num()`. It is a very simple function wrapping `formatC()`.

```
fmt_num <- function(x, digits, width = digits + 4) {
  formatC(
    x,
    digits = digits,
    format = "f",
    width = width
  )
}
```

The main reason to create the `fmt_num()` function is to enhance the readability of the analysis code.

For example, we can compare the two versions of code to format the percentage used in Chapter 2 and `fmt_num()`.

```
formatC(n / n() * 100,
  digits = 1, format = "f", width = 5
)
```

```
fmt_num(n / n() * 100, digits = 1)
```

To calculate the numbers and percentages of participants by groups, we provide a simple (but not robust) wrapper function, `count_by()`, using the dplyr and tidyr package.

The function can be enhanced in multiple ways, but here we only focus on simplicity and readability. More details about writing R functions can be found in the STAT 545 course.

```
count_by <- function(data, # Input data set
                     grp, # Group variable
                     var, # Analysis variable
                     var_label = var, # Analysis variable label
                     id = "USUBJID") { # Subject ID variable
  data <- data %>% rename(grp = !!grp, var = !!var, id = !!id)

  left_join(
    count(data, grp, var),
    count(data, grp, name = "tot"),
    by = "grp",
  ) %>%
    mutate(
      pct = fmt_num(100 * n / tot, digits = 1),
      n = fmt_num(n, digits = 0),
      npct = paste0(n, " (", pct, ")")
    ) %>%
    pivot_wider(
      id_cols = var,
      names_from = grp,
      values_from = c(n, pct, npct),
      values_fill = list(n = "0", pct = fmt_num(0, digits = 0))
    ) %>%
    mutate(var_label = var_label)
}
```

By using the `count_by()` function, we can simplify the analysis code as below.

```
count_by(adsl, "TRT01PN", "EFFFL") %>%
  select(-ends_with(c("_54", "_81")))
#> # A tibble: 2 x 5
#>   var   n_0    pct_0   npct_0          var_label
#>   <chr> <chr>  <chr>   <chr>           <chr>
#> 1 N     "   7" "  8.1" "   7 (  8.1)" EFFFL
#> 2 Y     "  79" " 91.9" "  79 ( 91.9)" EFFFL
```

3.2. Analysis code

With the helper function `count_by`, we can easily prepare a report dataset as

```
# Derive a randomization flag
adsl <- adsl %>% mutate(RANDFL = "Y")

pop <- count_by(adsl, "TRT01PN", "RANDFL",
  var_label = "Participants in Population"
) %>%
  select(var_label, starts_with("n_"))
```

```
pop1 <- bind_rows(
  count_by(adsl, "TRT01PN", "ITTFL",
    var_label = "Participants included in ITT population"
  ),
  count_by(adsl, "TRT01PN", "EFFFL",
    var_label = "Participants included in efficacy population"
  ),
  count_by(adsl, "TRT01PN", "SAFFL",
    var_label = "Participants included in safety population"
  )
```

```
) %>%
  filter(var == "Y") %>%
  select(var_label, starts_with("npct_"))
```

Now we combine individual rows into one table for reporting purpose. `tbl_pop` is used as input for r2rtf to create the final report.

```
names(pop) <- gsub("n_", "npct_", names(pop))
tbl_pop <- bind_rows(pop, pop1)

tbl_pop %>% select(var_label, npct_0)
#> # A tibble: 4 x 2
#>   var_label                                npct_0
#>   <chr>                                    <chr>
#> 1 Participants in Population               "  86"
#> 2 Participants included in ITT population  "  86 (100.0)"
#> 3 Participants included in efficacy population "  79 ( 91.9)"
#> 4 Participants included in safety population   "  86 (100.0)"
```

We define the format of the output using code below.

```
rel_width <- c(2, rep(1, 3))
colheader <- " | Placebo | Xanomeline line Low Dose| Xanomeline line High Dose"
tbl_pop %>%
  # Table title
  rtf_title(
    "Summary of Analysis Sets",
    "(All Participants Randomized)"
  ) %>%
  # First row of column header
  rtf_colheader(colheader,
    col_rel_width = rel_width
```

```
  ) %>%
  # Second row of column header
  rtf_colheader(" | n (%) | n (%) | n (%)",
    border_top = "",
    col_rel_width = rel_width
  ) %>%
  # Table body
  rtf_body(
    col_rel_width = rel_width,
    text_justification = c("l", rep("c", 3))
  ) %>%
  # Encoding RTF syntax
  rtf_encode() %>%
  # Save to a file
  write_rtf("tlf/tbl_pop.rtf")
```

Summary of Analysis Sets
(All Participants Randomized)

	Placebo n (%)	Xanomeline line Low Dose n (%)	Xanomeline line High Dose n (%)
Participants in Population	86	84	84
Participants included in ITT population	86 (100.0)	84 (100.0)	84 (100.0)
Participants included in efficacy population	79 (91.9)	81 (96.4)	74 (88.1)
Participants included in safety population	86 (100.0)	84 (100.0)	84 (100.0)

3. Analysis population

The procedure to generate an analysis population table can be summarized as follows:

- Step 1: Read data (i.e., `adsl`) into R.
- Step 2: Bind the counts/percentages of the ITT population, the efficacy population, and the safety population by row using the `count_by()` function.
- Step 3: Format the output from Step 2 using r2rtf.

4. Baseline characteristics

Following ICH E3 guidance, we need to summarize critical demographic and baseline characteristics of the participants in Section 11.2, Demographic and Other Baseline Characteristics.

In this chapter, we illustrate how to create a simplified baseline characteristics table for a study.

4. Baseline characteristics

Baseline Characteristics of Participants
(All Participants Randomized)

	Placebo (N=86)	Xanomeline High Dose (N=84)	Xanomeline Low Dose (N=84)	Overall (N=254)
SEX				
Female	53 (61.6%)	40 (47.6%)	50 (59.5%)	143 (56.3%)
Male	33 (38.4%)	44 (52.4%)	34 (40.5%)	111 (43.7%)
Age				
Mean (SD)	75.2 (8.59)	74.4 (7.89)	75.7 (8.29)	75.1 (8.25)
Median [Min, Max]	76.0 [52.0, 89.0]	76.0 [56.0, 88.0]	77.5 [51.0, 88.0]	77.0 [51.0, 89.0]
RACE				
Black or African American	8 (9.3%)	9 (10.7%)	6 (7.1%)	23 (9.1%)
White	78 (90.7%)	74 (88.1%)	78 (92.9%)	230 (90.6%)
American Indian or Alaska Native	0 (0%)	1 (1.2%)	0 (0%)	1 (0.4%)

There are many R packages that can efficiently summarize baseline information. The table1 R package is one of them.

```
library(table1)
library(r2rtf)
library(haven)
library(dplyr)
library(tidyr)
library(stringr)
library(tools)
```

As in previous chapters, we first read the `adsl` dataset that contains all the required information for the baseline characteristics table.

```
adsl <- read_sas("data-adam/adsl.sas7bdat")
```

For simplicity, we only analyze `SEX`, `AGE` and, `RACE` in this example using the table1 R package. More details of the table1 R package can be found in the package vignettes.

The table1 R package directly creates an HTML report.

```
ana <- adsl %>%
  mutate(
    SEX = factor(SEX, c("F", "M"), c("Female", "Male")),
    RACE = toTitleCase(tolower(RACE))
  )

tbl <- table1(~ SEX + AGE + RACE | TRT01P, data = ana)
tbl
```

	Placebo	Xanomeline High Dose	Xanome
	(N=86)	(N=84)	(N=84)
SEX			
Female	53 (61.6%)	40 (47.6%)	50 (59.5
Male	33 (38.4%)	44 (52.4%)	34 (40.5
Age			
Mean (SD)	75.2 (8.59)	74.4 (7.89)	75.7 (8.
Median [Min, Max]	76.0 [52.0, 89.0]	76.0 [56.0, 88.0]	77.5 [51
RACE			
Black or African American	8 (9.3%)	9 (10.7%)	6 (7.1%
White	78 (90.7%)	74 (88.1%)	78 (92.9
American Indian or Alaska Native	0 (0%)	1 (1.2%)	0 (0%)

The code below transfer the output into a dataframe that only contains ASCII characters recommended by regulatory agencies. `tbl_base` is used as input for r2rtf to create the final report.

```
tbl_base <- tbl %>%
  as.data.frame() %>%
  as_tibble() %>%
  mutate(across(
    everything(),
    ~ str_replace_all(.x, intToUtf8(160), " ")
  ))


names(tbl_base) <- str_replace_all(names(tbl_base), intToUtf8(160), " ")
tbl_base
#> # A tibble: 11 x 5
#>   ` `       Placebo     `Xanomeline High Dose` `Xanomeline Low Dose` Ove
#>   <chr>     <chr>       <chr>                  <chr>                 <ch
#> 1 ""        "(N=86)"    "(N=84)"               "(N=84)"              "(N
```

```
#> 2 "SEX"      ""           ""                       ""                    ""
#> 3 "  Female" "53 (61.6%)" "40 (47.6%)"           "50 (59.5%)"          "143 (56~
#> 4 "  Male"   "33 (38.4%)" "44 (52.4%)"           "34 (40.5%)"          "111 (43~
#> # i 7 more rows
```

We define the format of the output. We highlight items that are not discussed in previous discussion.

`text_indent_first` and `text_indent_left` are used to control the indent space of text. They are helpful when you need to control the white space of a long phrase, "AMERICAN INDIAN OR ALASKA NATIVE" in the table provides an example.

```
colheader1 <- paste(names(tbl_base), collapse = "|")
colheader2 <- paste(tbl_base[1, ], collapse = "|")
rel_width <- c(2.5, rep(1, 4))

tbl_base[-1, ] %>%
  rtf_title(
    "Baseline Characteristics of Participants",
    "(All Participants Randomized)"
  ) %>%
  rtf_colheader(colheader1,
    col_rel_width = rel_width
  ) %>%
  rtf_colheader(colheader2,
    border_top = "",
    col_rel_width = rel_width
  ) %>%
  rtf_body(
    col_rel_width = rel_width,
    text_justification = c("l", rep("c", 4)),
    text_indent_first = -240,
```

```
    text_indent_left = 180
  ) %>%
  rtf_encode() %>%
  write_rtf("tlf/tlf_base.rtf")
```

Baseline Characteristics of Participants
(All Participants Randomized)

	Placebo (N=86)	Xanomeline High Dose (N=84)	Xanomeline Low Dose (N=84)	Overall (N=254)
SEX				
Female	53 (61.6%)	40 (47.6%)	50 (59.5%)	143 (56.3%)
Male	33 (38.4%)	44 (52.4%)	34 (40.5%)	111 (43.7%)
Age				
Mean (SD)	75.2 (8.59)	74.4 (7.89)	75.7 (8.29)	75.1 (8.25)
Median [Min, Max]	76.0 [52.0, 89.0]	76.0 [56.0, 88.0]	77.5 [51.0, 88.0]	77.0 [51.0, 89.0]
RACE				
Black or African American	8 (9.3%)	9 (10.7%)	6 (7.1%)	23 (9.1%)
White	78 (90.7%)	74 (88.1%)	78 (92.9%)	230 (90.6%)
American Indian or Alaska Native	0 (0%)	1 (1.2%)	0 (0%)	1 (0.4%)

In conclusion, the procedure to generate demographic and baseline characteristics table is summarized as follows:

- Step 1: Read the data set.
- Step 2: Use `table1::table1()` to get the baseline characteristics table.
- Step 3: Transfer the output from Step 2 into a data frame that only contains ASCII characters.
- Step 4: Define the format of the RTF table by using the R package r2rtf.

5. Efficacy table

Following ICH E3 guidance, primary and secondary efficacy endpoints need to be summarized in Section 11.4, Efficacy Results and Tabulations of Individual Participant.

```
library(haven) # Read SAS data
library(dplyr) # Manipulate data
library(tidyr) # Manipulate data
library(r2rtf) # Reporting in RTF format
library(emmeans) # LS mean estimation
```

In this chapter, we illustrate how to generate an efficacy table for a study. For efficacy analysis, only the change from baseline glucose data at week 24 is analyzed.

5. *Efficacy table*

ANCOVA of Change from Baseline Glucose (mmol/L) at Week 24
LOCF
Efficacy Analysis Population

	Baseline		Week 24		Change from Baseline		
Treatment	N	Mean (SD)	N	Mean (SD)	N	Mean (SD)	LS Mean (95% CI)[a]
Placebo	79	5.7 (2.23)	57	5.7 (1.83)	57	-0.1 (2.68)	0.07 (-0.27, 0.41)
Xanomeline Low Dose	79	5.4 (0.95)	26	5.7 (1.26)	26	0.2 (0.82)	-0.11 (-0.45, 0.23)
Xanomeline High Dose	74	5.4 (1.37)	30	6.0 (1.92)	30	0.5 (1.94)	0.40 (0.05, 0.75)
Pairwise Comparison			Difference in LS Mean (95% CI)[a]				p-Value
Xanomeline Low Dose - Placebo			-0.17 (-0.65, 0.30)				0.757
Xanomeline High Dose - Placebo			0.33 (-0.16, 0.82)				0.381

[a]Based on an ANCOVA model after adjusting baseline value. LOCF approach is used to impute missing values.
ANCOVA = Analysis of Covariance, LOCF = Last Observation Carried Forward
CI = Confidence Interval, LS = Least Squares, SD = Standard Deviation

5.1. Analysis dataset

To prepare the analysis, both `adsl` and `adlbc` datasets are required.

```
adsl <- read_sas("data-adam/adsl.sas7bdat")
adlb <- read_sas("data-adam/adlbc.sas7bdat")
```

First, both the population and the data in scope are selected. The analysis is done on the efficacy population, identified by `EFFFL == "Y"`, and all records post baseline (`AVISITN >= 1`) and on or before Week 24 (`AVISITN <= 24`). Here the variable `AVISITN` is the numerical analysis visit. For example, if the analysis visit is recorded as "Baseline" (i.e., `AVISIT = Baseline`), `AVISITN = 0`; if the analysis visit is recorded as "Week 24" (i.e., `AVISIT = Week 24`), `AVISITN = 24`; if the analysis visit is blank, `AVISITN` is also blank. We will discuss these missing values in Section 6.4.

```
gluc <- adlb %>%
  left_join(adsl %>% select(USUBJID, EFFFL), by = "USUBJID") %>%
  # PARAMCD is parameter code and here we focus on Glucose (mg/dL)
  filter(EFFFL == "Y" & PARAMCD == "GLUC") %>%
  arrange(TRTPN) %>%
  mutate(TRTP = factor(TRTP, levels = unique(TRTP)))

ana <- gluc %>%
  filter(AVISITN > 0 & AVISITN <= 24) %>%
  arrange(AVISITN) %>%
  mutate(AVISIT = factor(AVISIT, levels = unique(AVISIT)))
```

Below is the first few records of the analysis dataset.

- AVAL: analysis value
- BASE: baseline value

5. *Efficacy table*

- CHG: change from baseline

```
ana %>% select(USUBJID, TRTPN, AVISIT, AVAL, BASE, CHG)
#> # A tibble: 1,377 x 6
#>   USUBJID     TRTPN AVISIT              AVAL  BASE     CHG
#>   <chr>       <dbl> <fct>              <dbl> <dbl>   <dbl>
#> 1 01-701-1015     0 "          Week 2"  4.66  4.72 -0.0555
#> 2 01-701-1023     0 "          Week 2"  5.77  5.33  0.444
#> 3 01-701-1047     0 "          Week 2"  5.55  5.55  0
#> 4 01-701-1118     0 "          Week 2"  4.88  4.05  0.833
#> # i 1,373 more rows
```

5.2. Helper functions

To prepare the report, we create a few helper functions by using the `fmt_num()` function defined in Chapter 3.

- Format estimators

```
fmt_num <- function(x, digits, width = digits + 4) {
  formatC(
    x,
    digits = digits,
    format = "f",
    width = width
  )
}
```

```
fmt_est <- function(.mean,
                    .sd,
                    digits = c(1, 2)) {
  .mean <- fmt_num(.mean, digits[1], width = digits[1] + 4)
```

```
  .sd <- fmt_num(.sd, digits[2], width = digits[2] + 3)
  paste0(.mean, " (", .sd, ")")
}
```

- Format confidence interval

```
fmt_ci <- function(.est,
                   .lower,
                   .upper,
                   digits = 2,
                   width = digits + 3) {
  .est <- fmt_num(.est, digits, width)
  .lower <- fmt_num(.lower, digits, width)
  .upper <- fmt_num(.upper, digits, width)
  paste0(.est, " (", .lower, ",", .upper, ")")
}
```

- Format p-value

```
fmt_pval <- function(.p, digits = 3) {
  scale <- 10^(-1 * digits)
  p_scale <- paste0("<", digits)
  if_else(.p < scale, p_scale, fmt_num(.p, digits = digits))
}
```

5.3. Summary of observed data

First the observed data at Baseline and Week 24 are summarized using code below:

```
t11 <- gluc %>%
  filter(AVISITN %in% c(0, 24)) %>%
  group_by(TRTPN, TRTP, AVISITN) %>%
  summarise(
    n = n(),
    mean_sd = fmt_est(mean(AVAL), sd(AVAL))
  ) %>%
  pivot_wider(
    id_cols = c(TRTP, TRTPN),
    names_from = AVISITN,
    values_from = c(n, mean_sd)
  )

t11
#> # A tibble: 3 x 6
#> # Groups:   TRTPN, TRTP [3]
#>   TRTP                TRTPN   n_0  n_24 mean_sd_0       mean_sd_24
#>   <fct>               <dbl> <int> <int> <chr>           <chr>
#> 1 Placebo                 0    79    57 "  5.7 ( 2.23)" "  5.7 ( 1.83)"
#> 2 Xanomeline Low Dose    54    79    26 "  5.4 ( 0.95)" "  5.7 ( 1.26)"
#> 3 Xanomeline High Dose   81    74    30 "  5.4 ( 1.37)" "  6.0 ( 1.92)"
```

Also the observed change from baseline glucose at Week 24 is summarized using code below:

```
t12 <- gluc %>%
  filter(AVISITN %in% 24) %>%
  group_by(TRTPN, AVISITN) %>%
  summarise(
    n_chg = n(),
    mean_chg = fmt_est(
      mean(CHG, na.rm = TRUE),
      sd(CHG, na.rm = TRUE)
```

```
    )
  )

t12
#> # A tibble: 3 x 4
#> # Groups:   TRTPN [3]
#>   TRTPN AVISITN n_chg mean_chg
#>   <dbl>   <dbl> <int> <chr>
#> 1     0      24    57 " -0.1 ( 2.68)"
#> 2    54      24    26 "  0.2 ( 0.82)"
#> 3    81      24    30 "  0.5 ( 1.94)"
```

5.4. Missing data imputation

In clinical trials, missing data is inevitable. In this study, there are missing values in glucose data.

```
count(ana, AVISIT)
#> # A tibble: 8 x 2
#>   AVISIT                n
#>   <fct>             <int>
#> 1 "          Week 2"   229
#> 2 "          Week 4"   211
#> 3 "          Week 6"   197
#> 4 "          Week 8"   187
#> # i 4 more rows
```

For simplicity and illustration purpose, we use the last observation carried forward (LOCF) approach to handle missing data. LOCF approach is a single imputation approach that is **not recommended** in real application. Interested readers can find more discussion on missing data approaches in

the book: The Prevention and Treatment of Missing Data in Clinical Trials.

```
ana_locf <- ana %>%
  group_by(USUBJID) %>%
  mutate(locf = AVISITN == max(AVISITN)) %>%
  filter(locf)
```

5.5. ANCOVA model

The imputed data is analyzed using the ANCOVA model with treatment and baseline glucose as covariates.

```
fit <- lm(CHG ~ BASE + TRTP, data = ana_locf)
summary(fit)
#>
#> Call:
#> lm(formula = CHG ~ BASE + TRTP, data = ana_locf)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -6.9907 -0.7195 -0.2367  0.2422  7.0754
#>
#> Coefficients:
#>                           Estimate Std. Error t value Pr(>|t|)
#> (Intercept)                3.00836    0.39392   7.637 6.23e-13 ***
#> BASE                      -0.53483    0.06267  -8.535 2.06e-15 ***
#> TRTPXanomeline Low Dose   -0.17367    0.24421  -0.711    0.478
#> TRTPXanomeline High Dose   0.32983    0.24846   1.327    0.186
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
```

```
#> Residual standard error: 1.527 on 226 degrees of freedom
#>   (2 observations deleted due to missingness)
#> Multiple R-squared:  0.2567, Adjusted R-squared:  0.2468
#> F-statistic: 26.01 on 3 and 226 DF,  p-value: 1.714e-14
```

The emmeans R package is used to obtain within and between group least square (LS) mean

```
fit_within <- emmeans(fit, "TRTP")
fit_within
#>  TRTP                 emmean    SE  df lower.CL upper.CL
#>  Placebo              0.0676 0.172 226   -0.272    0.407
#>  Xanomeline Low Dose  -0.1060 0.173 226   -0.447    0.235
#>  Xanomeline High Dose  0.3975 0.179 226    0.045    0.750
#>
#> Confidence level used: 0.95
```

```
t13 <- fit_within %>%
  as_tibble() %>%
  mutate(ls = fmt_ci(emmean, lower.CL, upper.CL)) %>%
  select(TRTP, ls)
t13
#> # A tibble: 3 x 2
#>   TRTP                 ls
#>   <fct>                <chr>
#> 1 Placebo              " 0.07 (-0.27, 0.41)"
#> 2 Xanomeline Low Dose  "-0.11 (-0.45, 0.23)"
#> 3 Xanomeline High Dose " 0.40 ( 0.05, 0.75)"
```

```
fit_between <- pairs(fit_within, reverse = TRUE)
fit_between
#>  contrast                                  estimate    SE  df t.ratio p.value
```

5. Efficacy table

```
#>  Xanomeline Low Dose - Placebo                -0.174 0.244 226  -0.711  0
#>  Xanomeline High Dose - Placebo                0.330 0.248 226   1.327  0
#>  Xanomeline High Dose - Xanomeline Low Dose    0.504 0.249 226   2.024  0
#>
#> P value adjustment: tukey method for comparing a family of 3 estimates
```

```
t2 <- fit_between %>%
  as_tibble() %>%
  mutate(
    ls = fmt_ci(
      estimate,
      estimate - 1.96 * SE,
      estimate + 1.96 * SE
    ),
    p = fmt_pval(p.value)
  ) %>%
  filter(stringr::str_detect(contrast, "- Placebo")) %>%
  select(contrast, ls, p)

t2
#> # A tibble: 2 x 3
#>   contrast                      ls                  p
#>   <chr>                         <chr>               <chr>
#> 1 Xanomeline Low Dose - Placebo  "-0.17 (-0.65, 0.30)" "  0.757"
#> 2 Xanomeline High Dose - Placebo " 0.33 (-0.16, 0.82)" "  0.381"
```

5.6. Reporting

`t11`, `t12` and `t13` are combined to get the first part of the report table

```
t1 <- cbind(
  t11 %>% ungroup() %>% select(TRTP, ends_with("0"), ends_with("24")),
  t12 %>% ungroup() %>% select(ends_with("chg")),
  t13 %>% ungroup() %>% select(ls)
)
t1
#>                   TRTP n_0     mean_sd_0 n_24    mean_sd_24 n_chg      mean_chg
#> 1              Placebo  79   5.7 ( 2.23)   57   5.7 ( 1.83)    57  -0.1 ( 2.68)
#> 2  Xanomeline Low Dose  79   5.4 ( 0.95)   26   5.7 ( 1.26)    26   0.2 ( 0.82)
#> 3 Xanomeline High Dose  74   5.4 ( 1.37)   30   6.0 ( 1.92)    30   0.5 ( 1.94)
#>                     ls
#> 1  0.07 (-0.27, 0.41)
#> 2 -0.11 (-0.45, 0.23)
#> 3  0.40 ( 0.05, 0.75)
```

Then r2rtf is used to prepare the table format for `t1`. We also highlight how to handle special characters in this example.

Special characters `^` and `_` are used to define superscript and subscript of text. And `{}` is to define the part that will be impacted. For example, `{^a}` provides a superscript `a` for footnote notation. r2rtf also supports most LaTeX characters. Examples can be found on the r2rtf get started page. The `text_convert` argument in `r2rtf_*()` functions controls whether to convert special characters.

```
t1_rtf <- t1 %>%
  data.frame() %>%
  rtf_title(c(
    "ANCOVA of Change from Baseline Glucose (mmol/L) at Week 24",
    "LOCF",
    "Efficacy Analysis Population"
  )) %>%
  rtf_colheader("| Baseline | Week 24 | Change from Baseline",
```

```
    col_rel_width = c(2.5, 2, 2, 4)
  ) %>%
  rtf_colheader(
    paste(
      "Treatment |",
      paste0(rep("N | Mean (SD) | ", 3), collapse = ""),
      "LS Mean (95% CI){^a}"
    ),
    col_rel_width = c(2.5, rep(c(0.5, 1.5), 3), 2)
  ) %>%
  rtf_body(
    text_justification = c("l", rep("c", 7)),
    col_rel_width = c(2.5, rep(c(0.5, 1.5), 3), 2)
  ) %>%
  rtf_footnote(c(
    "{^a}Based on an ANCOVA model after adjusting baseline value. LOCF approa
    "ANCOVA = Analysis of Covariance, LOCF = Last Observation Carried Forwar
    "CI = Confidence Interval, LS = Least Squares, SD = Standard Deviation"
  ))

t1_rtf %>%
  rtf_encode() %>%
  write_rtf("tlf/tlf_eff1.rtf")
```

ANCOVA of Change from Baseline Glucose (mmol/L) at Week 24
LOCF
Efficacy Analysis Population

	Baseline		Week 24		Change from Baseline		
Treatment	N	Mean (SD)	N	Mean (SD)	N	Mean (SD)	LS Mean (95% CI)[a]
Placebo	79	5.7 (2.23)	57	5.7 (1.83)	57	-0.1 (2.68)	0.07 (-0.27, 0.41)
Xanomeline Low Dose	79	5.4 (0.95)	26	5.7 (1.26)	26	0.2 (0.82)	-0.11 (-0.45, 0.23)
Xanomeline High Dose	74	5.4 (1.37)	30	6.0 (1.92)	30	0.5 (1.94)	0.40 (0.05, 0.75)

[a]Based on an ANCOVA model after adjusting baseline value. LOCF approach is used to impute missing values.
ANCOVA = Analysis of Covariance, LOCF = Last Observation Carried Forward
CI = Confidence Interval, LS = Least Squares, SD = Standard Deviation

5. Efficacy table

We also use r2rtf to prepare the table format for `t2`

```
t2_rtf <- t2 %>%
  data.frame() %>%
  rtf_colheader("Pairwise Comparison | Difference in LS Mean (95% CI){^a} | p
    col_rel_width = c(4.5, 4, 2)
  ) %>%
  rtf_body(
    text_justification = c("l", "c", "c"),
    col_rel_width = c(4.5, 4, 2)
  )

t2_rtf %>%
  rtf_encode() %>%
  write_rtf("tlf/tlf_eff2.rtf")
```

Pairwise Comparison	Difference in LS Mean (95% CI)[a]	p-Value
Xanomeline Low Dose - Placebo	-0.17 (-0.65, 0.30)	0.757
Xanomeline High Dose - Placebo	0.33 (-0.16, 0.82)	0.381

5. Efficacy table

Finally, we combine the two parts to get the final table using r2rtf. This is achieved by providing a list of `t1_rtf` and `t2_rtf` as input for `rtf_encode`.

```
list(t1_rtf, t2_rtf) %>%
  rtf_encode() %>%
  write_rtf("tlf/tlf_eff.rtf")
```

ANCOVA of Change from Baseline Glucose (mmol/L) at Week 24
LOCF
Efficacy Analysis Population

	Baseline		Week 24		Change from Baseline		
Treatment	N	Mean (SD)	N	Mean (SD)	N	Mean (SD)	LS Mean (95% CI)[a]
Placebo	79	5.7 (2.23)	57	5.7 (1.83)	57	-0.1 (2.68)	0.07 (-0.27, 0.41)
Xanomeline Low Dose	79	5.4 (0.95)	26	5.7 (1.26)	26	0.2 (0.82)	-0.11 (-0.45, 0.23)
Xanomeline High Dose	74	5.4 (1.37)	30	6.0 (1.92)	30	0.5 (1.94)	0.40 (0.05, 0.75)

Pairwise Comparison	Difference in LS Mean (95% CI)[a]	p-Value
Xanomeline Low Dose - Placebo	-0.17 (-0.65, 0.30)	0.757
Xanomeline High Dose - Placebo	0.33 (-0.16, 0.82)	0.381

[a]Based on an ANCOVA model after adjusting baseline value. LOCF approach is used to impute missing values.
ANCOVA = Analysis of Covariance, LOCF = Last Observation Carried Forward
CI = Confidence Interval, LS = Least Squares, SD = Standard Deviation

In conclusion, the procedure to generate the above efficacy results table is summarized as follows.

- Step 1: Read the data (i.e., `adsl` and `adlb`) into R.
- Step 2: Define the analysis dataset. In this example, we define two analysis datasets. The first dataset is the efficacy population (`gluc`). The second dataset is the collection of all records post baseline and on or before week 24 (`ana`).
- Step 3: Impute the missing values. In this example, we name the `ana` dataset after imputation as `ana_locf`.
- Step 4: Calculate the mean and standard derivation of efficacy endpoint (i.e., `gluc`), and then format it into an RTF table.
- Step 5: Calculate the pairwise comparison by ANCOVA model, and then format it into an RTF table.
- Step 6: Combine the outputs from steps 4 and 5 by rows.

6. Efficacy figure

Following the ICH E3 guidance, primary and secondary efficacy endpoints need to be summarized in Section 11.4, Efficacy Results and Tabulations of Individual Participant.

```
library(haven) # Read SAS data
library(dplyr) # Manipulate data
library(r2rtf) # Reporting in RTF format
library(survival) # Fit survival model
```

In this chapter, we illustrate how to create a simplified Kaplan-Meier plot in a study. For the survival analysis in efficacy, time to dermatologic event (TTDE) will be analyzed.

> **Note**
>
> R packages such as visR and survminer can create more informative Kaplan-Meier plots. Interested readers can find examples on their websites.

6.1. Analysis dataset

To prepare the analysis, the `adtte` dataset is required.

```
adtte <- read_sas("data-adam/adtte.sas7bdat")
```

First, to prepare the analysis ready data, filter all records for the efficacy endpoint of time to event of interest (`TTDE`) using `PARAMCD` (or `PARAM`, `PRAMN`), then select the survival analysis related variables:

- `TRTP`: treatment arm (using corresponding numeric code `TRTAN` to re-order the levels, "Placebo" will be the reference level)
- `AVAL`: time-to-event analysis value
- `CNSR`: event (censoring) status

```
adtte_ttde <- adtte %>%
  filter(PARAMCD == "TTDE") %>%
  select(TRTP, TRTAN, AVAL, CNSR) %>%
  mutate(
    TRTP = forcats::fct_reorder(TRTP, TRTAN), # Recorder levels
    AVAL_m = AVAL / 30.4367 # Convert Day to Month
  )
```

6.2. Create Kaplan-Meier curve

The survival package is used to obtain the K-M estimate.

```
# Fit survival model, convert the time value from Days to Month
fit <- survfit(Surv(AVAL_m, 1 - CNSR) ~ TRTP, data = adtte_ttde)
```

We save the simplified K-M plot into a `.png` file using code below.

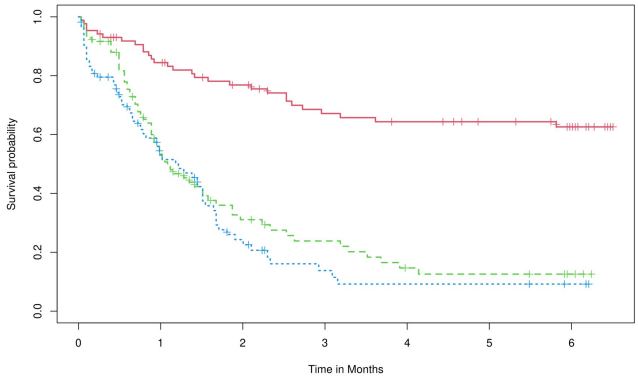
```
# Save as a PNG file
png(
  file = "tlf/fig_km.png",
  width = 3000,
  height = 2000,
  res = 300
)

plot(
  fit,
  xlab = "Time in Months",
  ylab = "Survival probability",
  mark.time = TRUE,
  lwd = 2,
  col = c(2, 3, 4),
  lty = c(1, 2, 3)
)

dev.off()
```

Now, we can use the r2rtf package to create a formatted RTF figure. More details can be found on the r2rtf website.

```
# Create RTF figure
rtf_read_figure("tlf/fig_km.png") %>% # Read the PNG file from the file path
  rtf_title(
    "Kaplan-Meier Plot for Time to First Dermatologic Event by Treatment Group",
    "All Participants"
  ) %>% # Add title or subtitle
  rtf_footnote("footnote") %>% # Add footnote
  rtf_source("[datasource: adam-adtte]") %>% # Add data source
  rtf_figure(fig_width = 6, fig_height = 4) %>% # Set proportional figure size to the origin
  rtf_encode(doc_type = "figure") %>% # Encode figure as rtf
```

```
write_rtf(file = "tlf/tlf_km.rtf")
```

Kaplan-Meier Plot for Time to First Dermatologic Event by Treatment Group
All Participants



footnote

[datasource: adam-adtte]

In conclusion, the steps to create a K-M plot are as follows.

- Step 1: Read the data `adtte` into R.
- Step 2: Define the analysis-ready dataset. In this example, we define the analysis dataset for the TTDE endpoint `adtte_ttde`.
- Step 3: Save figures into `png` files based on required analysis specification.
- Step 4: Create RTF output using the r2rtf package.

7. AE summary

Following ICH E3 guidance, we summarize number of participants that were included in each safety analysis in Section 12.2, Adverse Events (AEs).

```
library(haven) # Read SAS data
library(dplyr) # Manipulate data
library(tidyr) # Manipulate data
library(r2rtf) # Reporting in RTF format
```

In this chapter, we illustrate how to summarize AEs information for a study.

7. AE summary

Analysis of Adverse Event Summary
(Safety Analysis Population)

	Placebo		Xanomeline Low Dose		Xanomeline High Dose	
	n	(%)	n	(%)	n	(%)
Participants in population	86		84		84	
With one or more adverse events	69	(80.2)	77	(91.7)	79	(94.0)
With drug-related adverse events	44	(51.2)	73	(86.9)	70	(83.3)
With serious adverse events	0	(0.0)	1	(1.2)	2	(2.4)
With serious drug-related adverse events	0	(0.0)	1	(1.2)	1	(1.2)
Who died	2	(2.3)	1	(1.2)	0	(0.0)
Every subject is counted a single time for each applicable row and column.						

The data used to summarize AE information is in `adsl` and `adae` datasets.

```
adsl <- read_sas("data-adam/adsl.sas7bdat")
adae <- read_sas("data-adam/adae.sas7bdat")
```

We first summarize participants in population by treatment arm.

```
pop <- adsl %>%
  filter(SAFFL == "Y") %>%
  rename(TRTAN = TRT01AN) %>%
  count(TRTAN, name = "tot")

pop
#> # A tibble: 3 x 2
#>   TRTAN   tot
#>   <dbl> <int>
#> 1     0    86
#> 2    54    84
#> 3    81    84
```

We transform the data to simplify the analysis of each required AE criteria of interest.

- With one or more adverse events
- With drug-related adverse events
- With serious adverse events
- With serious drug-related adverse events
- Who died

```
tidy_ae <- adae %>%
  mutate(
    all = SAFFL == "Y",
```

```
    drug = AEREL %in% c("POSSIBLE", "PROBABLE"),
    ser = AESER == "Y",
    drug_ser = drug & ser,
    die = AEOUT == "FATAL"
  ) %>%
  select(USUBJID, TRTAN, all, drug, ser, drug_ser, die) %>%
  pivot_longer(cols = c(all, drug, ser, drug_ser, die))

tidy_ae
#> # A tibble: 5,955 x 4
#>   USUBJID     TRTAN name     value
#>   <chr>       <dbl> <chr>    <lgl>
#> 1 01-701-1015     0 all      TRUE
#> 2 01-701-1015     0 drug     TRUE
#> 3 01-701-1015     0 ser      FALSE
#> 4 01-701-1015     0 drug_ser FALSE
#> # i 5,951 more rows
```

We summarize the number and percentage of participants who meet each AE criteria.

```
fmt_num <- function(x, digits, width = digits + 4) {
  formatC(
    x,
    digits = digits,
    format = "f",
    width = width
  )
}
```

```
ana <- tidy_ae %>%
  filter(value == TRUE) %>%
  group_by(TRTAN, name) %>%
```

```
  summarise(n = n_distinct(USUBJID)) %>%
  left_join(pop, by = "TRTAN") %>%
  mutate(
    pct = fmt_num(n / tot * 100, digits = 1),
    n = fmt_num(n, digits = 0),
    pct = paste0("(", pct, ")")
  )

ana
#> # A tibble: 12 x 5
#> # Groups:   TRTAN [3]
#>    TRTAN name     n        tot pct
#>    <dbl> <chr>    <chr>  <int> <chr>
#>  1     0 all      "  69"    86 ( 80.2)
#>  2     0 die      "   2"    86 (  2.3)
#>  3     0 drug     "  44"    86 ( 51.2)
#>  4    54 all      "  77"    84 ( 91.7)
#>  5    54 die      "   1"    84 (  1.2)
#>  6    54 drug     "  73"    84 ( 86.9)
#>  7    54 drug_ser "   1"    84 (  1.2)
#>  8    54 ser      "   1"    84 (  1.2)
#>  9    81 all      "  79"    84 ( 94.0)
#> 10    81 drug     "  70"    84 ( 83.3)
#> 11    81 drug_ser "   1"    84 (  1.2)
#> 12    81 ser      "   2"    84 (  2.4)
```

We prepare reporting-ready dataset for each AE group.

```
t_ae <- ana %>%
  pivot_wider(
    id_cols = "name",
    names_from = TRTAN,
    values_from = c(n, pct),
```

```
    values_fill = list(
      n = "   0",
      pct = "(  0.0)"
    )
  )

t_ae <- t_ae %>%
  mutate(name = factor(
    name,
    c("all", "drug", "ser", "drug_ser", "die"),
    c(
      "With one or more adverse events",
      "With drug-related adverse events",
      "With serious adverse events",
      "With serious drug-related adverse events",
      "Who died"
    )
  )) %>%
  arrange(name)
```

We prepare reporting-ready dataset for the analysis population.

```
t_pop <- pop %>%
  mutate(
    name = "Participants in population",
    tot = fmt_num(tot, digits = 0)
  ) %>%
  pivot_wider(
    id_cols = name,
    names_from = TRTAN,
    names_prefix = "n_",
    values_from = tot
  )
```

```
t_pop
#> # A tibble: 1 x 4
#>   name                       n_0    n_54   n_81
#>   <chr>                      <chr>  <chr>  <chr>
#> 1 Participants in population "  86" "  84" "  84"
```

The final report data is saved in `tbl_ae_summary`.

```
tbl_ae_summary <- bind_rows(t_pop, t_ae) %>%
  select(name, ends_with("_0"), ends_with("_54"), ends_with("_81"))

tbl_ae_summary
#> # A tibble: 6 x 7
#>   name                            n_0    pct_0   n_54   pct_54  n_81   pct_81
#>   <chr>                           <chr>  <chr>   <chr>  <chr>   <chr>  <chr>
#> 1 Participants in population      "  86" <NA>    "  84" <NA>    "  84" <NA>
#> 2 With one or more adverse events "  69" ( 80.2) "  77" ( 91.7) "  79" ( 94.0)
#> 3 With drug-related adverse events "  44" ( 51.2) "  73" ( 86.9) "  70" ( 83.3)
#> 4 With serious adverse events     "   0" (  0.0) "   1" (  1.2) "   2" (  2.4)
#> # i 2 more rows
```

We define the format of the output using code below:

```
tbl_ae_summary %>%
  rtf_title(
    "Analysis of Adverse Event Summary",
    "(Safety Analysis Population)"
  ) %>%
  rtf_colheader(" | Placebo | Xanomeline Low Dose| Xanomeline High Dose",
    col_rel_width = c(3.5, rep(2, 3))
  ) %>%
  rtf_colheader(" | n | (%) | n | (%) | n | (%)",
```

```
    col_rel_width = c(3.5, rep(c(0.7, 1.3), 3)),
    border_top = c("", rep("single", 6)),
    border_left = c("single", rep(c("single", ""), 3))
  ) %>%
  rtf_body(
    col_rel_width = c(3.5, rep(c(0.7, 1.3), 3)),
    text_justification = c("l", rep("c", 6)),
    border_left = c("single", rep(c("single", ""), 3))
  ) %>%
  rtf_footnote("Every subject is counted a single time for each applicable ro
  rtf_encode() %>%
  write_rtf("tlf/tlf_ae_summary.rtf")
```

Analysis of Adverse Event Summary
(Safety Analysis Population)

	Placebo		Xanomeline Low Dose		Xanomeline High Dose	
	n	(%)	n	(%)	n	(%)
Participants in population	86		84		84	
With one or more adverse events	69	(80.2)	77	(91.7)	79	(94.0)
With drug-related adverse events	44	(51.2)	73	(86.9)	70	(83.3)
With serious adverse events	0	(0.0)	1	(1.2)	2	(2.4)
With serious drug-related adverse events	0	(0.0)	1	(1.2)	1	(1.2)
Who died	2	(2.3)	1	(1.2)	0	(0.0)
Every subject is counted a single time for each applicable row and column.						

7. AE summary

The procedure to generate an AE summary table can be summarized as follows:

- Step 1: Read data (i.e., `adae` and `adsl`) into R.
- Step 2: Summarize participants in population by treatment arm, and name the dataset as `t_pop`.
- Step 3: Summarize participants in population by required AE criteria of interest, and name the dataset as `t_ae`.
- Step 4: Row-wise combine `t_pop` and `t_ae` and format it by using r2rtf.

8. Specific AE

Following ICH E3 guidance, we need to summarize number of participants for each specific AE in Section 12.2, Adverse Events (AEs).

```
library(haven) # Read SAS data
library(dplyr) # Manipulate data
library(tidyr) # Manipulate data
library(r2rtf) # Reporting in RTF format
```

In this chapter, we illustrate how to summarize simplified specific AE information for a study.

Analysis of Participants With Specific Adverse Events
(Safety Analysis Population)

	Placebo n	Xanomeline Low Dose n	Xanomeline High Dose n
Participants in population	86	84	84
Cardiac Disorders	**13**	**13**	**18**
Atrial Fibrillation	1	1	3
Atrial Flutter	0	1	1
Atrial Hypertrophy	1	0	0
Atrioventricular Block First Degree	1	1	0
Atrioventricular Block Second Degree	2	0	3
Bradycardia	1	0	0
Bundle Branch Block Left	1	0	0
Bundle Branch Block Right	1	1	0
Cardiac Disorder	0	0	1
Cardiac Failure Congestive	1	0	0
Myocardial Infarction	4	2	4
Palpitations	0	2	0
Sinus Arrhythmia	1	0	0
Sinus Bradycardia	2	7	8
Supraventricular Extrasystoles	1	1	1
Supraventricular Tachycardia	0	1	0
Tachycardia	1	0	0
Ventricular Extrasystoles	0	2	1
Ventricular Hypertrophy	1	0	0
Wolff-Parkinson-White Syndrome	0	1	0
Congenital, Familial and Genetic Disorders	**0**	**1**	**2**
Ventricular Septal Defect	0	1	2
Ear and Labyrinth Disorders	**1**	**2**	**1**
Cerumen Impaction	0	1	0
Ear Pain	1	0	0
Tinnitus	0	1	0
Vertigo	0	1	1
Eye Disorders	**4**	**2**	**1**
Conjunctival Haemorrhage	0	1	0
Conjunctivitis	2	0	0
Eye Allergy	1	0	0

The data used to summarize AE information is in `adsl` and `adae` datasets.

```
adsl <- read_sas("data-adam/adsl.sas7bdat")
adae <- read_sas("data-adam/adae.sas7bdat")
```

For illustration purpose, we only provide counts in the simplified table. The percentage of participants for each AE can be calculated as shown in Chapter 7.

Here, we focus on the analysis script for two advanced features for a table layout.

- group content: AE can be summarized in multiple nested layers. (e.g., by system organ class (SOC, `AESOC`) and specific AE term (`AEDECOD`))
- pagenization: there are many AE terms that can not be covered in one page. Column headers and SOC information need to be repeated on every page.

In the code below, we count the number of participants in each AE term by SOC and treatment arm, and we create a new variable `order` and set it as `0`. The variable `order` can help with the data manipulation later.

```
fmt_num <- function(x, digits, width = digits + 4) {
  formatC(
    x,
    digits = digits,
    format = "f",
    width = width
  )
}
```

```
ana <- adae %>%
  mutate(
    AESOC = tools::toTitleCase(tolower(AESOC)),
    AEDECOD = tools::toTitleCase(tolower(AEDECOD))
  )

t1 <- ana %>%
  group_by(TRTAN, AESOC) %>%
  summarise(n = fmt_num(n_distinct(USUBJID), digits = 0)) %>%
  mutate(AEDECOD = AESOC, order = 0)

t1
#> # A tibble: 61 x 5
#> # Groups:   TRTAN [3]
#>    TRTAN AESOC                                                n     AEDECOD
#>    <dbl> <chr>                                                <chr> <chr>
#>  1     0 Cardiac Disorders                                    "  1~ Cardia~
#>  2     0 Ear and Labyrinth Disorders                          "    ~ Ear an~
#>  3     0 Eye Disorders                                        "    ~ Eye Di~
#>  4     0 Gastrointestinal Disorders                           "  1~ Gastro~
#>  5     0 General Disorders and Administration Site Conditio~ "  2~ Genera~
#>  6     0 Hepatobiliary Disorders                              "    ~ Hepato~
#>  7     0 Infections and Infestations                          "  1~ Infect~
#>  8     0 Injury, Poisoning and Procedural Complications       "    ~ Injury~
#>  9     0 Investigations                                       "  1~ Invest~
#> 10     0 Metabolism and Nutrition Disorders                   "    ~ Metabo~
#> # i 51 more rows
```

In the code below, we count the number of subjects in each AE term by SOC, AE term, and treatment arm. Here we also create a new variable `order` and set it as `1`.

```r
t2 <- ana %>%
  group_by(TRTAN, AESOC, AEDECOD) %>%
  summarise(n = fmt_num(n_distinct(USUBJID), digits = 0)) %>%
  mutate(order = 1)

t2
#> # A tibble: 373 x 5
#> # Groups:   TRTAN, AESOC [61]
#>    TRTAN AESOC             AEDECOD                              n      order
#>    <dbl> <chr>             <chr>                                <chr>  <dbl>
#>  1     0 Cardiac Disorders Atrial Fibrillation                  "   1"     1
#>  2     0 Cardiac Disorders Atrial Hypertrophy                   "   1"     1
#>  3     0 Cardiac Disorders Atrioventricular Block First Degree  "   1"     1
#>  4     0 Cardiac Disorders Atrioventricular Block Second Degree "   2"     1
#>  5     0 Cardiac Disorders Bradycardia                          "   1"     1
#>  6     0 Cardiac Disorders Bundle Branch Block Left             "   1"     1
#>  7     0 Cardiac Disorders Bundle Branch Block Right            "   1"     1
#>  8     0 Cardiac Disorders Cardiac Failure Congestive           "   1"     1
#>  9     0 Cardiac Disorders Myocardial Infarction                "   4"     1
#> 10     0 Cardiac Disorders Sinus Arrhythmia                     "   1"     1
#> # i 363 more rows
```

We prepare reporting data for AE information using code below:

```r
t_ae <- bind_rows(t1, t2) %>%
  pivot_wider(
    id_cols = c(AESOC, order, AEDECOD),
    names_from = TRTAN,
    names_prefix = "n_",
    values_from = n,
    values_fill = fmt_num(0, digits = 0)
  ) %>%
  arrange(AESOC, order, AEDECOD) %>%
```

```
  select(AESOC, AEDECOD, starts_with("n"))

t_ae
#> # A tibble: 265 x 5
#>   AESOC             AEDECOD             n_0    n_54   n_81
#>   <chr>             <chr>               <chr>  <chr>  <chr>
#> 1 Cardiac Disorders Cardiac Disorders   "  13" "  13" "  18"
#> 2 Cardiac Disorders Atrial Fibrillation "   1" "   1" "   3"
#> 3 Cardiac Disorders Atrial Flutter      "   0" "   1" "   1"
#> 4 Cardiac Disorders Atrial Hypertrophy  "   1" "   0" "   0"
#> # i 261 more rows
```

We prepare reporting data for analysis population using code below:

```
count_by <- function(data, # Input data set
                     grp, # Group variable
                     var, # Analysis variable
                     var_label = var, # Analysis variable label
                     id = "USUBJID") { # Subject ID variable
  data <- data %>% rename(grp = !!grp, var = !!var, id = !!id)

  left_join(
    count(data, grp, var),
    count(data, grp, name = "tot"),
    by = "grp",
  ) %>%
    mutate(
      pct = fmt_num(100 * n / tot, digits = 1),
      n = fmt_num(n, digits = 0),
      npct = paste0(n, " (", pct, ")")
    ) %>%
    pivot_wider(
      id_cols = var,
```

```
      names_from = grp,
      values_from = c(n, pct, npct),
      values_fill = list(n = "0", pct = fmt_num(0, digits = 0))
    ) %>%
    mutate(var_label = var_label)
}
```

```
t_pop <- adsl %>%
  filter(SAFFL == "Y") %>%
  count_by("TRT01AN", "SAFFL",
    var_label = "Participants in population"
  ) %>%
  mutate(
    AESOC = "pop",
    AEDECOD = var_label
  ) %>%
  select(AESOC, AEDECOD, starts_with("n_"))

t_pop
#> # A tibble: 1 x 5
#>   AESOC AEDECOD                    n_0    n_54   n_81
#>   <chr> <chr>                      <chr>  <chr>  <chr>
#> 1 pop   Participants in population "  86" "  84" "  84"
```

The final report data is saved in `tbl_ae_spec`. We also add a blank row between population and AE information in the reporting table.

```
tbl_ae_spec <- bind_rows(
  t_pop,
  data.frame(AESOC = "pop"),
  t_ae
) %>%
  mutate(AEDECOD = ifelse(AEDECOD == AESOC,
```

```
    AEDECOD, paste0("  ", AEDECOD)
  ))

tbl_ae_spec
#> # A tibble: 267 x 5
#>   AESOC             AEDECOD                        n_0    n_54   n_81
#>   <chr>             <chr>                          <chr>  <chr>  <chr>
#> 1 pop               "  Participants in population" "  86" "  84" "  84"
#> 2 pop                <NA>                           <NA>   <NA>   <NA>
#> 3 Cardiac Disorders "Cardiac Disorders"           "  13" "  13" "  18"
#> 4 Cardiac Disorders "  Atrial Fibrillation"       "   1" "   1" "   3"
#> # i 263 more rows
```

We define the format of the output as below:

To obtain the nested layout, we use the `page_by` argument in the `rtf_body` function. By defining `page_by="AESOC"`, r2rtf recognizes the variable as a group indicator.

After setting `pageby_row = "first_row"`, the first row is displayed as group header. If a group of information is broken into multiple pages, the group header row is repeated on each page by default.

We can also customize the text format by providing a matrix that has the same dimension as the input dataset (i.e., `tbl_ae_spec`). In the code below, we illustrate how to display **bold** text for group headers to highlight the nested structure of the table layout.

```
n_row <- nrow(tbl_ae_spec)
n_col <- ncol(tbl_ae_spec)
id <- tbl_ae_spec$AESOC == tbl_ae_spec$AEDECOD
id <- ifelse(is.na(id), FALSE, id)

text_format <- ifelse(id, "b", "")
```

```
tbl_ae_spec %>%
  rtf_title(
    "Analysis of Participants With Specific Adverse Events",
    "(Safety Analysis Population)"
  ) %>%
  rtf_colheader(" | Placebo | Xanomeline Low Dose| Xanomeline High Dose",
    col_rel_width = c(3, rep(1, 3))
  ) %>%
  rtf_colheader(" | n |  n | n ",
    border_top = "",
    border_bottom = "single",
    col_rel_width = c(3, rep(1, 3))
  ) %>%
  rtf_body(
    col_rel_width = c(1, 3, rep(1, 3)),
    text_justification = c("l", "l", rep("c", 3)),
    text_format = matrix(text_format, nrow = n_row, ncol = n_col),
    page_by = "AESOC",
    pageby_row = "first_row"
  ) %>%
  rtf_footnote("Every subject is counted a single time for each applicable row and column.")
  rtf_encode() %>%
  write_rtf("tlf/tlf_spec_ae.rtf")
```

Analysis of Participants With Specific Adverse Events
(Safety Analysis Population)

	Placebo n	Xanomeline Low Dose n	Xanomeline High Dose n
Participants in population	86	84	84
Cardiac Disorders	**13**	**13**	**18**
Atrial Fibrillation	1	1	3
Atrial Flutter	0	1	1
Atrial Hypertrophy	1	0	0
Atrioventricular Block First Degree	1	1	0
Atrioventricular Block Second Degree	2	0	3
Bradycardia	1	0	0
Bundle Branch Block Left	1	0	0
Bundle Branch Block Right	1	1	0
Cardiac Disorder	0	0	1
Cardiac Failure Congestive	1	0	0
Myocardial Infarction	4	2	4
Palpitations	0	2	0
Sinus Arrhythmia	1	0	0
Sinus Bradycardia	2	7	8
Supraventricular Extrasystoles	1	1	1
Supraventricular Tachycardia	0	1	0
Tachycardia	1	0	0
Ventricular Extrasystoles	0	2	1
Ventricular Hypertrophy	1	0	0
Wolff-Parkinson-White Syndrome	0	1	0
Congenital, Familial and Genetic Disorders	**0**	**1**	**2**
Ventricular Septal Defect	0	1	2
Ear and Labyrinth Disorders	**1**	**2**	**1**
Cerumen Impaction	0	1	0
Ear Pain	1	0	0
Tinnitus	0	1	0
Vertigo	0	1	1
Eye Disorders	**4**	**2**	**1**
Conjunctival Haemorrhage	0	1	0
Conjunctivitis	2	0	0
Eye Allergy	1	0	0

More discussion on `page_by`, `group_by` and `subline_by` features can be found on the r2rtf package website.

The procedure to generate a baseline characteristics table can be summarized as follows:

- Step 1: Read data (i.e., `adae` and `adsl`) into R.
- Step 2: Count the number of participants by SOC and treatment arm (rows with bold text) and save into `t1`.
- Step 3: Count the number of participants in each AE term by SOC, AE term, and treatment arm (rows without bold text) and save into `t2`.
- Step 4: Bind `t1` and `t2` by row into `t_ae`.
- Step 5: Count the number of participants in each arm as `t_pop`.
- Step 6: Row-wise combine `t_pop` and `t_ae` into `tbl_ae_spec`.
- Step 7: Format the output by using r2rtf.

9. Assemble TLFs

```
library(r2rtf)
```

After TLFs are created and saved into individual files, we need to assemble them into one file in a pre-specified order.

There are two general approaches to achieving the goal.

1. Combine RTF source code in individual files into one large RTF file.
2. Leverage the `Toggle Fields` feature in Microsoft Word to embed RTF files using hyperlinks.

Let's illustrate the idea by using selected TLFs generated from previous chapters. Here, we assume files are saved in the `tlf/` folder.

```
tlf_path <- c(
  "tlf/tbl_disp.rtf", # Disposition table
  "tlf/tlf_eff.rtf", # Efficacy table
  "tlf/tlf_km.rtf" # K-M plot
)
```

Disposition of Participants

	Placebo		Xanomeline Low Dose		Xanomeline High Dose	
	n	(%)	n	(%)	n	(%)
Participants in population	86		84		84	
Completed	58	67.4	25	29.8	27	32.1
Discontinued	28	32.6	59	70.2	57	67.9
Adverse Event	8	9.3	44	52.4	40	47.6
Death	2	2.3	1	1.2	0	0.0
I/E Not Met	1	1.2	0	0.0	2	2.4
Lack of Efficacy	3	3.5	0	0.0	1	1.2
Lost to Follow-up	1	1.2	1	1.2	0	0.0
Physician Decision	1	1.2	0	0.0	2	2.4
Protocol Violation	1	1.2	1	1.2	1	1.2
Sponsor Decision	2	2.3	2	2.4	3	3.6
Withdrew Consent	9	10.5	10	11.9	8	9.5

9.1. Combine RTF Source Code

> **Note**
>
> The code below requires r2rtf version >= 1.0.0.

The `r2rtf::assemble_rtf()` function allows the user to combine RTF source code in individual files into one larger RTF file.

> **Caution**
>
> One limitation of combining RTF source code is that we are not able to specify the page orientation of each TLF in the combined document.

```
r2rtf::assemble_rtf(
  input = tlf_path,
  output = "tlf/rtf-combine.rtf"
)
```

9.2. Using Toggle Fields

Microsoft Word uses toggle fields to embed files into one Word document. The approach is a dynamic link between files by providing the absolute file path.

> **Tip**
>
> There is a slight learning curve on how toggle fields work in Microsoft Word. After you become familiar with the workflow, toggle fields can extend your capability to manage a large number of TLFs in RTF format.

The `assemble_docx()` function allows you to create a `.docx` file with toggle fields as below. One benefit is to control the page direction of each TLF as below.

```
r2rtf::assemble_docx(
  tlf_path,
  output = "tlf/rtf-combine-toggle.docx",
  landscape = c(FALSE, FALSE, TRUE)
)
```

After opening the generated `.docx` file, you will see a blank file because the file only contains fields with hyperlinks.

By using Alt + F9 to display the fields and you will see information similar to the screenshot below.

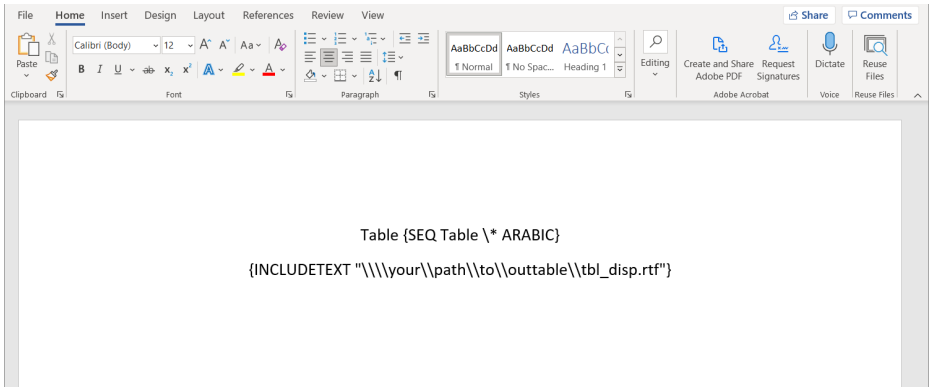


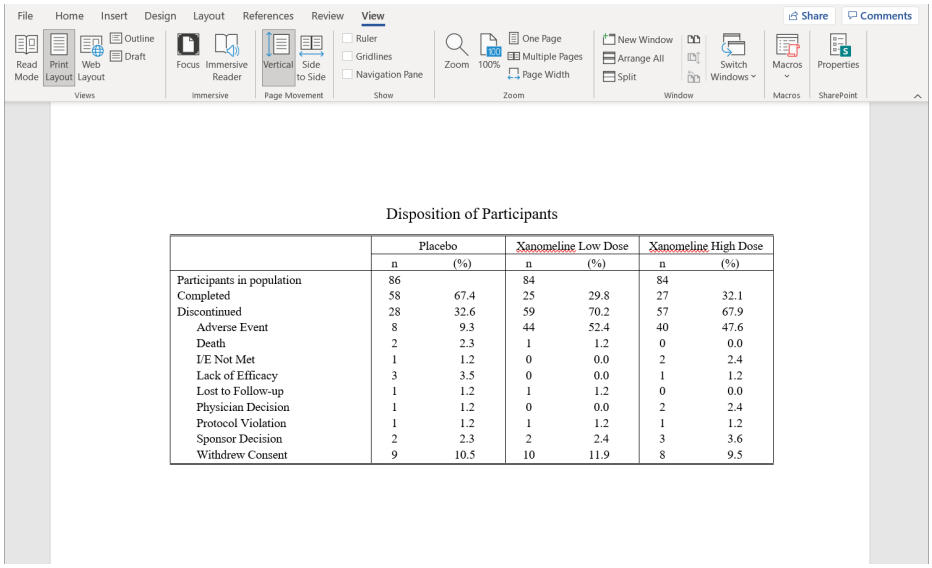
Figure 9.1.: Using Alt + F9 to display fields

> **Tip**
>
> A typical error message is that system can not find the file if you only provide a relative path. Please double-check that the correct

absolute file path is in the `INCLUDETEXT` field.

To test the toggle field, you can right-click an `INCLUDETEXT` filed and click `Update Field`.

If it works, you can see a table similar to the snapshot below by using Alt + F9. It is a shortcut to change between results and field display mode.



Disposition of Participants

	Placebo		Xanomeline Low Dose		Xanomeline High Dose	
	n	(%)	n	(%)	n	(%)
Participants in population	86		84		84	
Completed	58	67.4	25	29.8	27	32.1
Discontinued	28	32.6	59	70.2	57	67.9
Adverse Event	8	9.3	44	52.4	40	47.6
Death	2	2.3	1	1.2	0	0.0
I/E Not Met	1	1.2	0	0.0	2	2.4
Lack of Efficacy	3	3.5	0	0.0	1	1.2
Lost to Follow-up	1	1.2	1	1.2	0	0.0
Physician Decision	1	1.2	0	0.0	2	2.4
Protocol Violation	1	1.2	1	1.2	1	1.2
Sponsor Decision	2	2.3	2	2.4	3	3.6
Withdrew Consent	9	10.5	10	11.9	8	9.5

Figure 9.2.: Update fields

Now you can update all toggle fields to display all TLFs by selecting all fields (Ctrl + A), then press F9. We suggest testing one toggle field before updating all of them.

As the `.docx` file contain dynamic links, you can keep updating the TLFs if you need to refresh content in individual RTF files by selecting all fields (Ctrl + A), then press F9.

> **Tip**
>
> If you modify table content in the combined `.docx` file, you may get a weird table layout if you update all fields within a toggle field. To resolve the issue, please remove all `\* MERGEFORMAT` in the filed mode using Alt + F9 before updating all toggle fields.

After the combined TLF is ready for delivery, you can also unlink toggle fields to save table contents, because the absolute path may only work for some. To unlink toggle fields, you can select all fields (Ctrl + A), then press Ctrl + Shift + F9.

Part II.

Clinical trial project

10. Overview

In a late-stage clinical trial, the number of A&R deliverables can easily be in the hundreds. For an organization, it is common to have multiple ongoing clinical trials in a clinical program.

To deliver the A&R results of a clinical trial project, it is teamwork that typically requires collaborations from both statisticians and programmers. In this part, let's consider how to organize a clinical trial project as an A&R lead.

Chapter 11 will discuss how to organize source code, documents, and deliverables in an A&R clinical project. We recommend using the R package folder structure.

Chapter 12 will discuss a process or system development lifecycle to manage the A&R of a clinical project. We recommend following an agile management approach to define, develop, validate, and deliver work.

11. Project folder

A clearly defined clinical project folder structure can have many benefits to clinical development teams in an organization. Specifically, a well-defined project structure can achieve:

- Consistency: everyone works on the same folder structure.
- Reproducibility: analysis can be executed and reproduced by different team members months/years later.
- Automation: automatically check the integration of a project.
- Compliance: reduce compliance issues.

We will use the `esubdemo` R package to illustrate the project folder structure for the A&R project. You can clone the project using RStudio IDE with

```
git clone https://github.com/elong0527/esubdemo.git
```

For R users, you already benefit from a well-defined and consistent folder structure. That is the R package folder structure. Every R package developer is required to follow the same convention to organize their R functions before the R package can be disseminated through the Comprehensive R Archive Network (CRAN). As a user, you can easily install and use those R packages after downloading from CRAN. There are many good resources to guide developers on R package development, such as, the R Packages book by Hadley Wickham.

We recommend using the R package folder structure to organize analysis scripts for clinical trial development. Using the R package folder structure to streamline data analysis work has also been proposed before (see Marwick, Boettiger, and Mullen (2018), Wu et al. (2021)).

11.1. Consistency

For consistency, a well-defined folder structure with potential templates ensures project teams organize the A&R work consistently across multiple projects. Consistent folder structure also reduces communication costs between study team members and enhances the transparency of projects.

In this book, we refer to an R package as a **project-specific R package** if the purpose of an R package is to organize analysis scripts for a clinical project.

We refer to an R package as a **standard R package** if the purpose of an R package is to share commonly used R functions to be hosted in a code repository such as CRAN.

Below is minimal sufficient folders and files for a project-specific R package based on the R package folder structure.

- `*.Rproj`: RStudio project file used to open RStudio project.
- `DESCRIPTION`: Metadata for a package including authors, license, dependencies, etc.
- `R/`: Project-specific R functions.
- `vignettes/`: Analysis scripts using R Markdown.
- `man/`: Manual of project-specific R functions.

A general discussion of the R package folder structure can be found in Chapter 3 of the R Packages book (Wickham and Bryan 2023).

We demonstrate the idea using the `esubdemo` project.

In the `esubdemo` project, we saved all TLF generation scripts in previous chapters into the `vignettes/` folder.

> **Note**
>
> Under the `vignettes/` folder, there are two folders: `adam/` and `tlf/`. The `adam/` folder contains ADaM datasets. The `tlf/` folder contains output TLFs in RTF format. We put `adam/` and `tlf/` folders within the `vignettes/` folder only for illustration purposes. In an actual A&R report, you may have a different location to save your input and output.

```
vignettes
   data-adam
   tlf
   tlf-01-disposition.Rmd
   tlf-02-population.Rmd
   tlf-03-baseline.Rmd
   tlf-04-efficacy.Rmd
   tlf-05-ae-summary.Rmd
   tlf-06-ae-spec.Rmd
```

While creating those analysis scripts, we also defined a few helper functions (e.g., `fmt_num` and `count_by`). Those functions are saved in the `R/` folder.

```
R/
   count_by.R
   fmt.R
```

For a clinical trial project, it is also important to provide proper documentation for those help functions. We use roxygen2 package to document

functions. For example, the header below defines each variable in `fmt_est`. More details can be found in Chapter 16 of the R Packages book.

```
#' Format point estimator
#'
#' @param .mean mean of an estimator.
#' @param .sd sd of an estimator.
#' @param digits number of digits for `.mean` and `.sd`.
#'
#' @export
fmt_est <- function(.mean, .sd, digits = c(1, 2)) {
  .mean <- fmt_num(.mean, digits[1], width = digits[1] + 4)
  .sd <- fmt_num(.sd, digits[2], width = digits[2] + 3)
  paste0(.mean, " (", .sd, ")")
}
```

The roxygen2 documentation will be converted into standard R documentation format, and saved as `.Rd` files in the `man/` folder. This step is automatically handled by `devtools::document()`.

```
man
   count_by.Rd
   fmt_ci.Rd
   fmt_est.Rd
   fmt_num.Rd
   fmt_pval.Rd
```

The `man/` folder is used to save documentation automatically generated by roxygen2. A typical workflow is to add roxygen2 documentation before each function in the `R/` folder. Then `devtools::document()` is used to generate all the documentation files in the `man/` folder. More details can be found in Chapter 16 of the R Packages book.

11.2. Reproducibility

Reproducibility of analysis is one of the most important aspects of regulatory deliverables. To ensure a successful reproduction, we need a controlled R environment, including the control of the R version and the R package versions. By using the R package folder structure and proper tools (e.g., renv, packrat), we illustrate how to achieve reproducibility for R and R package versions.

> **Tip**
>
> This is the same level of reproducibility in most SAS environments: https://support.sas.com/en/technical-support/services-policies.html#altos

11.2.1. R version

First, we introduce the control of the R version. In the esubdemo project, a reproducible environment is created when you open the `esubdemo.Rproj` from RStudio IDE. When we open the `esubdemo` project, RStudio IDE will execute the R code in `.Rprofile` automatically. So we can use `.Rprofile` to set up a reproducible environment. More details can be found in https://rstats.wtf/r-startup.html. After we open the `esubdemo` project, the code in `.Rprofile` will automatically check the current R version is the same as we defined in `.Rprofile`.

```
# Set project R version
R_version <- "4.1.1"
```

If there is an R version mismatch, an error message is displayed as below.

```
Error: The current R version is not the same as the current project in R4.1.1
```

> **Caution**
>
> `.Rprofile` is only for project-specific R packages. A standard R package should not use `.Rprofile`.

11.2.2. R package version

Next, we introduce the control of the R package version, which is controlled in two layers. Firstly, we define a snapshot date in `.Rprofile`. The snapshot date allows us to freeze the source code repository.

```
# set up snapshot date
snapshot <- "2021-08-06"

# set up repository based on the snapshot date
repos <- paste0("https://packagemanager.posit.co/cran/", snapshot)

# define repo URL for project-specific package installation
options(repos = repos)
```

We can also define the package repository to be a specific snapshot date. For example, we used Posit Public Package Manager to define the snapshot date to be `2021-08-06`. The snapshot date freezes the R package repository.

In other words, all R packages installed in this R project are based on the frozen R version at the snapshot date. Here it is `2021-08-06` by using the Posit Package Manager.

The information below will be displayed after a new R session is opened.

```
Current project R package repository:
    https://packagemanager.posit.co/cran/2021-08-06
```

> **Note**
>
> Posit Public Package Manager hosts daily CRAN snapshots for Mondays to Fridays of the week. Posit Package Manager, when deployed internally within an organization, provides a solution to host both publicly available and internally developed R packages.

Secondly, we use renv to lock R package versions and save them in the `renv.lock` file. renv provides a robust and stable approach to managing R package versions for project-specific R packages. An introduction of renv can be found on its website.

```
source("renv/activate.R")
```

The R code above in the `.Rprofile` initiates the renv running environment. As a user, you can use `renv::init()`, `renv::snapshot()`, and `renv::restore()` to initialize, save and restore R packages used for the current analysis project.

In the analysis project, the renv package will

- create a `renv.lock` file to save the state of package versions.
- create a `renv/` folder to manage R packages for a project.

> **Caution**
>
> The `renv.lock` file and `renv/` folder are only for project-specific R package. A standard R package should not use renv.

In summary, the R package version is controlled in two layers.

- Define a snapshot date in `inst/startup.R`.
- Using renv to lock R versions within a project.

If the project is initiated properly, you should be able to see similar messages to inform how we control R package versions.

```
* Project '~/esubdemo' loaded. [renv 0.14.0]
```

Once R packages have been properly installed, the system will use the R packages located in the search path defined based on the order of `.libPaths()`. The startup message also provided the R package search path.

```
Below R package path are searching in order to find installed R packages in t
    "/home/zhanyilo/github-repo/esubdemo/renv/library/R-4.1/x86_64-pc-linux-g
    "/rtmp/RtmpT3ljoY/renv-system-library"
```

> **Tip**
>
> A cloud-based R environment (e.g., Posit Workbench) can enhance the reproducibility within an organization by using the same operating system, R version, and R package versions for an A&R project. More details can be found at https://environments.rstudio.com/.

> **Note**
>
> A container solution like Docker (Nüst et al. 2020) could further enhance the reproducibility across an organization at the operating system level but beyond the scope of this book.

In conclusion, to achieve reproducibility for a project-specific R package, a clinical project team can work under a controlled R environment in the same R version and R package versions defined by a repository snapshot date.

11.3. Automation

By using the R package folder structure, you will benefit from many outstanding tools to simplify and streamline your workflow.

We have learned a few functions in `devtools` to generate content automatically. Here is a list of tools that can enhance the workflow.

- devtools: make package development easier.
  - A good overview can be found in Chapter 2 of the R Packages book.
  - `devtools::load_all()`: load all functions in `R/` folder and running environment.
  - `devtools::document()`: automatically create documentation using roxygen2.
  - `devtools::check()`: automatically perform compliance check as an R package.
  - `devtools::build_site()`: automatically run analysis scripts in batch and create a pkgdown website.
- usethis: automates repetitive tasks that arise during project setup and development.
- testthat: streamline testing code.
  - A discussion of using the testthat for an A&R project can be found in (Ginnaram et al. (2021)).
- pkgdown: generate static HTML documentation website for an R package
  - It also allows you to run all analysis code in batch.

You may further automatically execute routines by leveraging CI/CD workflow. For example, the `esubdemo` project will rerun all required checks and build a pkgdown website by using Github Actions.

As the consistent folder is defined, it also becomes easier to create specific tools that fit the analysis and reporting purpose. Below are a few potential tools that can be helpful:

- Create project template using RStudio project templates;
- Add additional compliance checks for analysis and reporting;
- Save log files for running in batch.

11.4. Compliance

For a regulatory deliverable, it is important to maintain compliance. With a consistent folder structure, we can define specific criteria for compliance. Some compliance criteria can be implemented within the automatically checking steps.

For an R package, there are already criteria to ensure R package integrity. More details can be found in Chapter 20 of the R Packages book.

12. Project management

12.1. Setting up for success

A clinical data analysis project is not unlike typical data analysis projects or software projects. Therefore, the conventional wisdom and tricks for managing a successful project are also applicable here. At the same time, clinical projects also have unique traits, such as high standards for planning, development, validation, and delivery under strict time constraints.

Although many factors determine if a project can execute efficiently, we believe a few aspects are critical for long-term success, especially when managing clinical data analysis projects at scale.

12.1.1. Work as a team

As a general principle, all the team members involved in a project should take basic training on project management and understand how to work as a development team. Fitzpatrick and Collins-Sussman (2012) provides some valuable tips on this topic. As always, setting a clear goal and following a system development lifecycle (SDLC) is essential.

12.1.2. Design clean code architecture

Having a clean architecture design for your code improves the project's robustness and flexibility for future changes. For example, we should understand how to separate business logic from other layers; know what should be created as reusable components and what should be written as one-off analysis scripts; write low coupling, high cohesion code, and so on. Martin, Grenning, and Brown (2018) offers some helpful insights on this topic.

12.1.3. Set capability boundaries

Knowing what you can do is essential. Create a core capabilities list for your team.

Sometimes, it is also critical to understand **what not to do**. For example, the hidden cost of integrating with external systems or involving other programming languages can be prohibitively high. Remember, a simple, robust solution is almost always preferable to a complex solution that requires high maintenance and constant attention.

12.1.4. Contribute to the community

Every individual is limited in some way. The collective thinking from a community could benefit a project in the long term. When designing reusable components, make a plan to share with internal communities, or even better, with the open-source community.

12.2. The SDLC

For A&R deliverables in clinical project development, a clearly defined process or system development lifecycle (SDLC) is crucial to ensure regulatory compliance.

SDLC for the A&R deliverables can be defined in four stages.

- Planning: a planning stage to define the scope of a project.
- Development: a development stage to implement target deliverables.
- Validation: a validation stage to verify target deliverables.
- Operation: an operation stage to deliver work to stakeholders.

Importantly, we should not consider SDLC as a linear process. For example, if the study team identifies a new requirement in a development or validation stage, the team should return to the planning stage to discuss and align the scope. An agile project management approach is suitable and recommended for an A&R clinical development project. The goal is to embrace an iterative approach that continuously improves target deliverables based on frequent stakeholder feedback.

There are many good tools to implement agile project management strategy, for example:

- GitHub project board
- Jira

12.3. Planning

The planning stage is important in the SDLC lifecycle as the requirements for all A&R deliverables are gathered and documented.

In the planning stage, a project leader should identify all the deliverables, e.g., a list of tables, listings, and figures (TLFs). For each TLFs, the team should prepare the necessary specifications:

- mock-up tables
- validation level (e.g., independent review or double programming)
- etc.

The project leader should also align work assignments with team members. The purpose is to answer the question of "who is doing what?"

```
Warning in ensure_len_latex(background, nrows, off, include_thead, "white",
The number of provided values in background does not equal to the number of
rows.

Warning in ensure_len_latex(background, nrows, off, include_thead, "white",
The number of provided values in background does not equal to the number of
rows.

Warning in ensure_len_latex(background, nrows, off, include_thead, "white",
The number of provided values in background does not equal to the number of
rows.
```

The project lead should also set up a project folder, as discussed in Chapter 11. The project initiation can be simplified by creating an RStudio project template.

To enable reproducibility, the project leader should also review the startup file (i.e. `.Rprofile` discussed in Section 11.2) and define:

- R version
- Repository of R packages with a snapshot date
- Project package library path
- etc.

		Requirement/Specificati	
Program Name	Program Validation Category	Who	Status
count_by	3	Alice	C
fmt_ci	3	Alice	C
fmt_est	3	Alice	C
fmt_num	3	Alice	C
fmt_pval	3	Alice	C
tlf-01-disposition.Rmd	3	Bob	C
tlf-02-population.Rmd	3	Carol	C
tlf-03-baseline.Rmd	3	Dave	C
tlf-04-efficacy.Rmd	3	Alice	C
tlf-05-ae-summary.Rmd	3	Bob	C
tlf-06-ae-spec.Rmd	3	Carol	C

> **Caution**
>
> After project initiation, modifying `.Rprofile` will be a risk for reproducibility and should be handled carefully if necessary.

12.4. Development

After a project is initiated, the study team starts to develop TLFs based on pre-defined mock-up tables assigned to each team member.

The analysis code and relevant description can be saved in R Markdown files in the `vignettes/` folder.

The use of R Markdown allows developers to assemble narrative text, code, and its comments in one place to simplify documentation. It would be helpful to create a template and define a name convention for all TLFs deliverables. For example, we can use the `tlf_` prefix in the filename to indicate that the R Markdown file is for delivering TLFs. Multiple TLFs with similar designs can be included in one R Markdown file.

For example, in the `esubdemo` project, we have six R Markdown files to create TLFs.

If there are any project-specific R functions that need to be developed, the R functions can be placed in the `R/` folder as discussed in Section 11.1.

12.5. Validation

Validation is a crucial stage to ensure the deliverables are accurate and consistent. After the development stage is completed, the project team needs to validate the deliverables, including R Markdown files for TLFs

deliverables and project-specific R functions. The level of validation is determined at the define stage.

In an R package development, the validation or testing is completed under the `test/` folder. The testthat R package can be used to streamline the validation process. More details of the testthat package for R package validation can be found in Chapter 12 of the R package book.

It is recommended to have a name convention to indicate the type of validation. For example, we can use `test-developer-test`, `test-independent-test`, `test-double-programming` to classify the validation type.

It is recommended to follow the same organization for files in testthat folder as `R/` folder and `vignettes/` folder. Every single file in the `R/` folder and `vignettes/` folder should have a testing file saved in the `tests/testthat/` folder to validate the content.

For example, in `esubdemo` project, we can have a list of testing files below.

```
tests/testthat
   test-independent-test-tlf-01-disposition.R
   test-independent-test-tlf-02-population.R
   test-independent-test-tlf-03-baseline.R
   test-independent-test-tlf-04-efficacy.R
   test-independent-test-tlf-05-ae-summary.R
   test-independent-test-tlf-06-ae-spec.R
   test-independent-test-fmt.R
```

To validate the content of a table, we can save the last datasets ready for table generation as a `.Rdata` file. A validator can reproduce the TLF and compare it with the original result saved in the `.Rdata` file. A test is passed when the results match. Customers can directly review the formatting of the TLFs by comparing them with the mock-up.

To validate a figure, we can use the snapshot testing strategy.

After the validator completes the testing of project-specific functions and R Markdown files, the process to execute and report testing results is the same for a standard R package. The `devtools::test()` function automatically executes all testing cases and summarizes the testing results in a report.

After completing the validation, the validator updates the status in a validation tracker. The project lead reviews the tracking sheet to make sure all required activities in the SDLC are completed, and the tracking sheet has been filled correctly. The deliverables are ready for customer review after all the validation steps are completed. Any changes to the output requested by customers are documented.

12.6. Operation

After completion of development and required validation of all A&R deliverables, the project lead runs compliance checks for a project-specific R package similar to other R packages. `devtools::check()` is a convenient way to run compliance checks or `R CMD check`. `R CMD check` is an automated check of the contents in the R package for frequently encountered issues before submission to CRAN. Since the project-specific R package is not submitted to CRAN, some checks can be customized and skipped in `devtools::check()`. The project lead should work with the study team to ensure all reported errors, warnings, and notes by `devtools::check()` are fixed.

The project lead can also use the R package pkgdown to build a complete website for a project-specific R package. The pkgdown website is a convenient way to run all analyses in batch and integrate outputs in a website, which comprehensively covers project-specific R functions, TLF generation programs, outputs and validation tracking information, etc. For

example, in the `esubdemo` project, we created the pkgdown website at https://elong0527.github.io/esubdemo/.

Many of the tasks in SDLC can be completed automatically. An organization can leverage CI/CD workflow to automatically enable those tasks, such as running testing cases and creating a pkgdown website. For example, in the `esubdemo` project, we set up GitHub Actions for it. This can be done by using `usethis::use_github_action()`.

Part III.

eCTD submission

13. Overview

The electronic Common Technical Document (eCTD) is a standard format for the electronic submission of applications, amendments, supplements, and reports from the applicant to the regulator. The eCTD offers a solution to submit documents stored in a standard directory structure, with file integrity validation mechanisms in place.

To submit TLFs created by R to regulatory agencies, we should follow the spirit of the existing eCTD submission guidelines to prepare the deliverables, and provide the essential details in the relevant documents for review.

The goal of the following two chapters is to provide guidance to follow Section 4.1.2.10 of the FDA Study Data Technical Conformance Guide:

> Sponsors should provide the software programs used to create all ADaM datasets and **generate tables and figures associated with primary and secondary efficacy analyses**. Furthermore, sponsors should submit software programs used to generate additional information included in Section 14 CLINICAL STUDIES of the Prescribing Information (PI)26 if applicable. **The specific software utilized should be specified in the ADRG**. The main purpose of requesting the submission of these programs **is to understand the process by which the variables for the respective analyses were created and to confirm the analysis algorithms**. Sponsors should submit software programs in **ASCII text format**; however, executable file extensions should not be used.

Chapter 14 will focus on preparing proprietary R packages and analysis code into proper formats for submission.

Chapter 15 will discuss the recommendations to make the R code running environment reproducible for dry run tests and reviews.

14. Submission package

In this chapter, we will first give a high-level overview of what assets in the eCTD submission package we should focus on when submitting R code. Then, we will discuss how to prepare the proprietary R packages (if any), and make them be part of the submission package. In the end, we will provide reusable templates for updating Analysis Data Reviewer's Guide (ADRG) and Analysis Results Metadata (ARM) so that the reviewers receive proper instructions to reproduce the analysis results.

14.1. Prerequisites

This chapter uses pkglite (Zhao et al. 2023) to convert R source packages into text files and back.

```
install.packages("pkglite")
```

The demo project (R package) we will prepare for submission is called `esubdemo`, which is available on GitHub. You can download or clone it:

```
git clone https://github.com/elong0527/esubdemo.git
```

The demo submission package (not to be confused with the R package above) is `ectddemo`, which is also available on GitHub. You can download or clone it:

```
git clone https://github.com/elong0527/ectddemo.git
```

We assume the paths to the two folders are `esubdemo/` and `ectddemo/` below.

14.2. The whole game

In eCTD deliverable, the analysis datasets and source code are saved under the eCTD module 5 (clinical study reports) folder

```
ectddemo/m5/datasets/<study>/analysis/adam/
```

The files in two directories within the `adam/` folder are critical for documenting analysis using R: `datasets/` and `programs/`.

```
ectddemo/m5/datasets/ectddemo/analysis/adam/
  datasets
      adae.xpt
      ...
      adrg.pdf
      analysis-results-metadata.pdf
      define.xml
      define2-0-0.xsl
  programs
      r0pkgs.txt
      tlf-01-disposition.txt
      tlf-02-population.txt
      tlf-03-baseline.txt
      tlf-04-efficacy.txt
      tlf-05-ae-summary.txt
      tlf-06-ae-spec.txt
```

The special considerations for each component are listed below.

14.2.1. `datasets`

Folder path: `ectddemo/m5/datasets/ectddemo/analysis/adam/datasets/`.

- ADaM data in `.xpt` format: created by SAS or R.
- `define.xml`: created by Pinnacle 21.
- ADRG (Analysis Data Reviewer's Guide)
  - "Macro Programs" section: provide R and R package versions with a snapshot date.
  - Appendix: provide step-by-step instructions for reviewers to reproduce the running environment and rerun analyses.
- ARM (Analysis Results Metadata): provide the links between TLFs and analysis programs in tables.

14.2.2. `programs`

Folder path: `ectddemo/m5/datasets/ectddemo/analysis/adam/programs/`.

- `r0pkgs.txt`: contains all internally developed proprietary R packages.
- Other `.txt` files: each contains R code for a specific analysis.

14.2.3. Notes

To verify if the submission package works, rerun all analyses following the instructions defined in ADRG.

A few things need to be paid attention to in order to pass compliance checks:

- The file names under `programs/` should be in lower case letters (with no underscores or other special characters).
- The `.txt` files should only contain ASCII characters. This can be verified by `pkglite::verify_ascii()`
- All `.docx` files should be converted to PDF files for formal submission.

Now you have a general idea about the relevant components of the submission package. We will prepare the proprietary R packages in the following sections.

14.3. Practical considerations for R package submissions

Before we start, there are a few aspects to figure out in order to accurately identify the R packages for submission.

14.3.1. Source location

There are a few common places to host R (source) packages:

1. CRAN
2. Public Git repository
3. Private Git repository (accessible externally)
4. Private Git repository (inaccessible externally)

For R packages hosted on CRAN or a public Git repository, you probably do not need to submit them as part of the submission package, as the reviewers can install them directly by following the instructions in ADRG.

For R packages hosted in private repositories, to avoid any complications in infrastructure, authentication, and communication, it is often recommended to submit them as part of the submission package.

14.3.2. Dependency locations

R package dependency is another major factor to consider before preparing your proprietary R package for submission.

For dependencies available from CRAN or public Git repositories, you can declare them directly using the regular `Imports` and `Suggests` syntax or the remotes dependency syntax in the `DESCRIPTION` file.

For dependencies hosted in private Git repositories, you should pack them with the primary R package(s) you want to submit, as pkglite supports packing multiple R packages into a single text file; then restore and install them in the order they are packed.

14.3.3. R version

Always use a consistent version of R for developing the TLFs and for submission. For example, you could enforce a rule to only use R `x.y.z` where `z = 1`, such as R `4.0.1` or R `4.1.1`. This can be automatically checked using a startup script when the R project is opened.

14.3.4. Package repo version

Always use the same snapshot package repo for developing the TLFs and for submission. Again, this can be checked in the project startup script, as discussed in Section 11.2.

14.3.5. System environments

Introducing any extra external dependencies will likely increase the cost of qualification, validation, testing, and maintenance, especially under Windows. Therefore, it is recommended to keep the dependency chain simple, especially when involving compiled code (e.g., C, C++, Fortran).

14.4. Prepare R packages for submission

To prepare R packages for submission, one needs to pack the packages into text files, and then verify if the files only contain ASCII characters. With packed packages, one can unpack and install them from the text files, too.

14.4.1. Pack

Let's pack the `esubdemo` package into a text file. Assume the source package path is `esubdemo/`. You should be able to pack the package with a single pipe:

```
library("pkglite")

"esubdemo/" %>%
  collate(file_ectd(), file_auto("inst")) %>%
  pack(output = "r0pkgs.txt")
```

```
-- Packing into pkglite file ---------------------------------------------------
-- Reading package: esubdemo ---------------------------------------------------
Reading ".Rbuildignore"
Reading "DESCRIPTION"
Reading "NAMESPACE"
Reading "README.md"
Reading "R/count_by.R"
Reading "R/fmt.R"
Reading "R/utils-pipe.R"
Reading "R/zzz.R"
Reading "man/count_by.Rd"
Reading "man/fmt_ci.Rd"
Reading "man/fmt_est.Rd"
Reading "man/fmt_num.Rd"
Reading "man/fmt_pval.Rd"
Reading "man/pipe.Rd"
Reading "inst/pkgdown/assets/readme.txt"
Reading "inst/pkgdown/templates/readme.txt"
Reading "inst/startup.R"
Writing to: "r0pkgs.txt"
```

Figure 14.1.: Output of pkglite::pack()

Let's open the generated text file:

```
file.edit("r0pkgs.txt")
```

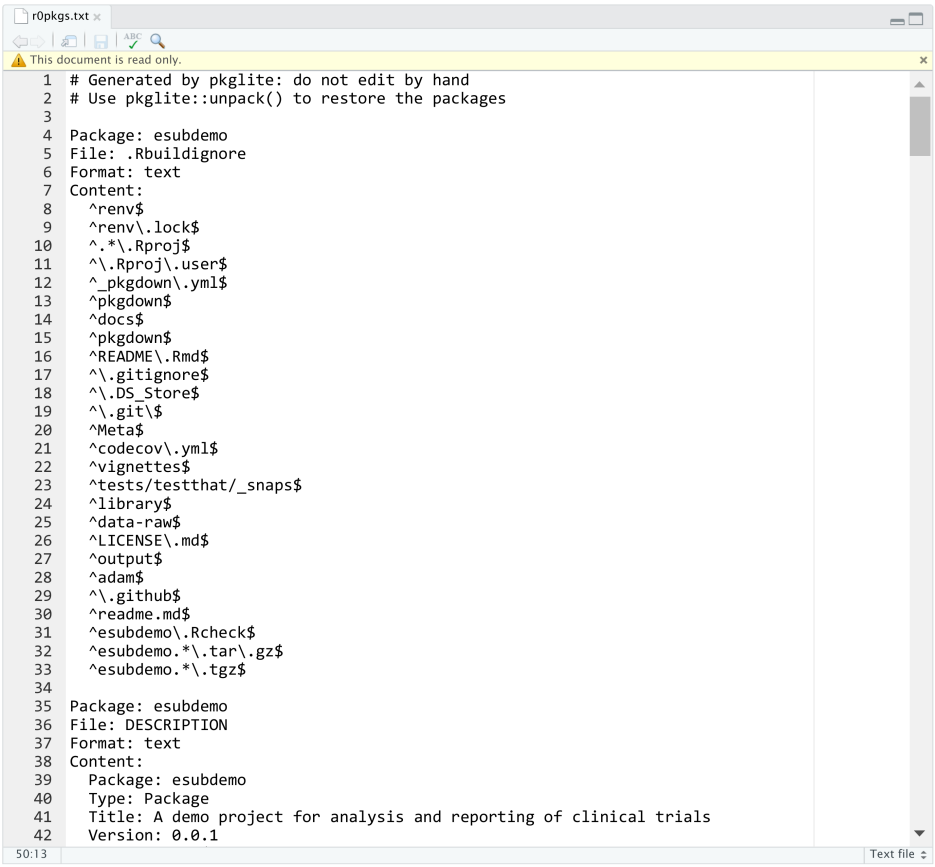


Figure 14.2.: Preview the generated text file

What happened in the pipe? The function `pkglite::collate()` evaluates a specified scope of folders and files defined by a list of **file specifications**, and generates a **file collection** object. This file collection contains the metadata required to properly convert the files into text which is then used by `pkglite::pack()`. With this flow, you can define the scope of the R source package to be packed for submission in a flexible yet principled way.

To pack multiple R packages, simply feed multiple file collections as inputs:

```
pack(
  "/path/to/pkg1/" %>% collate(file_ectd()),
  "/path/to/pkg2/" %>% collate(file_ectd()),
  output = "r0pkgs.txt"
)
```

The R packages are always packed in the specified order and are always unpacked and installed in the same order. Therefore, make sure to pack the low-level dependencies first.

For more details on how to customize file specifications and operate on file collections, check out the vignette generate file specifications and curate file collections.

14.4.2. Verify

You should always verify if the text file only contains ASCII characters:

```
verify_ascii("r0pkgs.txt")
```

This should give `TRUE` if the file only contains ASCII characters, or `FALSE` with the affected lines otherwise.

14.4.3. Unpack

One can unpack and install the package from the text file, too. For example:

```
unpack("r0pkgs.txt", output = "/tmp/", install = TRUE)
```

If the test is successful, this command can be used in the ADRG instructions for restoring and installing the packed R package(s).

You can then proceed to move the file `r0pkgs.txt` to the folder `ectddemo/m5/datasets/ectddemo/analysis/adam/programs/`, or specify the output text file path above directly.

14.5. Prepare analysis programs for submission

Besides the R packages, we need to convert the R Markdown (`.Rmd`) files into `.txt` files and saved them in the `programs/` folder. You can do this with `knitr::purl()`:

```
input_path <- "esubdemo/vignettes/"
output_path <- "ectddemo/m5/datasets/ectddemo/analysis/adam/programs/"

convert_rmd <- function(filename, input_dir, output_dir) {
  knitr::purl(
    file.path(input_dir, paste0(filename, ".Rmd")),
    output = file.path(output_dir, paste0(filename, ".txt"))
  )
}

"tlf-01-disposition" %>% convert_rmd(input_path, output_path)
"tlf-02-population" %>% convert_rmd(input_path, output_path)
"tlf-03-baseline" %>% convert_rmd(input_path, output_path)
"tlf-04-efficacy" %>% convert_rmd(input_path, output_path)
"tlf-05-ae-summary" %>% convert_rmd(input_path, output_path)
"tlf-06-ae-spec" %>% convert_rmd(input_path, output_path)
```

Optionally, you can add a header to the individual `.txt` files to explain the context and help the reviewers rerun the code. For example:

```
# Note to Reviewer
# To rerun the code below, please refer to the ADRG appendix.
# After the required packages are installed,
# the path variable needs to be defined by using the example code below.
#
# path = list(adam = "/path/to/esub/analysis/adam/datasets") # Modify to use actual location
# path$outtable = path$outgraph = "." # Outputs saved to the current folder
```

To automate this process:

```
header <- readLines(textConnection("# Note to Reviewer
# To rerun the code below, please refer to the ADRG appendix.
# After the required packages are installed,
# the path variable needs to be defined by using the example code below.
#
# path = list(adam = \"/path/to/esub/analysis/adam/datasets\") # Modify to use actual locati
# path$outtable = path$outgraph = \".\" # Outputs saved to the current folder"))

append_header <- function(filename, output_dir, header) {
  file <- file.path(output_dir, paste0(filename, ".txt"))
  x <- readLines(file)
  y <- c(header, "", x)
  writeLines(y, con = file)
  invisible(file)
}

"tlf-01-disposition" %>% append_header(output_path, header)
"tlf-02-population" %>% append_header(output_path, header)
"tlf-03-baseline" %>% append_header(output_path, header)
"tlf-04-efficacy" %>% append_header(output_path, header)
```

```
"tlf-05-ae-summary" %>% append_header(output_path, header)
"tlf-06-ae-spec" %>% append_header(output_path, header)
```

14.6. Update ADRG

After we converted the R packages and R Markdown files into the appropriate formats and verified that they can be restored and executed correctly, we need update the ADRG to provide guidelines on how to use them.

Specifically, we need to update two sections in ADRG.

The first section is "Macro Programs", where R and R package versions with a snapshot date are provided. For example:

```
7.x Macro Programs

Submitted R programs have [specific patterns] in filenames.
All internally developed R functions are saved in the r0pkgs.txt file.
The recommended steps to unpack these R functions for analysis output
programs are described in the Appendix.

The tables below contain the software version and instructions
for executing the R analysis output programs:
```

Program Name	Output Table	Title
tlf-01-disposition.txt	Table x.y.z	Disposition of Patients
tlf-02-population.txt	Table x.y.z	Participants Accounting in Analysis Population (A
tlf-03-baseline.txt	Table x.y.z	Participant Baseline Characteristics (All Participa
tlf-04-efficacy.txt	Table x.y.z	ANCOVA of Change from Baseline Glucose (mmo
tlf-05-ae-summary.txt	Table x.y.z	Analysis of Adverse Event Summary (Safety Anal
tlf-06-ae-spec.txt	Table x.y.z	Analysis of Participants With Specific Adverse Ev

Open-Source R Analysis Package	Package Version	Analysis Package Description
pkglite	0.2.0	Prepare submission package
haven	2.4.3	Read SAS datasets
dplyr	1.0.7	Manipulate datasets
tidyr	1.1.3	Manipulate datasets
emmeans	1.6.2-1	Least-squares means estimation
r2rtf	0.3.0	Create RTF tables

Proprietary R Analysis Package	Package Version	Analysis Package Description
esubdemo	0.1.0	A demo package for analysis and reporting of clinical t

The second section (Appendix) should include step-by-step instructions to reproduce the running environment and rerun analyses. For example:

```
Appendix: Instructions to Execute Analysis Program in R

1. Install R

Download and install R 4.1.1 for Windows from
https://cran.r-project.org/bin/windows/base/old/4.1.1/R-4.1.1-win.exe

2. Define working directory

Create a temporary working directory, for example, "C:\tempwork".
Copy all submitted R programs into the temporary folder.
All steps below should be executed in this working directory
represented as "." in the example R code below.

3. Specify R package repository

The R packages are based on CRAN at 2021-08-06. To install the exact
R package versions used in this project, run the code below to set
the snapshot repository.
```

```
options(repos = "https://packagemanager.posit.co/cran/2021-08-06")

4. Install open-source R packages

In the same R session, install the required packages by running the code belo

install.packages(c("pkglite", "publicpkg1", "publicpkg2"))

5. Install proprietary R packages

All internal R packages are packed in the file r0pkgs.txt. In the same R sess
restore the package structures and install them by running the code below.
Adjust the output path as needed to use a writable local directory.

pkglite::unpack("r0pkgs.txt", output = ".", install = TRUE)

6. Update path to dataset and TLFs

INPUT path: to rerun the analysis programs, define the path variable

- Path for ADaM data: path$adam

OUTPUT path: to save the analysis results, define the path variable

- Path for output TLFs: path$output

All these paths need to be defined before executing the analysis program. For

path = list(adam = "/path/to/esub/analysis/adam/datasets/") # Modify to use a
path$outtable = path$outgraph = "." # Outputs saved to the current folder

7. Execute analysis program
```

```
To reproduce the analysis results, rerun the following programs:

- tlf-01-disposition.txt
- tlf-02-population.txt
- tlf-03-baseline.txt
- tlf-04-efficacy.txt
- tlf-05-ae-summary.txt
- tlf-06-ae-spec.txt
```

An example ADRG following this template can be found in ectddemo.

14.7. Update ARM

The ARM (Analysis Results Metadata) should provide specific information related to R in two sections:

- Section 2: indicate the **Programming Language**;
- Section 3: document the details of the R programs listed in section 3.

For example, in ARM section 2, "Analysis Results Metadata Summary":

```
if (knitr::is_latex_output()) {
  df4 %>% kbl(format = "latex")
}
```

Table Reference	Table Title	Programming Language	Program Name (
[Ref. x.y.z: P001ZZZ9999: Table 1-1]	Disposition of Patients	R	tlf-01-disposition
...	...	...	...

```
if (knitr::is_html_output()) {
  df4 %>%
    kbl(format = "html") %>%
    kable_classic(full_width = FALSE, html_font = "'Times New Roman', Times,
    column_spec(1, extra_css = "border: 1px solid #000; text-align: center;")
    column_spec(2, extra_css = "border: 1px solid #000; text-align: center;")
    column_spec(3, extra_css = "border: 1px solid #000; text-align: center;")
    column_spec(4, extra_css = "border: 1px solid #000; text-align: center;")
    column_spec(5, extra_css = "border: 1px solid #000; text-align: center;")
    row_spec(0, background = "#DFDFDF", bold = TRUE, extra_css = "border: 1px
}
```

In ARM section 3, “Analysis Results Metadata Details”:

x1	x2
Table Reference: [Ref. x.y.z: P001ZZZ9999: Table 1-1]	...
Analysis Result	...
Analysis Parameters (s)	...
Analysis Reason	...
Analysis Purpose	...
...	...
Programming Statements	(R version 4.1.1), [P001ZZZ9999:

15. Running environment

In the previous chapter, we generated instructions to manually create the running environments for reproducing the A&R deliverables.

In this chapter, we focus on automating the creation of the R environments with R code to accelerate the dry run testing process, simplifying the ADRG instructions, and making it easy to recreate different environment settings with reproducible analysis results.

15.1. Prerequisites

cleanslate is an R package that offers a solution to create portable R environments.

> **Note**
>
> As of Q4 2021, the cleanslate package used in this chapter is still under active development and validation. This chapter gives a preview of the planned APIs. They may change in the future.

Install cleanslate from CRAN (once available):

```
install.packages("cleanslate")
```

Or from GitHub (once available):

```
remotes::install_github("Merck/cleanslate")
```

15.2. Practical considerations

The cleanslate package supports:

- Creating a project folder with project-specific context (`.Rproj`, `.Rprofile`, `.Renviron`)
- Installing a specific version of R into the project folder
- Installing a specific version of Rtools into the project folder

An essential feature of cleanslate is that it does **not** require administrator privileges to run R and Rtools installers. This makes it easier to deploy under enterprise settings and avoids security and portability concerns.

As many of the A&R deliverables are currently created, validated, and delivered under Windows, the primary focus is Windows at the moment, while the support for other platforms might be added in future versions.

15.3. Create canonical environments

One can create a running environment with "canonical" settings with a single function call to `use_cleanslate()`:

```
cleanslate::use_cleanslate(
  "C:/temp/",
  r_version = "4.1.1",
  from = "https://cran.r-project.org/",
  repo = "https://packagemanager.posit.co/cran/2021-08-06"
)
```

This will

- Create a project folder under `C:/temp/` with a `.Rproj` file;
- Download R `4.1.1` installer from CRAN, and install it into `C:/temp/R/`;
- Not install Rtools (by default, `rtools_version = NULL`);
- Create a `.Rprofile` file under the project folder, set `options(repos)` to use the specified `repo` (a Posit Public Package Manager snapshot in this example), and give instruction to set the R binary path in RStudio IDE;
- Create a `.Renviron` file under the project folder and set the library path to be the library of the project-specific R installation.

As a principle, one should always double-click the `.Rproj` file to open the project. This will ensure some sanity checks in the `.Rprofile`, such as whether the R and library are located within the project folder.

15.4. Create tailored environments

To create a more customized running environment, one can use the specific functions to tailor each aspect, for example:

```
library("cleanslate")

"C:/temp/" %>%
  use_project() %>%
  use_rprofile() %>%
  use_renviron() %>%
  use_r_version(version = "4.1.1") %>%
  use_rtools(version = "rtools40")
```

The project context functions (`use_project()`, `use_rprofile()`, `use_renviron`) support custom templates using brew.

The `use_r_*()` functions have variations that serve as shortcuts to use R versions defined by release lifecycles, for example, `use_r_release()`, `use_r_oldrel()`, and `use_r_devel()`. Note that to ensure better reproducibility, one should still use `use_r_version()` as the release, oldrel, and devel versions will shift as time goes by.

The helper functions `version_*()` and `snapshot_*()` can assist you in determining specific versions of R and Rtools that are currently available, besides generating and verifying the snapshot repo links.

15.5. Update ADRG

If you use cleanslate, remember to update the ADRG instructions for executing the analysis programs in R. Mostly, this can simplify the first three steps on creating a project, installing a specific version of R, and configuring the package repo location. For example:

```
Appendix: Instructions to Execute Analysis Program in R

1. Setup R environment

Open the existing R, install the required packages by running the code below

install.packages("cleanslate")

Create a temporary working directory, for example, "C:\tempwork".
Copy all submitted R programs into the temporary folder.
In the same R session, run the code below to create a project
with a portable R environment.

cleanslate::use_cleanslate(
  "C:/temp/",
  r_version = "4.1.1",
```

```
  from = "https://cran.r-project.org/",
  repo = "https://packagemanager.posit.co/cran/2021-08-06"
)

2. Open the project

Go to the working directory created above, double click the .Rproj file
to open the project in RStudio IDE. Follow the instructions to select the
project-specific R version, then restart RStudio IDE. If successful,
the R version and package repo should be printed as defined above.

3. Install open-source R packages

In the new R session, install the required packages by running the code below.

install.packages(c("pkglite", "publicpkg1", "publicpkg2"))

4. Install proprietary R packages

All internal R packages are packed in the file r0pkgs.txt. In the same R session,
restore the package structures and install them by running the code below.
Adjust the output path as needed to use a writable local directory.

pkglite::unpack("r0pkgs.txt", output = ".", install = TRUE)

5. Update path to dataset and TLFs

INPUT path: to rerun the analysis programs, define the path variable

- Path for ADaM data: path$adam

OUTPUT path: to save the analysis results, define the path variable
```

```
- Path for output TLFs: path$output

All these paths need to be defined before executing the analysis program. For

path = list(adam = "/path/to/esub/analysis/adam/datasets/") # Modify to use a
path$outtable = path$outgraph = "." # Outputs saved to the current folder

6. Execute analysis program

To reproduce the analysis results, rerun the following programs:

- tlf-01-disposition.txt
- tlf-02-population.txt
- tlf-03-baseline.txt
- tlf-04-efficacy.txt
- tlf-05-ae-summary.txt
- tlf-06-ae-spec.txt
```

15.6. RStudio addin

To make it convenient to use cleanslate in experiments, one can also use its RStudio IDE addin. After cleanslate is installed, click `Addins -> cleanslate -> Create portable R environment` in RStudio IDE, or call `cleanslate:::create_env_addin()` to open it.

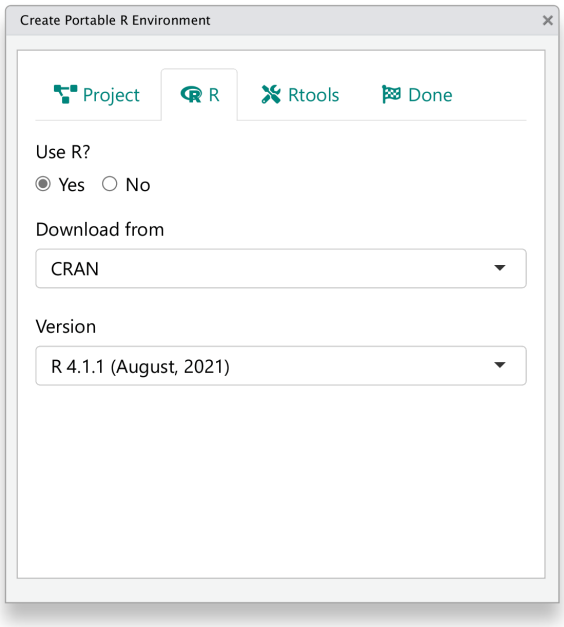


Figure 15.1.: cleanslate RStudio addin

The addin provides a wizard-like interface to help create the environment with the most important options, yet with less flexibility compared to the functional API demonstrated above.

References

Fitzpatrick, Brian, and Ben Collins-Sussman. 2012. *Team Geek: A Software Developer's Guide to Working Well with Others*. O'Reilly Media.

Ginnaram, Madhusudhan, Simiao Ye, Yalin Zhu, and Yilong Zhang. 2021. "A Process to Validate Internal Developed R Package Under Regulatory Environment." PharmaSUG.

Martin, Robert C, James Grenning, and Simon Brown. 2018. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.

Marwick, Ben, Carl Boettiger, and Lincoln Mullen. 2018. "Packaging Data Analytical Work Reproducibly Using R (and Friends)." *The American Statistician* 72 (1): 80–88.

Nüst, Daniel, Dirk Eddelbuettel, Dom Bennett, Robrecht Cannoodt, Dav Clark, Gergely Daróczi, Mark Edmondson, et al. 2020. "The Rocker-verse: Packages and Applications for Containerisation with R." *The R Journal* 12 (1): 437–61.

Wickham, Hadley, and Jennifer Bryan. 2023. *R Packages*. O'Reilly Media, Inc.

Wu, Peikun, Uday Preetham Palukuru, Yiwen Luo, Sarad Nepal, and Yilong Zhang. 2021. "Analysis and Reporting in Regulated Clinical Trial Environment Using R." PharmaSUG.

Zhao, Yujie, Nan Xiao, Keaven Anderson, and Yilong Zhang. 2023. "Electronic Common Technical Document Submission with Analysis Using R." *Clinical Trials* 20 (1): 89–92.