

Python for Clinical Study Reports and Submission

Yilong Zhang

Nan Xiao

Table of contents

Preface	4
1 Introduction	5
I Tables, Listings, and Figures	6
2 Disposition of Participants Table	7
2.1 Overview	7
2.2 Step 1: Load Data	7
2.3 Step 2: Count Total Participants	8
2.4 Step 3: Count Completed Participants	9
2.5 Step 4: Count Discontinued Participants	10
2.6 Step 5: Break Down Discontinuation Reasons	11
2.7 Step 6: Combine All Results	12
2.8 Step 7: Generate Publication-Ready Output	13
3 Study Population Table	14
3.1 Overview	14
3.2 Step 1: Load Data	14
3.3 Step 2: Calculate Treatment Group Totals	15
3.4 Step 3: Define Helper Function	15
3.5 Step 4: Count Each Population	16
3.5.1 All Randomized Participants	16
3.5.2 Intent-to-Treat Population	16
3.5.3 Efficacy Population	17
3.5.4 Safety Population	17
3.6 Step 5: Combine All Populations	18
3.7 Step 6: Calculate Percentages	18
3.8 Step 7: Format Display Values	19
3.9 Step 8: Create Final Table	20
3.10 Step 9: Generate Publication-Ready Output	21
4 Baseline Characteristics Table	22
4.1 Overview	22
4.2 Step 1: Load Data	22

4.3	Step 2: Calculate Summary Statistics	23
4.3.1	Continuous Variables (Age)	23
4.3.2	Categorical Variables (Sex, Race)	24
4.4	Step 3: Format Results	25
4.4.1	Format Age Statistics	25
4.4.2	Format Categorical Statistics	25
4.5	Step 4: Create Table Structure	26
4.6	Step 5: Generate Publication-Ready Output	28
5	Adverse Events Summary Table	30
5.1	Overview	30
5.2	Imports	30
5.3	Data Preparation	30
5.4	AE Summary Calculations	31
5.5	Format AE Summary Table	32
5.6	Create RTF Output	33
6	Specific Adverse Events Table	35
6.1	Setup	35
6.2	Prepare AE Summary Data	35
6.3	Create RTF Output	36
7	ANCOVA Efficacy Analysis	38
7.1	Setup	38
7.2	Prepare Analysis Data	38
7.3	Create Tables for RTF Output	40
7.4	Create RTF Document	41
8	Summary	43
	References	44

Preface

This is a Quarto book.

1 Introduction

This is a book created from Markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
import polars as pl
from datetime import datetime

df = pl.DataFrame(
    {
        "integer": [1, 2, 3],
        "date": [
            datetime(2025, 1, 1),
            datetime(2025, 1, 2),
            datetime(2025, 1, 3),
        ],
        "float": [4.0, 5.0, 6.0],
        "string": ["a", "b", "c"],
    }
)

print(df)
```

shape: (3, 4)

integer	date	float	string
---	---	---	---
i64	datetime[s]	f64	str
1	2025-01-01 00:00:00	4.0	a
2	2025-01-02 00:00:00	5.0	b
3	2025-01-03 00:00:00	6.0	c

Part I

Tables, Listings, and Figures

2 Disposition of Participants Table

2.1 Overview

Clinical trials need to track how participants flow through a study from enrollment to completion. Following [ICH E3 guidance](#), regulatory submissions require a disposition table in Section 10.1 that summarizes:

- **Enrolled:** Total participants who entered the study
- **Completed:** Participants who finished the study protocol
- **Discontinued:** Participants who left early and their reasons

This tutorial shows you how to create a regulatory-compliant disposition table using Python's `rtflite` package.

```
import polars as pl # Manipulate data
import rtflite as rtf # Reporting in RTF format
```

2.2 Step 1: Load Data

We start by loading the Subject-level Analysis Dataset (ADSL), which contains all participant information needed for our disposition table.

The ADSL dataset stores participant-level information including treatment assignments and study completion status. We're using the `parquet` format for data storage.

```
adsl = pl.read_parquet("data/adsl.parquet")
```

Let's examine the key variables we'll use to build our disposition table:

- **USUBJID:** Unique identifier for each participant
- **TRT01P:** Treatment name (text)
- **TRT01PN:** Treatment group (numeric code)
- **DISCONFL:** Flag indicating if participant discontinued (Y/N)
- **DCREASCD:** Specific reason for discontinuation

```
adsl.select(["USUBJID", "TRT01P", "TRT01PN", "DISCONFL", "DCREASCD"])
```

USUBJID	TRT01P	TRT01PN	DISCONFL	DCREASCD
str	str	i64	str	str
"01-701-1015"	"Placebo"	0	" "	"Completed"
"01-701-1023"	"Placebo"	0	"Y"	"Adverse Event"
"01-701-1028"	"Xanomeline High Dose"	81	" "	"Completed"
"01-701-1033"	"Xanomeline Low Dose"	54	"Y"	"Sponsor Decision"
"01-701-1034"	"Xanomeline High Dose"	81	" "	"Completed"
...
"01-718-1254"	"Xanomeline Low Dose"	54	" "	"Completed"
"01-718-1328"	"Xanomeline High Dose"	81	"Y"	"Withdrew Consent"
"01-718-1355"	"Placebo"	0	" "	"Completed"
"01-718-1371"	"Xanomeline High Dose"	81	"Y"	"Adverse Event"
"01-718-1427"	"Xanomeline High Dose"	81	"Y"	"Lack of Efficacy"

2.3 Step 2: Count Total Participants

First, we count how many participants were enrolled in each treatment group.

We group participants by treatment arm and count them using `.group_by()` and `.agg()`. The `.pivot()` operation reshapes our data from long format (rows for each treatment) to wide format (columns for each treatment), which matches the standard disposition table layout.

```
n_rand = (
    adsl
    .group_by("TRT01PN")
    .agg(n = pl.len())
    .with_columns([
        pl.lit("Participants in population").alias("row"),
        pl.lit(None, dtype=pl.Float64).alias("pct") # Placeholder for percentage (not applicable)
    ])
    .pivot(
        index="row",
        on="TRT01PN",
        values=["n", "pct"],
        sort_columns=True
    )
)
```


n_rand

row	n_0	n_54	n_81	pct_0	pct_54	pct_81
str	u32	u32	u32	f64	f64	f64
"Participants in population"	86	84	84	null	null	null

2.4 Step 3: Count Completed Participants

Next, we identify participants who successfully completed the study and calculate what percentage they represent of each treatment group.

We filter for participants where DCREASCD == "Completed", then calculate both counts and percentages. The .join() operation brings in the total count for each treatment group so we can compute percentages.

```
n_complete = (
    adsl
    .filter(pl.col("DCREASCD") == "Completed")
    .group_by("TRT01PN")
    .agg(n = pl.len())
    .join(
        adsl.group_by("TRT01PN").agg(total = pl.len()),
        on="TRT01PN"
    )
    .with_columns([
        pl.lit("Completed").alias("row"),
        (100.0 * pl.col("n") / pl.col("total")).round(1).alias("pct")
    ])
    .pivot(
        index="row",
        on="TRT01PN",
        values=["n", "pct"],
        sort_columns=True
    )
)

n_complete
```

row	n_0	n_54	n_81	pct_0	pct_54	pct_81
str	u32	u32	u32	f64	f64	f64
"Completed"	58	25	27	67.4	29.8	32.1

2.5 Step 4: Count Discontinued Participants

Now we count participants who left the study early, regardless of their specific reason.

We filter for participants where the discontinuation flag `DISCONFL == "Y"`, then follow the same pattern of counting and calculating percentages within each treatment group.

```
n_disc = (
    adsl
    .filter(pl.col("DISCONFL") == "Y")
    .group_by("TRT01PN")
    .agg(n = pl.len())
    .join(
        adsl.group_by("TRT01PN").agg(total = pl.len()),
        on="TRT01PN"
    )
    .with_columns([
        pl.lit("Discontinued").alias("row"),
        (100.0 * pl.col("n") / pl.col("total")).round(1).alias("pct")
    ])
    .pivot(
        index="row",
        on="TRT01PN",
        values=["n", "pct"],
        sort_columns=True
    )
)

n_disc
```

row	n_0	n_54	n_81	pct_0	pct_54	pct_81
str	u32	u32	u32	f64	f64	f64
"Discontinued"	28	59	57	32.6	70.2	67.9

2.6 Step 5: Break Down Discontinuation Reasons

For regulatory reporting, we need to show the specific reasons why participants discontinued.

We filter out completed participants, then group by both treatment and discontinuation reason. The indentation (four spaces) in the row labels helps show these are subcategories under “Discontinued”. We also use `.fill_null(0)` to handle cases where certain discontinuation reasons don’t occur in all treatment groups.

```
n_reason = (
    adsl
    .filter(pl.col("DCREASCD") != "Completed")
    .group_by(["TRT01PN", "DCREASCD"])
    .agg(n = pl.len())
    .join(
        adsl.group_by("TRT01PN").agg(total = pl.len()),
        on="TRT01PN"
    )
    .with_columns([
        pl.concat_str([pl.lit("    "), pl.col("DCREASCD")]).alias("row"),
        (100.0 * pl.col("n") / pl.col("total")).round(1).alias("pct")
    ])
    .pivot(
        index="row",
        on="TRT01PN",
        values=["n", "pct"],
        sort_columns=True
    )
    .with_columns([
        pl.col(["n_0", "n_54", "n_81"]).fill_null(0),
        pl.col(["pct_0", "pct_54", "pct_81"]).fill_null(0.0)
    ])
    .sort("row")
)

n_reason
```

row	n_0	n_54	n_81	pct_0	pct_54	pct_81
str	u32	u32	u32	f64	f64	f64
" Adverse Event"	8	44	40	9.3	52.4	47.6
" Death"	2	1	0	2.3	1.2	0.0

row	n_0	n_54	n_81	pct_0	pct_54	pct_81
str	u32	u32	u32	f64	f64	f64
" I/E Not Met"	1	0	2	1.2	0.0	2.4
" Lack of Efficacy"	3	0	1	3.5	0.0	1.2
" Lost to Follow-up"	1	1	0	1.2	1.2	0.0
" Physician Decision"	1	0	2	1.2	0.0	2.4
" Protocol Violation"	1	1	1	1.2	1.2	1.2
" Sponsor Decision"	2	2	3	2.3	2.4	3.6
" Withdrew Consent"	9	10	8	10.5	11.9	9.5

2.7 Step 6: Combine All Results

Now we stack all our individual summaries together to create the complete disposition table.

Using `pl.concat()`, we combine the enrollment counts, completion counts, discontinuation counts, and detailed discontinuation reasons into a single table that flows logically from top to bottom.

```
tbl_disp = pl.concat([
    n_rand,
    n_complete,
    n_disc,
    n_reason
])

tbl_disp
```

row	n_0	n_54	n_81	pct_0	pct_54	pct_81
str	u32	u32	u32	f64	f64	f64
"Participants in population"	86	84	84	null	null	null
"Completed"	58	25	27	67.4	29.8	32.1
"Discontinued"	28	59	57	32.6	70.2	67.9
" Adverse Event"	8	44	40	9.3	52.4	47.6
" Death"	2	1	0	2.3	1.2	0.0
...
" Lost to Follow-up"	1	1	0	1.2	1.2	0.0
" Physician Decision"	1	0	2	1.2	0.0	2.4
" Protocol Violation"	1	1	1	1.2	1.2	1.2
" Sponsor Decision"	2	2	3	2.3	2.4	3.6

row	n_0	n_54	n_81	pct_0	pct_54	pct_81
str	u32	u32	u32	f64	f64	f64
" Withdrew Consent"	9	10	8	10.5	11.9	9.5

2.8 Step 7: Generate Publication-Ready Output

Finally, we format our table in RTF format using the `rtflite` package.

The `RTFDocument` class handles the complex formatting required for clinical reports, including proper column headers, borders, and spacing. The resulting RTF file can be directly included in regulatory submissions or converted to PDF for review.

```
doc_disp = rtf.RTFDocument(
    df=tbl_disp.select("row", "n_0", "pct_0", "n_54", "pct_54", "n_81", "pct_81"),
    rtf_title=rtf.RTFTitle(text=["Disposition of Participants"]),
    rtf_column_header=[
        rtf.RTFColumnHeader(
            text=["", "Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"],
            col_rel_width=[3] + [2] * 3,
            text_justification=["l"] + ["c"] * 3,
        ),
        rtf.RTFColumnHeader(
            text=["", "n", "(%)", "n", "(%)", "n", "(%)"],
            col_rel_width=[3] + [1] * 6,
            text_justification=["l"] + ["c"] * 6,
            border_top=[""] + ["single"] * 6,
            border_left=["single"] + ["single", ""] * 3
        )
    ],
    rtf_body=rtf.RTFBody(
        col_rel_width=[3] + [1] * 6,
        text_justification=["l"] + ["c"] * 6,
        border_left=["single"] + ["single", ""] * 3
    ),
    rtf_source=rtf.RTFSource(text=["Source: ADSL dataset"]) # Required source attribution
)

doc_disp.write_rtf("rtf/tlf_disposition.rtf") # Save as RTF for submission
```

rtf/tlf_disposition.rtf

3 Study Population Table

3.1 Overview

Clinical trials define multiple analysis populations based on different inclusion criteria. Following [ICH E3 guidance](#), regulatory submissions must clearly document the number of participants in each analysis population to support the validity of statistical analyses.

The key analysis populations typically include:

- **All Randomized:** Total participants who entered the study
- **Intent-to-Treat (ITT):** Participants included in the primary efficacy analysis
- **Efficacy Population:** Participants who meet specific criteria for efficacy evaluation
- **Safety Population:** Participants who received at least one dose of study treatment

This tutorial shows you how to create a population summary table using Python's `rtflite` package.

```
import polars as pl # Data manipulation
import rtflite as rtf # RTF reporting
```

3.2 Step 1: Load Data

We start by loading the Subject-level Analysis Dataset (ADSL), which contains population flags for each participant.

```
adsl = pl.read_parquet("data/adsl.parquet")
```

Let's examine the key population flag variables we'll use:

- **USUBJID:** Unique participant identifier
- **TRT01P:** Planned treatment group
- **ITTFL:** Intent-to-treat population flag (Y/N)
- **EFFFL:** Efficacy population flag (Y/N)
- **SAFFL:** Safety population flag (Y/N)

```
adsl.select(["USUBJID", "TRT01P", "ITTFL", "EFFFL", "SAFFL"])
```

USUBJID	TRT01P	ITTFL	EFFFL	SAFFL
str	str	str	str	str
"01-701-1015"	"Placebo"	"Y"	"Y"	"Y"
"01-701-1023"	"Placebo"	"Y"	"Y"	"Y"
"01-701-1028"	"Xanomeline High Dose"	"Y"	"Y"	"Y"
"01-701-1033"	"Xanomeline Low Dose"	"Y"	"Y"	"Y"
"01-701-1034"	"Xanomeline High Dose"	"Y"	"Y"	"Y"
...
"01-718-1254"	"Xanomeline Low Dose"	"Y"	"Y"	"Y"
"01-718-1328"	"Xanomeline High Dose"	"Y"	"Y"	"Y"
"01-718-1355"	"Placebo"	"Y"	"Y"	"Y"
"01-718-1371"	"Xanomeline High Dose"	"Y"	"Y"	"Y"
"01-718-1427"	"Xanomeline High Dose"	"Y"	"Y"	"Y"

3.3 Step 2: Calculate Treatment Group Totals

First, we calculate the total number of randomized participants in each treatment group, which will serve as the denominator for percentage calculations.

```
totals = adsl.group_by("TRT01P").agg(
    total = pl.len()
)

totals
```

TRT01P	total
str	u32
"Placebo"	86
"Xanomeline High Dose"	84
"Xanomeline Low Dose"	84

3.4 Step 3: Define Helper Function

We create a reusable function to count participants by treatment group for any population subset.

```
def count_by_treatment(data, population_name):
    """Count participants by treatment group and add population label"""
    return data.group_by("TRT01P").agg(
        n = pl.len()
    ).with_columns(
        population = pl.lit(population_name)
    )
```

3.5 Step 4: Count Each Population

Now we calculate participant counts for each analysis population.

3.5.1 All Randomized Participants

```
pop_all = count_by_treatment(
    data=adsl,
    population_name="Participants in population"
)

pop_all
```

TRT01P	n	population
str	u32	str
"Placebo"	86	"Participants in population"
"Xanomeline Low Dose"	84	"Participants in population"
"Xanomeline High Dose"	84	"Participants in population"

3.5.2 Intent-to-Treat Population

```
adsl_itt = adsl.filter(pl.col("ITTFL") == "Y")
pop_itt = count_by_treatment(
    data=adsl_itt,
    population_name="Participants included in ITT population"
)

pop_itt
```


TRT01P	n	population
str	u32	str
"Placebo"	86	"Participants included in ITT p...
"Xanomeline Low Dose"	84	"Participants included in ITT p...
"Xanomeline High Dose"	84	"Participants included in ITT p...

3.5.3 Efficacy Population

```
adsl_eff = adsl.filter(pl.col("EFFFL") == "Y")
pop_eff = count_by_treatment(
    data=adsl_eff,
    population_name="Participants included in efficacy population"
)

pop_eff
```

TRT01P	n	population
str	u32	str
"Placebo"	79	"Participants included in effic...
"Xanomeline Low Dose"	81	"Participants included in effic...
"Xanomeline High Dose"	74	"Participants included in effic...

3.5.4 Safety Population

```
adsl_saf = adsl.filter(pl.col("SAFFL") == "Y")
pop_saf = count_by_treatment(
    data=adsl_saf,
    population_name="Participants included in safety population"
)

pop_saf
```

TRT01P	n	population
str	u32	str
"Xanomeline Low Dose"	84	"Participants included in safet...

TRT01P	n	population
str	u32	str
"Xanomeline High Dose"	84	"Participants included in safet...
"Placebo"	86	"Participants included in safet...

3.6 Step 5: Combine All Populations

We stack all population counts together into a single dataset.

```
all_populations = pl.concat([
    pop_all,
    pop_itt,
    pop_eff,
    pop_saf
])

all_populations
```

TRT01P	n	population
str	u32	str
"Placebo"	86	"Participants in population"
"Xanomeline Low Dose"	84	"Participants in population"
"Xanomeline High Dose"	84	"Participants in population"
"Placebo"	86	"Participants included in ITT p...
"Xanomeline Low Dose"	84	"Participants included in ITT p...
...
"Xanomeline Low Dose"	81	"Participants included in effic...
"Xanomeline High Dose"	74	"Participants included in effic...
"Xanomeline Low Dose"	84	"Participants included in safet...
"Xanomeline High Dose"	84	"Participants included in safet...
"Placebo"	86	"Participants included in safet...

3.7 Step 6: Calculate Percentages

We join with the total counts and calculate what percentage each population represents of the total randomized participants.

```
stats_with_pct = all_populations.join(
  totals,
  on="TRT01P"
).with_columns(
  pct = (100.0 * pl.col("n") / pl.col("total")).round(1)
)

stats_with_pct
```

TRT01P	n	population	total	pct
str	u32	str	u32	f64
"Placebo"	86	"Participants in population"	86	100.0
"Xanomeline Low Dose"	84	"Participants in population"	84	100.0
"Xanomeline High Dose"	84	"Participants in population"	84	100.0
"Placebo"	86	"Participants included in ITT p..."	86	100.0
"Xanomeline Low Dose"	84	"Participants included in ITT p..."	84	100.0
...
"Xanomeline Low Dose"	81	"Participants included in effic..."	84	96.4
"Xanomeline High Dose"	74	"Participants included in effic..."	84	88.1
"Xanomeline Low Dose"	84	"Participants included in safet..."	84	100.0
"Xanomeline High Dose"	84	"Participants included in safet..."	84	100.0
"Placebo"	86	"Participants included in safet..."	86	100.0

3.8 Step 7: Format Display Values

For the final table, we format the display text. The total randomized count shows just “N”, while subset populations show “N (%)”.

```
formatted_stats = stats_with_pct.with_columns(
  display = pl.when(pl.col("population") == "Participants in population")
    .then(pl.col("n").cast(str))
    .otherwise(
      pl.concat_str([
        pl.col("n").cast(str),
        pl.lit(" ("),
        pl.col("pct").round(1).cast(str),
        pl.lit("%")
      ])
    )
)
```

```
)
```

```
formatted_stats
```

TRT01P str	n u32	population str	total u32	pct f64	display str
"Placebo"	86	"Participants in population"	86	100.0	"86"
"Xanomeline Low Dose"	84	"Participants in population"	84	100.0	"84"
"Xanomeline High Dose"	84	"Participants in population"	84	100.0	"84"
"Placebo"	86	"Participants included in ITT p..."	86	100.0	"86 (100.0)"
"Xanomeline Low Dose"	84	"Participants included in ITT p..."	84	100.0	"84 (100.0)"
...
"Xanomeline Low Dose"	81	"Participants included in effic..."	84	96.4	"81 (96.4)"
"Xanomeline High Dose"	74	"Participants included in effic..."	84	88.1	"74 (88.1)"
"Xanomeline Low Dose"	84	"Participants included in safet..."	84	100.0	"84 (100.0)"
"Xanomeline High Dose"	84	"Participants included in safet..."	84	100.0	"84 (100.0)"
"Placebo"	86	"Participants included in safet..."	86	100.0	"86 (100.0)"

3.9 Step 8: Create Final Table

We reshape the data from long format (rows for each treatment-population combination) to wide format (columns for each treatment group).

```
df_overview = formatted_stats.pivot(  
    values="display",  
    index="population",  
    on="TRT01P",  
    maintain_order=True  
) .select(  
    ["population", "Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]  
)
```

```
df_overview
```

population str	Placebo str	Xanomeline Low Dose str	Xanomeline High Dose str
"Participants in population"	"86"	"84"	"84"
"Participants included in ITT p..."	"86 (100.0)"	"84 (100.0)"	"84 (100.0)"

population str	Placebo str	Xanomeline Low Dose str	Xanomeline High Dose str
"Participants included in effic...	"79 (91.9)"	"81 (96.4)"	"74 (88.1)"
"Participants included in safet...	"86 (100.0)"	"84 (100.0)"	"84 (100.0)"

3.10 Step 9: Generate Publication-Ready Output

Finally, we format the population table for regulatory submission using the `rtflite` package.

```
doc_overview = rtf.RTFDocument(
    df=df_overview,
    rtf_title=rtf.RTFTitle(
        text=["Analysis Population", "All Participants Randomized"]
    ),
    rtf_column_header=rtf.RTFColumnHeader(
        text=["", "Placebo\nn (%)", "Xanomeline Low Dose\nn (%)", "Xanomeline High Dose\nn (%)"],
        col_rel_width=[4, 2, 2, 2],
        text_justification=["l", "c", "c", "c"],
    ),
    rtf_body=rtf.RTFBody(
        col_rel_width=[4, 2, 2, 2],
        text_justification=["l", "c", "c", "c"],
    ),
    rtf_source=rtf.RTFSource(text=["Source: ADSL dataset"])
)

doc_overview.write_rtf("rtf/tlf_population.rtf")
```

rtf/tlf_population.rtf

4 Baseline Characteristics Table

4.1 Overview

Baseline characteristics tables summarize the demographic and clinical characteristics of study participants at enrollment. Following [ICH E3 guidance](#), these tables are essential for understanding the study population and assessing comparability between treatment groups.

This tutorial shows you how to create a baseline characteristics table using Python's `rtflite` package.

```
import polars as pl # Data manipulation
import rtflite as rtf # RTF reporting
```

4.2 Step 1: Load Data

We start by loading the Subject-level Analysis Dataset (ADSL) and filtering to the safety population.

```
adsl = (
    pl.read_parquet("data/adsl.parquet")
    .select(["USUBJID", "TRT01P", "AGE", "SEX", "RACE"])
)

adsl
```

USUBJID	TRT01P	AGE	SEX	RACE
str	str	f64	str	str
"01-701-1015"	"Placebo"	63.0	"Female"	"White"
"01-701-1023"	"Placebo"	64.0	"Male"	"White"
"01-701-1028"	"Xanomeline High Dose"	71.0	"Male"	"White"
"01-701-1033"	"Xanomeline Low Dose"	74.0	"Male"	"White"
"01-701-1034"	"Xanomeline High Dose"	77.0	"Female"	"White"
...

USUBJID	TRT01P	AGE	SEX	RACE
str	str	f64	str	str
"01-718-1254"	"Xanomeline Low Dose"	78.0	"Male"	"White"
"01-718-1328"	"Xanomeline High Dose"	86.0	"Male"	"White"
"01-718-1355"	"Placebo"	79.0	"Male"	"White"
"01-718-1371"	"Xanomeline High Dose"	69.0	"Female"	"White"
"01-718-1427"	"Xanomeline High Dose"	74.0	"Female"	"Black Or African American"

4.3 Step 2: Calculate Summary Statistics

We'll create separate functions to handle continuous and categorical variables.

4.3.1 Continuous Variables (Age)

For continuous variables, we calculate mean (SD) and median [min, max].

```
def summarize_continuous(df, var):
    """Calculate summary statistics for continuous variables"""
    return df.groupby("TRT01P").agg([
        pl.col(var).mean().round(1).alias("mean"),
        pl.col(var).std().round(2).alias("sd"),
        pl.col(var).median().alias("median"),
        pl.col(var).min().alias("min"),
        pl.col(var).max().alias("max"),
        pl.len().alias("n")
    ])

age_stats = summarize_continuous(adsl, "AGE")
age_stats
```

TRT01P	mean	sd	median	min	max	n
str	f64	f64	f64	f64	f64	u32
"Xanomeline Low Dose"	75.7	8.29	77.5	51.0	88.0	84
"Xanomeline High Dose"	74.4	7.89	76.0	56.0	88.0	84
"Placebo"	75.2	8.59	76.0	52.0	89.0	86

4.3.2 Categorical Variables (Sex, Race)

For categorical variables, we calculate counts and percentages.

```
def summarize_categorical(df, var):
    """Calculate counts and percentages for categorical variables"""
    # Get counts by treatment and category
    counts = df.groupby(["TRT01P", var]).len()

    # Get treatment totals for percentage calculations
    totals = df.groupby("TRT01P").len().rename({"len": "total"})

    # Calculate percentages
    result = counts.join(totals, on="TRT01P").with_columns([
        (100.0 * pl.col("len") / pl.col("total")).round(1).alias("pct")
    ])

    return result

sex_stats = summarize_categorical(adsl, "SEX")
sex_stats
```

TRT01P	SEX	len	total	pct
str	str	u32	u32	f64
"Xanomeline High Dose"	"Male"	44	84	52.4
"Placebo"	"Male"	33	86	38.4
"Xanomeline Low Dose"	"Female"	50	84	59.5
"Xanomeline Low Dose"	"Male"	34	84	40.5
"Placebo"	"Female"	53	86	61.6
"Xanomeline High Dose"	"Female"	40	84	47.6

```
race_stats = summarize_categorical(adsl, "RACE")
race_stats
```

TRT01P	RACE	len	total	pct
str	str	u32	u32	f64
"Xanomeline High Dose"	"Black Or African American"	9	84	10.7
"Xanomeline High Dose"	"White"	74	84	88.1
"Placebo"	"White"	78	86	90.7

TRT01P	RACE	len	total	pct
str	str	u32	u32	f64
"Xanomeline Low Dose"	"Black Or African American"	6	84	7.1
"Placebo"	"Black Or African American"	8	86	9.3
"Xanomeline Low Dose"	"White"	78	84	92.9
"Xanomeline High Dose"	"American Indian Or Alaska Nati..."	1	84	1.2

4.4 Step 3: Format Results

Now we format the statistics into the standard baseline table format.

4.4.1 Format Age Statistics

```
# Format age as "Mean (SD)" and "Median [Min, Max]"
age_formatted = age_stats.with_columns([
    pl.format("{} ({})", pl.col("mean"), pl.col("sd")).alias("mean_sd"),
    pl.format("{} [{}, {}]", pl.col("median"), pl.col("min"), pl.col("max")).alias("median_range")
]).select(["TRT01P", "mean_sd", "median_range"])

age_formatted
```

TRT01P	mean_sd	median_range
str	str	str
"Xanomeline Low Dose"	"75.7 (8.29)"	"77.5 [51.0, 88.0]"
"Xanomeline High Dose"	"74.4 (7.89)"	"76.0 [56.0, 88.0]"
"Placebo"	"75.2 (8.59)"	"76.0 [52.0, 89.0]"

4.4.2 Format Categorical Statistics

```
# Format categorical as "n (%)"
sex_formatted = sex_stats.with_columns([
    pl.format("{} ({}%)", pl.col("len"), pl.col("pct")).alias("n_pct")
]).select(["TRT01P", "SEX", "n_pct"])

race_formatted = race_stats.with_columns([
```

```
pl.format("{} ({}%)", pl.col("len"), pl.col("pct")).alias("n_pct")
).select(["TRT01P", "RACE", "n_pct"])

sex_formatted
```

TRT01P	SEX	n_pct
str	str	str
"Xanomeline High Dose"	"Male"	"44 (52.4%)"
"Placebo"	"Male"	"33 (38.4%)"
"Xanomeline Low Dose"	"Female"	"50 (59.5%)"
"Xanomeline Low Dose"	"Male"	"34 (40.5%)"
"Placebo"	"Female"	"53 (61.6%)"
"Xanomeline High Dose"	"Female"	"40 (47.6%)"

4.5 Step 4: Create Table Structure

We'll build the table row by row following the standard baseline table format.

```
# Helper function to get value for a treatment group
def get_value(df, treatment):
    """Get value for a specific treatment group or return default"""
    result = df.filter(pl.col("TRT01P") == treatment)
    return result[result.columns[-1]][0] if result.height > 0 else "0 (0.0%)"

# Build the baseline table structure
table_rows = []

# Age section
table_rows.append(["Age (years)", "", "", ""])

# Age Mean (SD) row
age_mean_row = [" Mean (SD)"] + [
    get_value(age_formatted.select(["TRT01P", "mean_sd"]), trt).replace("0 (0.0%)", "")
    for trt in ["Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
]
table_rows.append(age_mean_row)

# Age Median [Min, Max] row
age_median_row = [" Median [Min, Max]"] + [
```

```

        get_value(age_formatted.select(["TRT01P", "median_range"]), trt).replace("0 (0.0%)", "")
    for trt in ["Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
]
table_rows.append(age_median_row)

# Sex section
table_rows.append(["Sex", "", "", ""])

for sex_cat in ["Female", "Male"]:
    sex_data = sex_formatted.filter(pl.col("SEX") == sex_cat)
    sex_row = [f" {sex_cat}"] + [
        get_value(sex_data, trt)
        for trt in ["Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
    ]
    table_rows.append(sex_row)

# Race section
table_rows.append(["Race", "", "", ""])

for race_cat in ["White", "Black Or African American", "American Indian Or Alaska Native"]:
    race_data = race_formatted.filter(pl.col("RACE") == race_cat)
    race_row = [f" {race_cat}"] + [
        get_value(race_data, trt)
        for trt in ["Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
    ]
    table_rows.append(race_row)

# Create DataFrame from table rows
baseline_table = pl.DataFrame(
    table_rows,
    schema=["Characteristic", "Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"],
    orient="row"
)

baseline_table

```

Characteristic str	Placebo str	Xanomeline Low Dose str	Xanomeline High Dose str
"Age (years)"	"	"	"
" Mean (SD)"	"75.2 (8.59)"	"75.7 (8.29)"	"74.4 (7.89)"
" Median [Min, Max]"	"76.0 [52.0, 89.0]"	"77.5 [51.0, 88.0]"	"76.0 [56.0, 88.0]"

Characteristic str	Placebo str	Xanomeline Low Dose str	Xanomeline High Dose str
"Sex"	" "	" "	" "
" Female"	"53 (61.6%)"	"50 (59.5%)"	"40 (47.6%)"
" Male"	"33 (38.4%)"	"34 (40.5%)"	"44 (52.4%)"
"Race"	" "	" "	" "
" White"	"78 (90.7%)"	"78 (92.9%)"	"74 (88.1%)"
" Black Or African American"	"8 (9.3%)"	"6 (7.1%)"	"9 (10.7%)"
" American Indian Or Alaska Na..."	"0 (0.0%)"	"0 (0.0%)"	"1 (1.2%)"

4.6 Step 5: Generate Publication-Ready Output

Finally, we format the baseline table for regulatory submission using the `rtflite` package.

```
# Get treatment group sizes for column headers
treatment_n = adsl.group_by("TRT01P").len().sort("TRT01P")
n_placebo = treatment_n.filter(pl.col("TRT01P") == "Placebo")["len"][0]
n_low = treatment_n.filter(pl.col("TRT01P") == "Xanomeline Low Dose")["len"][0]
n_high = treatment_n.filter(pl.col("TRT01P") == "Xanomeline High Dose")["len"][0]

doc_baseline = rtf.RTFDocument(
    df=baseline_table,
    rtf_title=rtf.RTFTitle(
        text=[
            "Baseline Characteristics of Participants",
            "(All Participants Randomized)"
        ]
    ),
    rtf_column_header=rtf.RTFColumnHeader(
        text=[
            "Characteristic",
            f"Placebo\n(N={n_placebo})",
            f"Xanomeline Low Dose\n(N={n_low})",
            f"Xanomeline High Dose\n(N={n_high})"
        ],
        text_justification=["l", "c", "c", "c"],
        col_rel_width=[3, 2, 2, 2]
    ),
    rtf_body=rtf.RTFBody(
        text_justification=["l", "c", "c", "c"],
```

```
        col_rel_width=[3, 2, 2, 2]
    ),
    rtf_source=rtf.RTFSource(text=["Source: ADSL dataset"])
)

doc_baseline.write_rtf("rtf/tlf_baseline.rtf") # Save as RTF for submission
```

rtf/tlf_baseline.rtf

5 Adverse Events Summary Table

This article demonstrates how to create an adverse events (AE) summary table for clinical study reports using rtflite.

5.1 Overview

Adverse events summary tables are critical safety assessments in clinical trials. They typically show the number and percentage of participants experiencing various categories of adverse events by treatment group.

5.2 Imports

```
import polars as pl
import rtflite as rtf
```

5.3 Data Preparation

```
# Load and prepare safety population data
adsl = pl.read_parquet("data/adsl.parquet")
adae = pl.read_parquet("data/adae.parquet")

# Safety population
adsl_safety = adsl.filter(pl.col("SAFFL") == "Y").select(["USUBJID", "TRT01A"])
adae_safety = adae.join(adsl_safety, on="USUBJID").with_columns(pl.col("TRT01A").alias("TRTA"))
```

5.4 AE Summary Calculations

```
# Define AE categories with their filter conditions
ae_categories = {
    "Participants in population": None, # Special case - uses total N
    "With any adverse event": pl.lit(True), # All AEs
    "With drug-related adverse event": pl.col("AEREL").is_in(["POSSIBLE", "PROBABLE", "DEFINITE", "RELATED"]),
    "With serious adverse event": pl.col("AESER") == "Y",
    "With serious drug-related adverse event": (
        (pl.col("AESER") == "Y") &
        pl.col("AEREL").is_in(["POSSIBLE", "PROBABLE", "DEFINITE", "RELATED"])
    ),
    "Who died": pl.col("AEOUT") == "FATAL",
    "Discontinued due to adverse event": pl.col("AEACN") == "DRUG WITHDRAWN"
}

# Calculate population totals
pop_counts = adsl_safety.group_by("TRT01A").agg(pl.len().alias("N"))

# Calculate AE counts for each category
results = []
for category, filter_expr in ae_categories.items():
    if category == "Participants in population":
        # Special handling for population row
        for row in pop_counts.iter_rows(named=True):
            results.append({
                "Category": category,
                "TRT01A": row["TRT01A"],
                "n": row["N"],
                "pct_display": ""
            })
    else:
        # Count unique subjects meeting criteria
        ae_counts = (
            adae_safety
            .filter(filter_expr)
            .group_by("TRTA")
            .agg(pl.col("USUBJID").n_unique().alias("n"))
        )

        # Join with population to calculate percentages
```

```

merged = (
    pop_counts
    .join(ae_counts, left_on="TRT01A", right_on="TRTA", how="left")
    .with_columns([
        pl.col("n").fill_null(0),
        (100.0 * pl.col("n").fill_null(0) / pl.col("N")).round(1).alias("pct"),
        pl.when(pl.col("n").fill_null(0) > 0)
            .then(pl.concat_str([pl.lit("("), (100.0 * pl.col("n").fill_null(0) / pl.col("N")).round(1).alias("pct"), pl.lit(")")]
            .otherwise(pl.lit("(0.0)"))
            .alias("pct_display")
    ])
)

for row in merged.iter_rows(named=True):
    results.append({
        "Category": category,
        "TRT01A": row["TRT01A"],
        "n": row["n"],
        "pct_display": row["pct_display"]
    })

ae_summary = pl.DataFrame(results)

```

5.5 Format AE Summary Table

```

# Define treatment order and category order for consistent display
treatments = ["Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
category_order = list(ae_categories.keys())

# Pivot to wide format with separate n and (%) columns
df_ae_summary = (
    ae_summary
    .with_columns(pl.col("Category").cast(pl.Enum(category_order)))
    .pivot(
        values=["n", "pct_display"],
        index="Category",
        on="TRT01A"
    )
    .with_columns([pl.col(f"n_{trt}").cast(str) for trt in treatments])
)

```



```

    .select(
      ["Category"] +
      [col for trt in treatments for col in [f"n_{trt}", f"pct_display_{trt}"]]
    )
    .rename({"Category": ""})
  )
df_ae_summary

```

enum	n_Placebo str	pct_display_Placebo str	n_Xanomeline Low Dose str	pct_disp str
"Participants in population"	"86"	" "	"84"	" "
"With any adverse event"	"69"	"(80.2)"	"77"	"(91.7)"
"With drug-related adverse even..."	"44"	"(51.2)"	"73"	"(86.9)"
"With serious adverse event"	"0"	"(0.0)"	"1"	"(1.2)"
"With serious drug-related adve..."	"0"	"(0.0)"	"1"	"(1.2)"
"Who died"	"2"	"(2.3)"	"1"	"(1.2)"
"Discontinued due to adverse ev..."	"0"	"(0.0)"	"0"	"(0.0)"

5.6 Create RTF Output

```

# Create RTF document
doc_ae_summary = rtf.RTFDocument(
  df=df_ae_summary,
  rtf_title=rtf.RTFTitle(
    text=[
      "Analysis of Adverse Event Summary",
      "(Safety Analysis Population)"
    ]
  ),
  rtf_column_header=[
    rtf.RTFColumnHeader(
      text = [""] + treatments,
      col_rel_width=[4, 2, 2, 2],
      text_justification=["l", "c", "c", "c"],
    ),
    rtf.RTFColumnHeader(
      text=[

```

```

        "",          # Empty for first column
        "n", "(%)",  # Placebo columns
        "n", "(%)",  # Low Dose columns
        "n", "(%)",  # High Dose columns
    ],
    col_rel_width=[4] + [1] * 6,
    text_justification=["l"] + ["c"] * 6,
    border_left = ["single"] + ["single", ""] * 3,
    border_top = [""] + ["single"] * 6
)
],
rtf_body=rtf.RTFBody(
    col_rel_width=[4] + [1] * 6,
    text_justification=["l"] + ["c"] * 6,
    border_left = ["single"] + ["single", ""] * 3
),
rtf_footnote=rtf.RTFFootnote(
    text=[
        "Every subject is counted a single time for each applicable row and column."
    ]
),
rtf_source=rtf.RTFSource(
    text=["Source: ADSL and ADAE datasets"],
)
)

# Write RTF file
doc_ae_summary.write_rtf("rtf/tlf_ae_summary.rtf")

```

rtf/tlf_ae_summary.rtf

PosixPath('pdf/tlf_ae_summary.pdf')

6 Specific Adverse Events Table

This article demonstrates how to create a specific adverse events table by System Organ Class and Preferred Term.

6.1 Setup

```
import polars as pl
import rtflite as rtf

adsl = pl.read_parquet("data/adsl.parquet")
adae = pl.read_parquet("data/adae.parquet")
treatments = ["Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
```

6.2 Prepare AE Summary Data

```
# Get safety population counts and AE data
adsl_safety = adsl.filter(pl.col("SAFFL") == "Y").select(["USUBJID", "TRT01A"])
adae_safety = adae.join(adsl_safety, on="USUBJID", how="inner")
pop_counts = adsl_safety.group_by("TRT01A").agg(N=pl.len()).sort("TRT01A")

# Calculate AE counts by SOC and term
ae_counts = (
    adae_safety.with_columns(pl.col("AEDECOD").str.to_titlecase())
    .group_by(["TRT01A", "AEBODSYS", "AEDECOD"])
    .agg(n=pl.col("USUBJID").n_unique())
    .sort(["AEBODSYS", "AEDECOD", "TRT01A"])
)

# Build table rows
table_data = [
    ["Participants in population"] + [str(pop_counts.filter(pl.col("TRT01A") == t)["N"][0])]
```

```

    [""] * 4 # Blank row
]

# Add SOC and AE term rows
for soc in ae_counts["AEBODSYS"].unique().sort():
    table_data.append([soc] + [""] * 3)
    soc_data = ae_counts.filter(pl.col("AEBODSYS") == soc)

    for ae in soc_data["AEDECOD"].unique().sort():
        row = [f" {ae}"]
        for trt in treatments:
            count = soc_data.filter((pl.col("AEDECOD") == ae) & (pl.col("TRT01A") == trt))
            row.append(str(count["n"][0]) if count.height > 0 else "0")
        table_data.append(row)

df_ae_specific = pl.DataFrame(table_data, schema=[""] + treatments, orient="row")

```

6.3 Create RTF Output

```

doc_ae_specific = rtf.RTFDocument(
    df=df_ae_specific,
    rtf_title=rtf.RTFTitle(text=["Specific Adverse Events", "(Safety Analysis Population)"]),
    rtf_column_header=rtf.RTFColumnHeader(
        text=["", "Placebo\n\n", "Xanomeline Low Dose\n\n", "Xanomeline High Dose\n\n"],
        col_rel_width=[4, 1, 1, 1],
        text_justification=["l", "c", "c", "c"],
    ),
    rtf_body=rtf.RTFBody(
        col_rel_width=[4, 1, 1, 1],
        text_justification=["l", "c", "c", "c"],
        text_font_style=lambda df, i, j: "bold" if j == 0 and " " not in str(df[i, j]) else "",
    ),
    rtf_footnote=rtf.RTFFootnote(text=["Number of participants with specific adverse events."]),
    rtf_source=rtf.RTFSource(text=["Source: ADSL and ADAE datasets"])
)

doc_ae_specific.write_rtf("rtf/tlf_ae_specific.rtf")

```

rtf/tlf_ae_specific.rtf

```
PosixPath('pdf/tlf_ae_specific.pdf')
```

7 ANCOVA Efficacy Analysis

This article demonstrates how to create an ANCOVA efficacy table for glucose levels at Week 24 with LOCF imputation.

7.1 Setup

```
import polars as pl
import rtflite as rtf
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
from scipy import stats as scipy_stats
from importlib.resources import files

adsl = pl.read_parquet("data/adsl.parquet")
adlbc = pl.read_parquet("data/adlbc.parquet")
treatments = ["Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
```

7.2 Prepare Analysis Data

```
# Clean data types and filter for efficacy population
adlbc_clean = adlbc.with_columns(
    [pl.col(c).cast(str).str.strip_chars() for c in ["USUBJID", "PARAMCD", "AVISIT", "TRTP"]]
)
adsl_eff = adsl.filter(pl.col("EFFFL") == "Y").select(["USUBJID"])
adlbc_eff = adlbc_clean.join(adsl_eff, on="USUBJID", how="inner")

# Apply LOCF for glucose data up to Week 24
gluc_data = (
    adlbc_eff.filter((pl.col("PARAMCD") == "GLUC") & (pl.col("AVISITN") <= 24))
    .sort(["USUBJID", "AVISITN"])
```

```

.group_by("USUBJID")
.agg([
    pl.col("TRTP").first(),
    pl.col("BASE").first(),
    pl.col("AVAL").filter(pl.col("AVISITN") == 0).first().alias("Baseline"),
    pl.col("AVAL").last().alias("Week 24")
])
.filter(pl.col("Baseline").is_not_null() & pl.col("Week 24").is_not_null())
.with_columns((pl.col("Week 24") - pl.col("Baseline")).alias("CHG"))
)

# Calculate descriptive statistics
desc_stats = []
for trt in treatments:
    trt_data = gluc_data.filter(pl.col("TRTP") == trt)
    baseline_full = adlbc_eff.filter(
        (pl.col("PARAMCD") == "GLUC") & (pl.col("AVISIT") == "Baseline") & (pl.col("TRTP") ==
    )

    desc_stats.append({
        "Treatment": trt,
        "N_Baseline": baseline_full.height,
        "Baseline_Mean": baseline_full["AVAL"].mean() if baseline_full.height > 0 else np.nan,
        "Baseline_SD": baseline_full["AVAL"].std() if baseline_full.height > 0 else np.nan,
        "N_Week24": trt_data.height,
        "Week24_Mean": trt_data["Week 24"].mean() if trt_data.height > 0 else np.nan,
        "Week24_SD": trt_data["Week 24"].std() if trt_data.height > 0 else np.nan,
        "N_Change": trt_data.height,
        "Change_Mean": trt_data["CHG"].mean() if trt_data.height > 0 else np.nan,
        "Change_SD": trt_data["CHG"].std() if trt_data.height > 0 else np.nan
    })

# Perform ANCOVA
ancova_df = gluc_data.to_pandas()
ancova_df["TRTP"] = pd.Categorical(ancova_df["TRTP"], categories=treatments)
model = smf.ols("CHG ~ TRTP + BASE", data=ancova_df).fit()

# Calculate LS means and confidence intervals
base_mean = ancova_df["BASE"].mean()
var_cov = model.cov_params()
ls_means = []

```

```

for i, trt in enumerate(treatments):
    x_pred = np.array([1, int(i==1), int(i==2), base_mean])
    ls_mean = model.predict(pd.DataFrame({"TRTP": [trt], "BASE": [base_mean]}))[0]
    se_pred = np.sqrt(x_pred @ var_cov @ x_pred.T)

    ls_means.append({
        "Treatment": trt,
        "LS_Mean": ls_mean,
        "CI_Lower": ls_mean - 1.96 * se_pred,
        "CI_Upper": ls_mean + 1.96 * se_pred
    })

```

7.3 Create Tables for RTF Output

```

# Table 1: Descriptive Statistics
tbl1_data = [
    [
        s["Treatment"],
        str(s["N_Baseline"]),
        f"{s['Baseline_Mean']:.1f} ({s['Baseline_SD']:.2f})",
        str(s["N_Week24"]),
        f"{s['Week24_Mean']:.1f} ({s['Week24_SD']:.2f})",
        str(s["N_Change"]),
        f"{s['Change_Mean']:.1f} ({s['Change_SD']:.2f})",
        f"{ls['LS_Mean']:.2f} ({ls['CI_Lower']:.2f}, {ls['CI_Upper']:.2f})"
    ]
    for s, ls in zip(desc_stats, ls_means)
]

tbl1 = pl.DataFrame(tbl1_data, orient="row", schema=[
    "Treatment", "N_Base", "Mean_SD_Base", "N_Wk24", "Mean_SD_Wk24",
    "N_Chg", "Mean_SD_Chg", "LS_Mean_CI"
])

# Table 2: Pairwise Comparisons
tbl2_data = []
for comp_name, trt_name in [("Xanomeline Low Dose - Placebo", "TRTP[T.Xanomeline Low Dose]"),
                             ("Xanomeline High Dose - Placebo", "TRTP[T.Xanomeline High Dose]"):
    coef = model.params[trt_name]
    se = model.bse[trt_name]

```



```

t_stat = coef / se
p_value = 2 * (1 - scipy_stats.t.cdf(abs(t_stat), model.df_resid))

tbl2_data.append([
    comp_name,
    f"{coef:.2f} ({coef - 1.96*se:.2f}, {coef + 1.96*se:.2f})",
    f"{p_value:.3f}"
])

tbl2 = pl.DataFrame(tbl2_data, orient="row", schema=["Comparison", "Diff_CI", "P_Value"])

```

7.4 Create RTF Document

```

# Create RTF document with two sections
doc_ancova = rtf.RTFDocument(
    df=[tbl1, tbl2],
    rtf_title=rtf.RTFTitle(text=[
        "ANCOVA of Change from Baseline Glucose (mmol/L) at Week 24", "LOCF",
        "Efficacy Analysis Population"
    ]),
    rtf_column_header=[
        [rtf.RTFColumnHeader(text=["", "Baseline", "Week 24", "Change from Baseline"],
            col_rel_width=[3, 2, 2, 4], text_justification=["l", "c", "c", "c"],
            border_bottom="single"),
        rtf.RTFColumnHeader(text=["Treatment", "N", "Mean (SD)", "N", "Mean (SD)", "N",
            "Mean (SD)", "LS Mean (95% CI){^a}"],
            col_rel_width=[3, 0.7, 1.3, 0.7, 1.3, 0.7, 1.3, 2],
            text_justification=["l"] + ["c"] * 7, border_bottom="single")],
        [rtf.RTFColumnHeader(text=["Pairwise Comparison", "Difference in LS Mean (95% CI){^a}"],
            col_rel_width=[5, 4, 2], text_justification=["l", "c", "c"])]
    ],
    rtf_body=[
        rtf.RTFBody(col_rel_width=[3, 0.7, 1.3, 0.7, 1.3, 0.7, 1.3, 2],
            text_justification=["l"] + ["c"] * 7),
        rtf.RTFBody(col_rel_width=[5, 4, 2], text_justification=["l", "c", "c"])
    ],
    rtf_footnote=rtf.RTFFootnote(text=[
        "{^a}Based on an ANCOVA model after adjusting baseline value. LOCF approach is used t
        "ANCOVA = Analysis of Covariance, LOCF = Last Observation Carried Forward",
        "CI = Confidence Interval, LS = Least Squares, SD = Standard Deviation"
    ])

```

```
    ]),  
    rtf_source=rtf.RTFSource(text=["Source: ADLBC dataset"])  
)  
  
doc_ancova.write_rtf("rtf/tlf_efficacy_ancova.rtf")
```

rtf/tlf_efficacy_ancova.rtf

PosixPath('pdf/tlf_efficacy_ancova.pdf')

8 Summary

In summary, this book has no content whatsoever.

1 + 1

2

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111.