PHUSE 2024 - Paper OS11

Advances in Clinical Table Creation - Python Updates

Phil Bowsher, Posit/RStudio PBC;
Richard Iannone, Posit/RStudio PBC;
Michael Chow, Posit/RStudio PBC;

Abstract

This paper will introduce the Great Tables Python package for creating complex tables often used in clinical trials reporting. The main Great Tables site for reference is here: https://posit-dev.github.io/great-tables/articles/intro.html

The Great Tables package helps statistical programmers create wonderful-looking tables using the Python programming language. It is based on learnings and developments in creating the R package gt. TLGs (tables, listings, and graphs - also known as TLFs/TFLs with F for figures) are an important part of clinical trial reporting. The goal of this paper is to introduce statistical programmers to the Great Tables package for table creation in Python for submissions to regulatory bodies. This paper is aimed at statistical programmers that are building clinical tables and have wondered how to approach TLGs with the Great Tables Python package. You can find all of the examples in this paper as well as the Phuse Conference presentation slides here:
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables

A presentation about using Great Tables can be found here:
https://www.youtube.com/watch?v=M5zwlb8OzS0&ab_channel=PositPBC

You can test out Great Tables live here via Shinylive Editor here: https://shinylive.io/py/examples/

The examples below are maintained at the following github link:
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables

Introduction

Many pharmaceutical companies are migrating to Next-Gen Statistical Computing Environments (SCE) where the backbone of clinical reporting is in open source languages. Roche recently discussed this journey in the webinar “Roche's End-to-End R Journey to Submission”: https://www.youtube.com/watch?v=BIJNILSoZIM&ab_channel=PositPBC

A companion webinar is the Roche Keynote at R in Pharma 2023 by James Black where he discusses “The importance of the SCE in enabling our shift to open-source data science”:
https://youtu.be/XO-oPX1TBuw?list=PLMtxz1fUYA5DiGZTLx8WmVni85Ps2fjMg&t=1653

In the talk, James highlights the use of Python for workflow management with systems like snakemake. He also discusses a future case where clinical reporting SCEs are generalized into data science SCEs to support incoming new-hires with experience in open source (R, Python etc.) and other drug development use-cases like Real-World Evidence. Moreover, many organizations use data platforms (Databricks, Snowflake etc.) where ETL and data ingestion processes are often in Python and access clinical trial and real world data through such platforms.

Given the large Python community and as more submissions use real world data, like Tacrolimus (https://www.accessdata.fda.gov/drugsatfda_docs/nda/2022/050708Orig1s053;%20050709Orig1s045;%20210115Orig1s005.pdf) by Astellas Pharma US, it seems likely that the role of multilingual programming (R, Python etc.) in drug development will increase. You can see a list of submission documents here that incorporate Python:
https://github.com/philbowsher/Open-Source-in-New-Drug-Applications-NDAs-FDA

Many pharmaceutical companies use or are migrating to R for TLGs given the robust packages available such as rtables, gt and gtsummary. However, when using Python for tables, UCB wrote “when it comes to creating outputs with Python, we encountered some challenges”:
https://phuse.s3.eu-central-1.amazonaws.com/Archive/2023/Connect/EU/Birmingham/PAP_DH02.pdf

Replicating complex tables in open source has been improving. Posit sought to help in R with the creation of the gt package in 2018 as explained here:
https://github.com/philbowsher/Clinical-Tables-in-R-with-gt/blob/master/R%20Tables%20via%20GT%20for%20Regulatory%20Submissions.pdf

While Python has several libraries to perform data analysis such as Pandas and Polars, creating complex tables in Python, like those often seen in R with gt, was laborious. You can see many recent pharma papers highlighting the power of Python for data science and interactive visualizations here:
https://github.com/philbowsher/Use-of-Python-in-regulatory-submission-related-work

As organizations continue to migrate to multilingual environments and integrate tools like parquet, git, and Docker,

they are ever focused on creating language-agnostic frameworks where users/developers can use various tools. This paper will highlight the use of Great Tables for creating wonderful-looking tables in Python in such an environment!

Understanding Great Tables

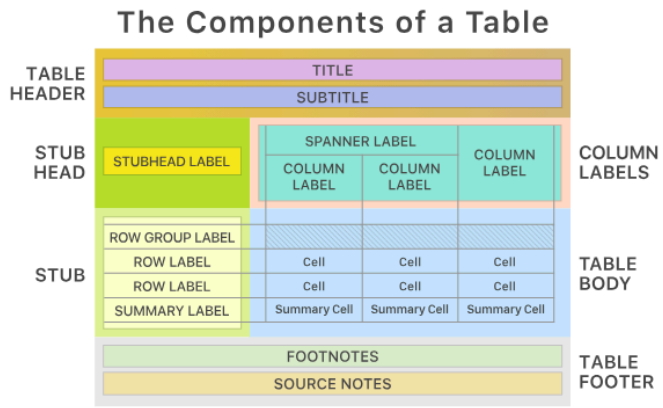
The Great Tables package is a Python package fully supported by Posit/RStudio PBC, where work started in 2022. Its first release to PyPI was in December 2023. The primary purpose of the package is to help Python users easily generate information-rich, publication-quality tables. Like the R package, gt (grammar of tables), Great Tables followed the "gt" naming convention and is also based on information from the Manual of Tabular Presentation by the Bureau of the Census. The document is focused on the theory and practice in the presentation of statistical data in tables for publication.
https://www2.census.gov/library/publications/1949/general/tabular-presentation.pdf

Like ggplot2 and gt, Great Tables describes a set of underlying components that can be recombined in different ways to solve different problems. This grammar helps developers to enforce good and sensible data-management practices when creating tables.

Pharmaceutical companies often have unique needs around TLGs such as various output formats like RTF, PDF or Word documents. Often unique formatting requirements also exist for items such as scientific notation, coloring, complex headers or multi-page formats. To help meet the evolving needs of pharmaceutical requirements, the Great Tables package also evolves to meet these specifications. If you discover that features are missing, please create an "issue" on github here: https://github.com/posit-dev/great-tables/issues

In R, there are various packages that build on top of gt. An example of that is the gtsummary package, which provides an elegant and flexible way to create publication-ready analytical and summary tables. In similar fashion, the recent PySummaries Python package makes it easily produce table summarizations from pandas dataframes.
https://github.com/Genentech/pysummaries

Great Tables is best understood by familiarizing yourself with the parts of a table as shown below:



The main parts are table header, the stub, the column labels and spanner column labels, the table body, and the table footer. Title and footnotes are integral parts of Tables, Figures or Listings (TFLs) in Clinical Study Reports (CSRs) and Case Report Tabulation (CRT). Titles and footnotes are created or typed in a requirement document such as Statistical Analyses Plan (SAP). Please refer to the E3 Structure and Content of Clinical Study Reports:
https://www.fda.gov/regulatory-information/search-fda-guidance-documents/e3-structure-and-content-clinical-study-reports
Great Tables accepts both Pandas and Polars DataFrames as the input data. The main entry point into the Great Tables API is the `GT()` class where the data is passed in. It is important to understand the data to be displayed and often there are data wrangling parts separate from Great Tables. After the input DataFrame is prepared, methods from Great Tables are used to format cells, add parts to the table (e.g., table header, a stub, and sometimes source notes in the table footer), modify the arrangement of the rows and columns, and format the data cells. The `save()` method allows users to save the table as a standalone image file; the `as_raw_html()` method returns the table as an HTML fragment. Furthermore, Great Tables can be integrated and rendered in a notebook environment or a Quarto document. Below is example usage of Great Tables where a DataFrame is prepared and then fashioned into a table for display:

```
import polars as pl
from great_tables import GT, md, html
from great_tables.data import islands

islands_mini = (
    pl.from_pandas(islands).sort("size", descending=True)
    .head(10)
)
(
    GT(islands_mini)
    .tab_header(
        title="Large Landmasses of the World",
        subtitle="The top ten largest are presented"
    )
    .tab_stub(rowname_col="name")
    .tab_source_note(source_note="Source: The World Almanac and Book of Facts, 1975,
page 406.")
    .tab_source_note(
        source_note=md("Reference: McNeil, D. R. (1977) *Interactive Data Analysis*.
Wiley.")
    )
    .tab_stubhead(label="landmass")
    .fmt_integer(columns="size")
)
```

This code will generate the following HTML table:

Large Landmasses of the World

The top ten largest are presented

landmass	size
Asia	16,988
Africa	11,506
North America	9,390
South America	6,795
Antarctica	5,500
Europe	3,745
Australia	2,968
Greenland	840
New Guinea	306
Borneo	280

Source: The World Almanac and Book of Facts, 1975, page 406.

Reference: McNeil, D. R. (1977) *Interactive Data Analysis*. Wiley.

You can learn more about the basics of Great Tables at the following examples:
https://posit-dev.github.io/great-tables/examples/

Simple Adverse Events Example

Below we will look at a simple Great Tables example using adverse events data. The full example and code are here:
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables/tree/main/Great_Tables_Openfda
The FDA maintains an API that houses a number of high-value, high priority and scalable structured datasets, including adverse events, drug product labeling, and recall enforcement reports. https://open.fda.gov/

This first part is creating a DataFrame (Pandas or Polars) to pass into Great Tables from openFDA data. Then we will bind the male and female DataFrames into one set, modify some cell values, reshape the table into a wide form, add a column, and take 10 rows from that table.

```
import requests
import pandas as pd
```

```
from great_tables import GT, md

def get_openfda_data(endpoint, query_params):
    base_url = "https://api.fda.gov"
    try:
        url = f"{base_url}/{endpoint}.json"
        response = requests.get(url, params=query_params)
        if response.status_code == 200:
            data = response.json()
            return data
        else:
            print(f"Error: {response.status_code}")
            return None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None

# Define endpoint and query parameters
endpoint = "drug/event"
query_params = {
    "search": "patient.drug.medicinalproduct:tylenol",
    "limit": 1000,  # Set limit to 1000 records
}
# Fetch data from OpenFDA
data = get_openfda_data(endpoint, query_params)
# Prepare the data for male and female
records = []
if data and "results" in data:
    for event in data["results"]:
        gender_code = event.get("patient", {}).get("patientsex", None)
        gender = "Male" if gender_code == "1" else "Female" if gender_code == "2" else
"Unknown"
        events = [reaction["reactionmeddrapt"] for reaction in event.get("patient",
{}).get("reaction", [])]

        for ev in events:
            records.append({
                "Adverse Event": ev,
                "Gender": gender
            })
else:
    print("No data retrieved or error in response structure.")
# Create DataFrame from records
df = pd.DataFrame(records)
# Pivot and count occurrences by gender
pivot_df = df.pivot_table(
    index='Adverse Event',
    columns='Gender',
    aggfunc='size',
    fill_value=0
).reset_index()
# Calculate a total count across all genders
pivot_df['Total'] = pivot_df.select_dtypes(include=['number']).sum(axis=1)

# Sort by the total count and select the top 10
top10_df = pivot_df.sort_values(by='Total', ascending=False).head(10)
```

Given that it's ready for Great Tables, we pass that into the GT() class and use methods from Great Tables to build a presentation-ready table:

```
# Create and configure the table using GreatTable
GT(top10_df).tab_header(
```

```
    title="Top 10 Adverse Events for Tylenol by Gender",
    subtitle="Most frequent adverse event reports for male and female patients
retrieved from OpenFDA"
).tab_stub(
    rowname_col="Adverse Event"
).tab_stubhead(
    label="Patient Reaction"
).tab_source_note(
    source_note=md("Data obtained from the OpenFDA database of 1000 data points.")
)
```

With the collection of Great Tables methods used above, we've created the following table:

Top 10 Adverse Events for Tylenol by Gender

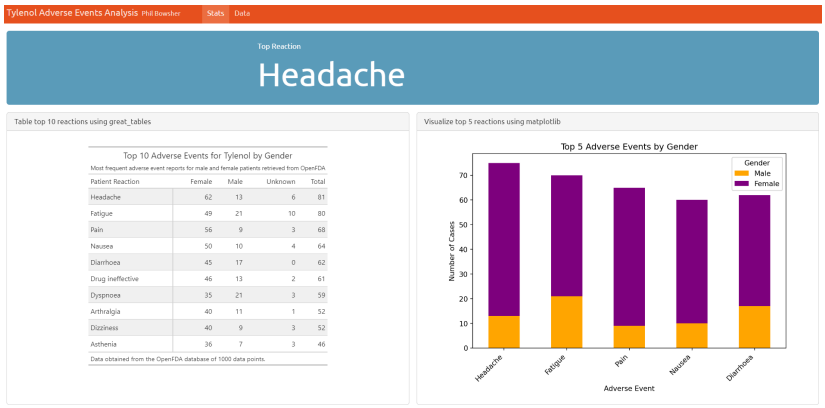
Most frequent adverse event reports for male and female patients retrieved from OpenFDA

Patient Reaction	Female	Male	Unknown	Total
Headache	62	13	6	81
Fatigue	49	21	10	80
Pain	56	9	3	68
Nausea	50	10	4	64
Diarrhoea	45	17	0	62
Drug ineffective	46	13	2	61
Dyspnoea	35	21	3	59
Arthralgia	40	11	1	52
Dizziness	40	9	3	52
Asthenia	36	7	3	46

Data obtained from the OpenFDA database of 1000 data points.

Please note that often Great Tables will be presented in a literate programming tool such as Quarto: https://quarto.org/

An example can be seen at the links below in the HTML based Quarto Dashboard:
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables/tree/main/Quarto_Dashboard_Great_Tables



CDISC TLG - 3 EXAMPLES

The CDISC pilot package is real clinical trial data, provided by Eli Lilly and Company, de-identified & documents redacted. CDISC pilot package description:

Case Study Title: Safety and Efficacy of the Xanomeline Transdermal Therapeutic System (TTS) in Patients with Mild to Moderate Alzheimer's Disease
Indication: Alzheimer's Randomized, double-blind, placebo-controlled, parallel-group study
Three treatment arms: low dose, high dose, placebo
Approximately 300 patients, multiple centers

We will use the CDISC pilot package to replicate 3 submission outputs to highlight the functionality of Great Tables for TLGs. All of the examples below are available here:
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables/tree/main/great-tables-and-tfls
The tables include, Table 14.2.01 "Summary of Demographic and Baseline Characteristics", Table 14.3.13 "CIBIC+ - Categorical Analysis - LOCF" and Table 14-5.01 Incidence of Treatment Emergent Adverse Events by Treatment Group. The examples below work with ADaM data. The ADam data used in the following examples were prepared using R. Similar data preparation can be done with Pharmaverse packages like sdtm.oak, Admiral and metacore which are beyond the scope of this paper. For more information about this topic, please refer to the R/Pharma 2024 workshops:

1. Building ADaMs with pharmaverse R packages admiral, metacore/metatools and xportr
2. SDTM programming in R using {sdtm.oak} package

These sessions can be viewed here: https://www.youtube.com/c/RinPharma

For the tables below, most of the data preparation stays out of Great Tables and is implemented viaPandas or Polars. Each table will include a small amount of data prep that are focused on the requirements needed by Great Tables such as pulling each of the *N* values for the required columns. The examples below will walk through the Great Tables sections. You can see the data preparation via the github link below.

Example 1 - Table 14.2.01 "Summary of Demographic and Baseline Characteristics"

For this first example, we will break apart each component that will generate the multi-page table below. Often Great Tables examples are combined. Below is broken out for learning purposes.
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables/blob/main/great-tables-and-tfls/gt-01-14_2_01.qmd

Table 14.2.01

Summary of Demographic and Baseline Characteristics

category	label	Placebo (N=86)	Xanomeline Low Dose (N=84)	Xanomeline High Dose (N=84)	Total (N=254)	p-value[1]
Age (y)	n	86.0	84.0	84.0	254	0.5934
	Mean	75.2	75.7	74.4	75.1	
	SD	8.59	8.29	7.89	8.25	
	Median	76.0	77.5	76.0	77.0	
	Min	52.0	51.0	56.0	51.0	
	Max	89.0	88.0	88.0	89.0	
	<65 yrs	14.0 (16%)	8.00 (10%)	11.0 (13%)	33.0 (13%)	0.1439
	65-80 yrs	42.0 (49%)	47.0 (56%)	55.0 (65%)	144 (57%)	
	>80 yrs	30.0 (35%)	29.0 (35%)	18.0 (21%)	77.0 (30%)	
Sex	n	86.0	84.0	84.0	254	0.1409
	Male	33.0 (38%)	34.0 (40%)	44.0 (52%)	111 (44%)	

Bottom:

Baseline height (cm)	n	86.0	84.0	84.0	254	0.1262
	Mean	163	163	166	164	
	SD	11.5	10.4	10.1	10.8	
	Median	163	163	165	163	
	Min	137	136	146	136	
	Max	185	196	190	196	
Baseline BMI	n	86.0	83.0	84.0	253	0.0133
	Mean	23.6	25.1	25.3	24.7	
	SD	3.67	4.27	4.16	4.09	
	Median	23.4	24.3	24.8	24.2	
	Min	15.1	17.7	13.7	13.7	
	Max	33.3	40.1	34.5	40.1	
	<25	59.0 (69%)	47.0 (56%)	44.0 (52%)	150 (59%)	0.2326
	25-<30	21.0 (24%)	27.0 (32%)	28.0 (33%)	76.0 (30%)	
	>=30	6.00 (7%)	10.0 (12%)	12.0 (14%)	28.0 (11%)	
Program Source: 14-2.01 Executed: (Draft) 2024-10-22						

Steps:

1. Introduce the `tbl` DataFrame to the `GT()` class and print the table.

```
```{python}
gt_table = GT(tbl)
gt_table
```
```

2. We will continue to build the Great Tables with methods. Next, we will add a table header. This contains a title (`"Table 14.2.01"`) and a subtitle (`"Summary of Demographic and Baseline Characteristics"`).

```
```{python}
gt_table = (
  gt_table
  .tab_header(
    title="Table 14.2.01",
    subtitle="Summary of Demographic and Baseline Characteristics"
  )
)
gt_table
```
```

3. Let's remove all of those cells that have `None` in them. We can do this with the `sub_missing()` method. Ensure that the replacement text is an empty string (`""`).

```
```{python}
gt_table = gt_table.sub_missing(missing_text="")
gt_table
```
```

4. We don't need the percentage columns (all end in `_pct`) so let's hide them with `cols_hide()`.

```
```{python}
gt_table = gt_table.cols_hide(columns=cs.ends_with("_pct"))
gt_table
```
```

5. We need column labels that work better than the short ones commonly used for data analysis. With `cols_label()` we can replace the default labels (i.e., column names are used as the default labels). We wrap the label strings with `md()` to enable Markdown formatting. To add a line break, use the following bit of text `" \n"`. Note that we are adding in data from variables we declared earlier.

```
```{python}
gt_table = (
  gt_table
  .cols_label(
```
```

```
    placebo=md(f"Placebo  \n(N={placebo_n})"),
    xanomeline_ld=md(f"Xanomeline  \nLow Dose  \n(N={xanomeline_ld_n})"),
    xanomeline_hd=md(f"Xanomeline  \nHigh Dose  \n(N={xanomeline_hd_n})"),
    total=md(f"Total  \n(N={total_n})"),
    p="p-value"
  )
)
gt_table
```

6. Let's add a source note. This can be done with the `tab_source_note()` method.

```
```{python}
gt_table = (
  gt_table
  .tab_source_note(
    source_note=f"Program Source: 14-2.01 Executed: (Draft)
{datetime.now().date().isoformat()}"
  )
)
gt_table
```
```

7. Putting it all together, you can write the table as one block of code.

```
```{python}
gt_table_final = (
  GT(tbl)
  .tab_header(
    title="Table 14.2.01",
    subtitle="Summary of Demographic and Baseline Characteristics"
  )
  .sub_missing(missing_text="")
  .cols_hide(columns=cs.ends_with("_pct"))
  .cols_label(
    placebo=md(f"Placebo  \n(N={placebo_n})"),
    xanomeline_ld=md(f"Xanomeline  \nLow Dose  \n(N={xanomeline_ld_n})"),
    xanomeline_hd=md(f"Xanomeline  \nHigh Dose  \n(N={xanomeline_hd_n})"),
    total=md(f"Total  \n(N={total_n})"),
    p="p-value[1]"
  )
  .tab_source_note(
    source_note=f"Program Source: 14-2.01 Executed: (Draft)
{datetime.now().date().isoformat()}"
  )
)
gt_table_final
gt_table_final.save("image.png")
# If you wanted the HTML for the table, you can use the .as_raw_html() method. It
doesn't write an HTML page to a file though, it returns a string with the HTML text
for a table element (wrapped in a div with a style element). You can try using
gt_table_final.as_raw_html() to get the HTML for the table. It doesn't write HTML to a
file though, it returns that table HTML as a string.
```
```

Example 2 - Table 14.3.13 "CIBIC+ - Categorical Analysis - LOCF"

The code for the second example is here:
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables/blob/main/great-tables-and-tfls/gt-02-14_3_13.qmd
You can view each step explained in the code above. As this example is similar to the first, we will only show the code in one code block. The second table will look like:

Table 14.3.13

CIBIC+ - Categorical Analysis - LOCF

	Assessment	Placebo (N=79)	Xanomeline Low Dose (N=81)	Xanomeline High Dose (N=74)	p-value
Week 8	n	77	81	73	0.2727
	Marked improvement	0	0	0	
	Moderate improvement	1 (1%)	2 (2%)	1 (1%)	
	Minimal improvement	19 (25%)	16 (20%)	13 (18%)	
	No Change	45 (58%)	48 (59%)	38 (52%)	
	Minimal worsening	10 (13%)	14 (17%)	20 (27%)	
	Moderate worsening	2 (3%)	1 (1%)	1 (1%)	
	Marked worsening	0	0	0	

Bottom:

Week 24	n	79	81	74	0.6183
	Marked improvement	0	0	0	
	Moderate improvement	1 (1%)	1 (1%)	0	
	Minimal improvement	9 (11%)	14 (17%)	11 (15%)	
	No Change	38 (48%)	37 (46%)	33 (45%)	
	Minimal worsening	28 (35%)	27 (33%)	25 (34%)	
	Moderate worsening	3 (4%)	2 (2%)	5 (7%)	
	Marked worsening	0	0	0	

Overall comparison of treatments using CMH test (Pearson Chi-Square), controlling for site group.

2024-10-22

Putting it all together, you can write the table as one block of code.

```
```{python}
gt_table_final = (
  GT(tbl)
  .tab_header(
    title="Table 14.3.13",
    subtitle="CIBIC+ - Categorical Analysis - LOCF"
  )
  .sub_missing(missing_text="")
  .cols_hide(columns=cs.ends_with("_pct"))
  .cols_label(
    category="",
    label="Assessment",
    placebo=md(f"Placebo  \n(N={placebo_n})"),
    xanomeline_ld=md(f"Xanomeline  \nLow Dose  \n(N={xanomeline_ld_n})"),
    xanomeline_hd=md(f"Xanomeline  \nHigh Dose  \n(N={xanomeline_hd_n})"),
    p="p-value"
  )
  .tab_source_note(
    source_note="Overall comparison of treatments using CMH test (Pearson Chi-Square),
controlling for site group."
  )
  .tab_source_note(
    source_note=f"{datetime.now().date().isoformat()}"
  )
)
gt_table_final
gt_table_final.save("image.png")
```
```

```
# If you wanted the HTML for the table, you can use the .as_raw_html() method. It doesn't write an HTML page to a file though, it returns a string with the HTML text for a table element (wrapped in a div with a style element). You can try using gt_table_final.as_raw_html() to get the HTML for the table. It doesn't write HTML to a file though, it returns that table HTML as a string.
```

Example 3 - Table 14-5.01 Incidence of Treatment Emergent Adverse Events by Treatment Group

The code for the third example is here:
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables/blob/main/great-tables-and-tfls/gt-03-14_5_01.qmd
The third table adds some complexity. This table is built up using many of the same methods we used in the two examples above so we do not explain each part as was done in the first table. There is a notable difference though, the `tab_spanner()` method is used to create spanner column labels (these are labels over top the column labels)
The third table will look like this:

Table 14-5.01

Incidence of Treatment Emergent Adverse Events by Treatment Group

	Placebo (N=86)		Xanomeline Low Dose (N=84)		Xanomeline High Dose (N=84)		Fisher's Exact p-values	
Label	n(%)	[AEs]	n(%)	[AEs]	n(%)	[AEs]	Placebo vs. Low Dose	Placebo vs. High Dose
ANY BODY SYSTEM	65 (76%)	[281]	77 (92%)	[412]	76 (90%)	[433]	0.007	0.014
CARDIAC DISORDERS	12 (14%)	[26]	13 (16%)	[30]	15 (18%)	[30]	0.831	0.534
SINUS BRADYCARDIA	2 (2%)	[2]	7 (8%)	[10]	8 (10%)	[12]	0.097	0.056
MYOCARDIAL INFARCTION	4 (5%)	[4]	2 (2%)	[4]	4 (5%)	[8]	0.682	0.999
ATRIAL FIBRILLATION	1 (1%)	[1]	1 (1%)	[1]	3 (4%)	[5]	0.999	0.365
ATRIAL FLUTTER	0		1 (1%)	[1]	1 (1%)	[2]	0.494	0.494
CARDIAC DISORDER	0		0		1 (1%)	[1]		0.494
SUPRAVENTRICULAR EXTRASYSTOLES	1 (1%)	[2]	1 (1%)	[2]	1 (1%)	[1]	0.999	0.999
VENTRICULAR EXTRASYSTOLES	0		2 (2%)	[4]	1 (1%)	[1]	0.243	0.494
ATRIAL HYPERTROPHY	1 (1%)	[2]	0		0		0.999	0.999

Bottom:

SOCIAL CIRCUMSTANCES	0		0		1 (1%)	[1]		0.494
ALCOHOL USE	0		0		1 (1%)	[1]		0.494
SURGICAL AND MEDICAL PROCEDURES	2	[2]	1 (1%)	[1]	2 (2%)	[2]	0.999	0.999
ACROCHORDON EXCISION	0		0		1 (1%)	[1]		0.494
SKIN LESION EXCISION	0		0		1 (1%)	[1]		0.494
CATARACT OPERATION	1 (1%)	[1]	1 (1%)	[1]	0		0.999	0.999
EYE LASER SURGERY	1 (1%)	[1]	0		0		0.999	0.999
VASCULAR DISORDERS	3 (4%)	[7]	3 (4%)	[3]	1 (1%)	[1]	0.999	0.621
WOUND HAEMORRHAGE	0		0		1 (1%)	[1]		0.494
HOT FLUSH	0		1 (1%)	[1]	0		0.494	
HYPERTENSION	1 (1%)	[2]	1 (1%)	[1]	0		0.999	0.999
HYPOTENSION	2 (2%)	[3]	1 (1%)	[1]	0		0.999	0.497
ORTHOSTATIC HYPOTENSION	1 (1%)	[2]	0		0		0.999	0.999

Note: Treatment emergent events are defined as events which start on or after the start of treatment.
Note: Adverse events are coded using MedDRA.
Note: Percentages are based on the number of subjects in the safety population within each treatment group.
Note: P-values are based on Fisher's Exact test for the comparison of placebo versus each active treatment group. An asterisk is appended to p-values that are less than 0.15.
Note: The column [AE] represents the total number of times an event was recorded.
Program Source: t-14-5-01.R Executed: (Draft), 2024-10-22

The code is below:
Putting it all together, you can write the table as one block of code.

```
```{python}
gt_table_final = (
  GT(tbl)
  .tab_header(
```
```

```
    title="Table 14-5.01",
    subtitle="Incidence of Treatment Emergent Adverse Events by Treatment Group",
  )
  .sub_missing(missing_text="")
  .cols_hide(columns=cs.ends_with("_pct"))
  .cols_label(
    label="Label",
    placebo_n="n(%)",
    xanomeline_ld_n="n(%)",
    xanomeline_hd_n="n(%)",
    placebo_ae="[AEs]",
    xanomeline_ld_ae="[AEs]",
    xanomeline_hd_ae="[AEs]",
    fisher_placebo_v_xanomeline_ld=md("Placebo  \nvs.  \nLow Dose"),
    fisher_placebo_v_xanomeline_hd=md("Placebo  \nvs.  \nHigh Dose")
  )
  .tab_spanner(
    label=md(f"Placebo  \n(N={placebo_n})"),
    columns=cs.starts_with("placebo")
  )
  .tab_spanner(
    label=md(f"Xanomeline  \nLow Dose  \n(N={xanomeline_ld_n})"),
    columns=cs.starts_with("xanomeline_ld")
  )
  .tab_spanner(
    label=md(f"Xanomeline  \nHigh Dose  \n(N={xanomeline_hd_n})"),
    columns=cs.starts_with("xanomeline_hd")
  )
  .tab_spanner(
    label=md("Fisher's Exact  \np-values"),
    columns=cs.starts_with("fisher")
  )
  .tab_source_note(source_note = "Note: Treatment emergent events are defined as
events which start on or after the start of treatment.")
  .tab_source_note(source_note = "Note: Adverse events are coded using MedDRA.")
  .tab_source_note(source_note = "Note: Percentages are based on the number of
subjects in the safety population within each treatment group.")
  .tab_source_note(source_note = "Note: P-values are based on Fisher's Exact test for
the comparison of placebo versus each active treatment group. An asterisk is appended
to p-values that are less than 0.15.")
  .tab_source_note(source_note = "Note: The column [AE] represents the total number of
times an event was recorded.")
  .tab_source_note(source_note = f"Program Source: t-14-5-01.R     Executed: (Draft),
{datetime.now().date().isoformat()}")
)
gt_table_final
# gt_table_final.save("image.png")
# If you wanted the HTML for the table, you can use the .as_raw_html() method. It
doesn't write an HTML page to a file though, it returns a string with the HTML text
for a table element (wrapped in a div with a style element). You can try using
gt_table_final.as_raw_html() to get the HTML for the table. It doesn't write HTML to a
file though, it returns that table HTML as a string.
```

Future Considerations - interactive TLGs, ARDs & Shiny for Python

Many pharmaceutical companies are interested in using interactivity and or Shiny for clinical trials, either for internal analysis, generating TLGs or for inclusion within a submission to the regulatory agency. The R Submissions Working Group is currently piloting to the FDA the R Submission Pilot 4 which is utilizing alternative methods of distributing a self-contained submission bundle. Part of the pilot is using WebAssembly. In R, WebR brings R compiled for the browser and Node.js using WebAssembly, via Emscripten. Shinylive utilizes these tools to enable running Shiny entirely in the browser, without any need for a hosted server. For Python, when using Shinylive, Python and Shiny run in the web browser and the browser becomes both the client and server. These tools can use Python packages like

Great Tables discussed in this paper. You can read more about this architecture here:
https://shiny.posit.co/py/docs/shinylive.html
A popular package in R for making TLGs is gtsummary. Agustin Calatroni recently replicated the CDISC Pilot replication with gtsummary: https://agstn.github.io/CPR_gtsummary/CPR_gtsummary.html

gtsummary recently added new functions `tbl_ard_summary()` and `tbl_ard_continuous()` for creating bespoke summary tables that accept an ARD object (Analysis Results Dataset often created with the cards or cardx packages). You can learn more about this here:
https://www.danieldsjoberg.com/china-pharma-keynote-2024/slides/#/title-slide
And also at the R/Pharma recording of the workshop "Unlocking Analysis Results Datasets: A Practical Workshop for Creating and Utilizing ARDs for Clinical Reporting" here: https://www.youtube.com/c/RinPharma

It will be interesting to see if PySummaries or other Python packages implement the ARD capabilities.

Great Table works well with Shiny for Python and Quarto, where organizations can add dynamic and interactive components. Roche built a framework called teal that leverages the R Shiny package to scale development of our shiny apps. You can learn more about teal below:
https://posit.co/blog/roche-shifting-to-an-open-source-backbone-in-clinical-trials/
https://pharmaverse.org/e2eclinical/tlg/
https://insightsengineering.github.io/teal/latest-tag/
It will be interesting to see if teal is extended to Python as well. While there are many open-source Python libraries to make custom web apps, a new option is Shiny for Python: https://shiny.posit.co/py/

You can see an example of a Shiny for Python app with Great Tables here:
https://github.com/philbowsher/Clinical-Tables-in-Python-with-Great-Tables/tree/main/great_tables_Shiny_python
You can see the main Great Tables section in the App Server logic like this:

```
table = (
                table
                .cols_label(year="Year")
                .fmt_number(
                    columns=list(input.countries()),
                    use_seps=True,
                    decimals=0
                )
                .tab_spanner(
                    label="Population",
                    columns=list(input.countries())
                )
                .tab_header(
                    title="Country Comparison",
                    subtitle=f"Data from {input.start_year()} to {input.end_year()}"
                )
                .cols_width(
                    {"year": "100px", **{col: "130px" for col in input.countries()}}
                )
                .cols_align(align="center", columns="year")
                .tab_source_note(source_note=source_note)
            )
        return table
```

Conclusion & Contact

The information above highlights the exciting Great Tables Python package and example use cases, and how well established tools like these can help modernize clinical processes for reporting via reports, dashboards and Shiny for Python applications.

Phil Bowsher

phil@posit.co

https://github.com/philbowsher