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 **14**
3.1 Setup . . . 14
3.2 Load Data . . . 14
3.3 Create Population Table . . . 14
3.4 Step 2: Calculate counts for each population flag . . . 15
3.5 Step 3: Stack all populations together . . . 17
3.6 Step 4: Add total N and calculate percentages . . . 17
3.7 Step 5: Format the display text . . . 18
3.8 Step 6: Pivot from long to wide format . . . 19
3.9 Create RTF Output . . . 19

4 Baseline Characteristics Table **21**
4.1 Overview . . . 21
4.2 Imports . . . 21
4.3 Data Preparation . . . 21
4.4 Statistics Function . . . 22
4.5 Build Table Data . . . 23
4.6 Create RTF Output . . . 25

5 Adverse Events Summary Table **26**
5.1 Overview . . . 26
5.2 Imports . . . 26
5.3 Data Preparation . . . 26
5.4 AE Summary Calculations . . . 27
5.5 Format AE Summary Table . . . 28
5.6 Create RTF Output . . . 29

6 Specific Adverse Events Table **31**
6.1 Setup . . . 31
6.2 Prepare AE Summary Data . . . 31
6.3 Create RTF Output . . . 32

7 ANCOVA Efficacy Analysis **34**
7.1 Setup . . . 34
7.2 Prepare Analysis Data . . . 34
7.3 Create Tables for RTF Output . . . 36
7.4 Create RTF Document . . . 37

8 Summary **39**

References **40**

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 needs 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 str	TRT01P str	TRT01PN i64	DISCONFL str	DCREASCD 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 applica
    ])
    .pivot(
        index="row",
        on="TRT01PN",
        values=["n", "pct"],
        sort_columns=True
    )
)
```

```
n_rand
```

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

row str	n_0 u32	n_54 u32	n_81 u32	pct_0 f64	pct_54 f64	pct_81 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 str	n_0 u32	n_54 u32	n_81 u32	pct_0 f64	pct_54 f64	pct_81 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 str	n_0 u32	n_54 u32	n_81 u32	pct_0 f64	pct_54 f64	pct_81 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

This article demonstrates how to create an analysis population overview table.

3.1 Setup

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

3.2 Load Data

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

3.3 Create Population Table

```
# Step 1: Calculate total N for each treatment (denominator for percentages)
totals = adsl.group_by("TRT01P").agg(
    total = pl.len()
)

totals
```

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

3.4 Step 2: Calculate counts for each population flag

```
# Simple function to count participants by treatment group
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)
    )
```

```
# Row 1: Total participants (no filter needed)
pop_all = count_by_treatment(
    data=adsl,
    population_name="Participants in population"
)

pop_all
```

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

```
# Row 2: ITT population (filter first, then count)
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 str	n u32	population str
”Xanomeline High Dose”	84	”Participants included in ITT p…
”Placebo”	86	”Participants included in ITT p…

TRT01P str	n u32	population str
”Xanomeline Low Dose”	84	”Participants included in ITT p…

```
# Row 3: Efficacy population (filter first, then count)
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 str	n u32	population str
”Placebo”	79	”Participants included in effic…
”Xanomeline High Dose”	74	”Participants included in effic…
”Xanomeline Low Dose”	81	”Participants included in effic…

```
# Row 4: Safety population (filter first, then count)
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 str	n u32	population str
”Xanomeline High Dose”	84	”Participants included in safet…
”Placebo”	86	”Participants included in safet…
”Xanomeline Low Dose”	84	”Participants included in safet…

3.5 Step 3: Stack all populations together

```
# Combine all population dataframes into one
all_populations = pl.concat([
    pop_all,
    pop_itt,
    pop_eff,
    pop_saf
])

all_populations
```

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

3.6 Step 4: Add total N and calculate percentages

```
# Join with totals to calculate percentages
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 str	n u32	population str	total u32	pct f64
”Xanomeline High Dose”	84	”Participants in population”	84	100.0
”Placebo”	86	”Participants in population”	86	100.0
”Xanomeline Low Dose”	84	”Participants in population”	84	100.0
”Xanomeline High Dose”	84	”Participants included in ITT p…	84	100.0
”Placebo”	86	”Participants included in ITT p…	86	100.0
…	…	…	…	…
”Xanomeline High Dose”	74	”Participants included in effic…	84	88.1
”Xanomeline Low Dose”	81	”Participants included in effic…	84	96.4
”Xanomeline High Dose”	84	”Participants included in safet…	84	100.0
”Placebo”	86	”Participants included in safet…	86	100.0
”Xanomeline Low Dose”	84	”Participants included in safet…	84	100.0

3.7 Step 5: Format the display text

```
# First row shows just N, other rows show "N (pct)"
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
”Xanomeline High Dose”	84	”Participants in population”	84	100.0	”84”
”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 included in ITT p…	84	100.0	”84 (100.0)”

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

3.8 Step 6: Pivot from long to wide format

```
# Transform to wide format for final table
df_overview = formatted_stats.pivot(
    values="display",
    index="population",
    on="TRT01P",
    maintain_order=True
).select(
    ["population", "Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
).rename(
    {"population": ""}
)

df_overview
```

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)”
”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)”

3.9 Create RTF Output

```
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
```

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

4 Baseline Characteristics Table

This article demonstrates how to create a baseline characteristics table for clinical study reports using rtflite.

4.1 Overview

Baseline characteristics tables summarize demographic and clinical characteristics of study participants at enrollment. These tables are essential for understanding the study population and assessing comparability between treatment groups.

4.2 Imports

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

4.3 Data Preparation

```
adsl_baseline = (
    pl.read_parquet("data/adsl.parquet")
    .filter(pl.col("SAFFL") == "Y")
    .select(["USUBJID", "TRT01P", "AGE", "SEX", "RACE"])
    .with_columns([
        pl.col("SEX").replace({"F": "Female", "M": "Male"}),
        pl.col("RACE").str.to_titlecase()
    ])
)
```

4.4 Statistics Function

```
def get_statistics(df, var, is_continuous=False):
    expr = [
            pl.col(var).mean().round(1).alias("mean"),
            pl.col(var).std().round(1).alias("sd"),
            pl.col(var).median().round(1).alias("median"),
            pl.col(var).min().alias("min"),
            pl.col(var).max().alias("max")
        ]

    if is_continuous:
        # Continuous statistics by treatment
        by_treatment = df.group_by("TRT01P").agg(expr)

        # Overall statistics
        overall = df.select(expr).row(0)

        return by_treatment, overall
    else:
        # Categorical counts and percentages
        total_n = df.height
        by_treatment = (
            df.group_by(["TRT01P", var])
            .len()
            .join(df.group_by("TRT01P").len().rename({"len": "total"}), on="TRT01P")
            .with_columns(
                pl.format("{} ({}%)",
                    pl.col("len"),
                    (100 * pl.col("len") / pl.col("total")).round(1)
                ).alias("formatted")
            )
        )

        # Overall counts
        overall = (
            df.group_by(var)
            .len()
            .with_columns(
                pl.format("{} ({}%)",
                    pl.col("len"),
```

```
                    (100 * pl.col("len") / total_n).round(1)
                ).alias("formatted")
            )
        )

        return by_treatment, overall
```

4.5 Build Table Data

```
# Treatment groups and counts
treatments = ["Placebo", "Xanomeline Low Dose", "Xanomeline High Dose"]
treatment_counts = dict(
    adsl_baseline.group_by("TRT01P").len().iter_rows()
)

def create_variable_rows(df, var_name, categories=None, is_continuous=False):
    rows = [[var_name, "", "", "", ""]]

    by_treatment, overall_stats = get_statistics(df, var_name, is_continuous)

    if is_continuous:
        # Mean (SD) row
        row = ["  Mean (SD)"]
        for trt in treatments:
            trt_stats = by_treatment.filter(pl.col("TRT01P") == trt)
            if trt_stats.height > 0:
                mean, sd = trt_stats.select(["mean", "sd"]).row(0)
                row.append(f"{mean} ({sd})")
            else:
                row.append("")
        row.append(f"{overall_stats[0]} ({overall_stats[1]})")
        rows.append(row)

        # Median [Min, Max] row
        row = ["  Median [Min, Max]"]
        for trt in treatments:
            trt_stats = by_treatment.filter(pl.col("TRT01P") == trt)
            if trt_stats.height > 0:
                median, min_val, max_val = trt_stats.select(["median", "min", "max"]).row(0)
```

```
                row.append(f"{median} [{min_val}, {max_val}]")
            else:
                row.append("")
        row.append(f"{overall_stats[2]} [{overall_stats[3]}, {overall_stats[4]}]")
        rows.append(row)
    else:
        # Categorical variable rows
        for cat in categories:
            row = [f"  {cat}"]

            for trt in treatments:
                trt_data = by_treatment.filter(
                    (pl.col("TRT01P") == trt) & (pl.col(var_name) == cat)
                )
                if trt_data.height > 0:
                    row.append(trt_data["formatted"][0])
                else:
                    row.append("0 (0.0%)")

            # Overall column
            overall_data = overall_stats.filter(pl.col(var_name) == cat)
            if overall_data.height > 0:
                row.append(overall_data["formatted"][0])
            else:
                row.append("0 (0.0%)")
            rows.append(row)

    return rows

# Build complete table
table_data = []
table_data.extend(create_variable_rows(adsl_baseline, "SEX", ["Female", "Male"]))
table_data.extend(create_variable_rows(adsl_baseline, "AGE", is_continuous=True))
table_data.extend(create_variable_rows(
    adsl_baseline, "RACE",
    ["Black Or African American", "White", "American Indian Or Alaska Native"]
))

df_baseline = pl.DataFrame(table_data, orient="row")

df_baseline
```

column_0 str	column_1 str	column_2 str	column_3 str	column_4 str
"SEX"	""	""	""	""
" Female"	"53 (61.6%)"	"50 (59.5%)"	"40 (47.6%)"	"143 (56.3%
" Male"	"33 (38.4%)"	"34 (40.5%)"	"44 (52.4%)"	"111 (43.7%
"AGE"	""	""	""	""
" Mean (SD)"	"75.2 (8.6)"	"75.7 (8.3)"	"74.4 (7.9)"	"75.1 (8.2)"
" Median [Min, Max]"	"76.0 [52.0, 89.0]"	"77.5 [51.0, 88.0]"	"76.0 [56.0, 88.0]"	"77.0 [51.0,
"RACE"	""	""	""	""
" Black Or African American"	"8 (9.3%)"	"6 (7.1%)"	"9 (10.7%)"	"23 (9.1%)"
" White"	"78 (90.7%)"	"78 (92.9%)"	"74 (88.1%)"	"230 (90.6%
" American Indian Or Alaska Na…	"0 (0.0%)"	"0 (0.0%)"	"1 (1.2%)"	"1 (0.4%)"

4.6 Create RTF Output

```
# Column headers with N counts
col_headers = [""] + [f"{trt}\n(N={treatment_counts[trt]})" for trt in treatments] + [f"Overa

doc_baseline = rtf.RTFDocument(
    df=df_baseline,
    rtf_title=rtf.RTFTitle(
        text=["Baseline Characteristics of Participants", "(All Participants Randomized)"]
    ),
    rtf_column_header=rtf.RTFColumnHeader(
        text=col_headers,
        text_justification=["l"] + ["c"] * 4
    ),
    rtf_body=rtf.RTFBody(
        col_rel_width=[2] + [1] * 4,
        text_justification=["l"] + ["c"] * 4
    )
)

doc_baseline.write_rtf("rtf/tlf_baseline.rtf")
```

```
rtf/tlf_baseline.rtf
```

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

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", "DEFINI
    "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.co
                  .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\nn", "Xanomeline Low Dose\nn", "Xanomeline High Dose\nn"],
        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
         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
        "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.