PharmaSUG 2023 - Paper DV-134

Life Table Analysis for Time to First Event Onset

Abhinav Srivastva, Exelixis Inc., Alameda, CA

ABSTRACT

Life Table Analysis is a useful way to study subject's survival with respect to an event of interest over a time period. It provides a good indicator of drug safety or toxicity over the course of a clinical trial due to the occurrence of a related event. For example, life table can be used to study the relation between a drug which is highly immunogenic in nature and the type of events it can trigger over a time period, such as increased liver events indicating signs of liver disease. In this paper we take a graphical approach to represent this information which is then enhanced to add exploratory and interactive features for the reviewer. Data preprocessing is done in SAS®, while all the plots are created in Python using open-source libraries such as matplotlib, seaborn, plotly and dash.

INTRODUCTION

Life table is a quick and information tool to understand the relation between a drug and associated events over a period of time as it relates to subject's survivability. The paper provides detailed instructions on data preprocessing steps before creating estimates such as proportion and 95% confidence intervals (CI). Confidence intervals are created using Clopper-Pearson exact method with FREQ procedure. Plots are created using **matplotlib** and **seaborn**. An interactive plot is also demonstrated using **plotly** and **dash** which lets an end user get additional insights about the data while hovering over a data point.

DATA PRE-PROCESSING

Consider an event "INCREASED ALT LEVEL" that we are tracking over the course of the clinical trial, and our goal is to calculate proportions and 95% CI of this event onset at specified time intervals. The events can be typically found in Adverse Events analysis dataset (ADAE) as below:

USUBJID	AETERM	AEDECOD	AETOXGR	ASTDTC	AENDTC	ASTDY
ABC-123	INCREASED ALT LEVEL	ALT Increased	3	2022-03-01	2022-03-15	60

Table 1: ALT Onset at Day 60

We can note that the onset of this adverse event is on Day 60. The subject should be considered at risk at all timepoints on/prior to Day 60, but after the event has occurred the subject is no longer at risk at subsequent time points. For a better visualization, the time-points can be divided into intervals such as Week 1-5, 6-10, 11-15, and so on. This decision to divide the intervals can be based on Protocol's schedule of assessments or medically motivated based on drug's profile.

In our example, we will divide the intervals as 5 weeks apart for every subject in the safety population. When an event is met, subject will get 1 = EVENT and 2 = NO EVENT for the time-interval where the onset falls. For subject ABC-123, event onset of Day 60 corresponds to 60 / 7 = 8.6 week or interval "Week 5-10" as demonstrated below (Table 2)

USUBJID	ONSET DAY	ONSET WEEK	INTERVAL	EVENT (1 = Event, 2 = No Event)
ABC-123	60	8.6	Week 1 – 5	2
ABC-123	60	8.6	Week 6 – 10	1
ABC-123	60	8.6	Week 11 – 15	2
ABC-123	60	8.6	Week 16+	2

Table 2: Subject Event Intervals

For a life-table, when a subject has encountered an event, it should no longer be considered at risk for the subsequent time intervals, hence the revised layout is as shown below.

USUBJID	ONSET DAY	ONSET WEEK	INTERVAL	EVENT (1 = Event, 2 = No Event)
ABC-123	60	8.6	Week 1 – 5	2
ABC-123	60	8.6	Week 6 – 10	1

Table 3: Revised Subject Event Intervals

We will create this structure for every subject in the population with the consideration that when an event has occurred then it should not be considered at risk for subsequent time intervals. Another consideration is for cases where a subject met other critical events such as Death, treatment discontinuation, withdrawn consent which would disqualify as being at risk. This subject-level information can often be found in ADSL dataset. Let's reinforce this with a simple example where subject *never* encountered INCREASED ALT LEVEL event that we are tracking, but had a death reported on Day 91 (Week 11-15), leading it to being discarded for subsequent interval i.e. Week 16+.

ADSL:

USUBJID	DTHDTC	DTHDY
ABC-456	2022-04-01	91

Table 4: Subject with a reported Death

USUBJID	ONSET DAY	ONSET WEEK	INTERVAL	EVENT (1 = Event, 2 = No Event)
ABC-456	91	13	Week 1 – 5	2
ABC-456	91	13	Week 6 – 10	2
ABC-456	91	13	Week 11 – 15	2

Table 5: Subject with No ALT Event, but accounting for Death

CALCULATING ESTIMATES

The proportions in life table at each time interval can be calculated as:

$$\text{Proportion} = \frac{\#\ Subjects\ with\ events}{\#\ Subjects\ at\ Risk}$$

and 95% Confidence interval is calculated using Clopper-Pearson Exact Method [1]. Once the data is prepared, we can calculate these metrics using FREQ procedure as below:

```
ods output binomial = stats; * stats will contain estimates;
proc freq data = prepped_data ;
     by treatment_id risk_week;
        tables event / binomial;
        exact binomial;
run;
```

where,

prepped_data = input dataset
treatment_id = 0 (Active) or 1 (Placebo)
risk_week = Intervals 'W 1-5', 'W 6-10', 'W 11-15', 'W 16+'
event = 1 (Event) or 2 (No event)

Caution should be taken if any interval has '0' events (i.e. no record with EVENT=1). In that case, proportions and CI will be misleading as they get computed from non-event perspective. A workaround is to create a dummy record with EVENT=1 in the missing category, and assign a weight = 0, keeping the weights for the rest as "1".

INTERVAL	EVENT	COUNTS	WGT	Notes
Week 1-5	1	0	0	This is a dummy record created with EVENT=1 and WEIGHT set to 0
Week 1-5	2	xx	1	

Table 6: A case of missing events in a given Interval (eg. Week 1-5)

Updated FREQ procedure will be:

```
ods output binomial = stats;
proc freq data = prepped_data ;
     by treatment_id risk_week;
        tables event / binomial;
        weight wgt / zeros; * to account for missing events;
        exact binomial;
run;
```

Here is a summary table created out of fictitious data that will be used for plotting in the next section.

Interval	Interval_Desc	Treatment	Treatment_Desc	At_Risk	Events	Prop (%)	LCL (%)	UCL (%)
1	Week 1 - 5	0	Active	100	4	4.0	1.1	9.9
1	Week 1 - 5	1	Placebo	100	2	2.0	0.2	7.0
2	Week 6 - 10	0	Active	96	5	5.2	1.7	11.7
2	Week 6 - 10	1	Placebo	98	3	3.1	0.6	8.7
3	Week 11 - 15	0	Active	91	8	8.8	3.9	16.6
3	Week 11 - 15	1	Placebo	95	9	9.5	4.4	17.2
4	Week 16 +	0	Active	83	11	13.3	6.8	22.5
4	Week 16 +	1	Placebo	86	15	17.4	10.1	27.1

Table 7: Summary Table

LIFE TABLE PLOT

For plotting Table 7 we will convert this into a Pandas DataFrame named "**`alt_df`**" (using Python) as below to be used for plotting with matplotlib and seaborn as covered in this section.

	Event_query	Interval	Interval_desc	Treatment	Treatment_desc	At_risk	Events	Prop (%)	LCL (%)	UCL (%)
0	ALT Grade >=3	1	Week 1 - 5	0	Active	100	4	4.000000	1.1	9.9
1	ALT Grade >=3	1	Week 1 - 5	1	Placebo	100	2	2.000000	0.2	7.0
2	ALT Grade >=3	2	Week 6 - 10	0	Active	96	5	5.208333	1.7	11.7
3	ALT Grade >=3	2	Week 6 - 10	1	Placebo	98	3	3.061224	0.6	8.7
4	ALT Grade >=3	3	Week 11 - 15	0	Active	91	8	8.791209	3.9	16.6
5	ALT Grade >=3	3	Week 11 - 15	1	Placebo	95	9	9.473684	4.4	17.2
6	ALT Grade >=3	4	Week 16 +	0	Active	83	11	13.253012	6.8	22.5
7	ALT Grade >=3	4	Week 16 +	1	Placebo	86	15	17.441860	10.1	27.1

Let's separate this dataframe by treatment which will come handy for plotting treatment sequence individually with matplotlib.

```
# create sequences by Treatment into separate dataframes

active = alt_df[alt_df['Treatment'] == 0]
pbo = alt_df[alt_df['Treatment'] == 1]
```

The plot is made-up of 2 parts – Line plot showing proportion and 95% CI, and a supporting table at the bottom showing subjects at risk and with events. The final plot is displayed below.

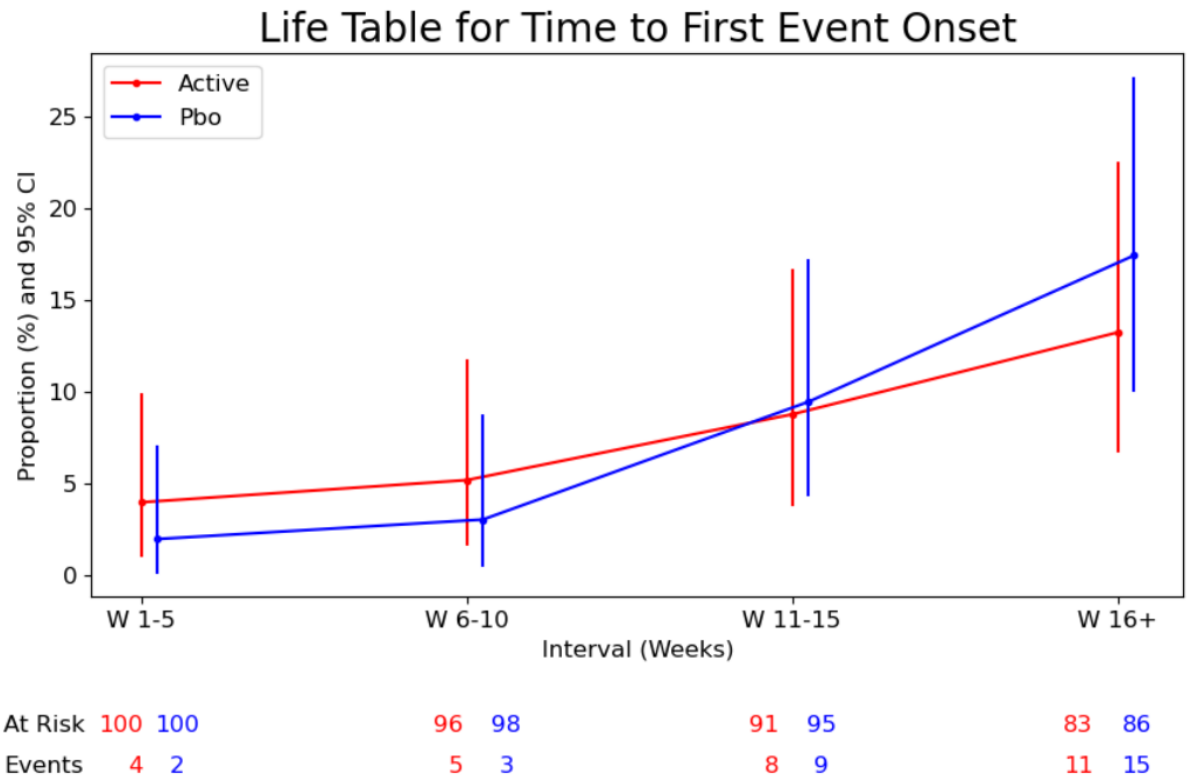

Figure 1: Life Table Plot

Let's set-up some parameters which will be used for plotting.

```
# line plot parameters

xlabels = ['W 1-5', 'W 6-10', 'W 11-15', 'W 16+']
offset  = 0.05 # show an offset for Placebo so it doesn't overlap with Active
```

Number of subjects at risk and with events from the dataframe has to be transposed into a specific format for using as a bottom table in the plot as shown below.

```
# Bottom table parameters and structure

rowLabel = ['At Risk', 'Events']

table_text = np.concatenate((alt_df.T.loc['At_risk'].values[np.newaxis],
                             alt_df.T.loc['Events'].values[np.newaxis]),
                             axis=0)
print(table_text)
```

```
[Output]>[[100 100  96  98  91  95  83  86]
          [4 2 5 3 8 9 11 15]]
```

The line plot can be done with matplotlib's **plot** method, and the table below the plot can be plotted with **table** method. Below is the complete code for plotting Figure 1, and notes are added to provide more details.

```
import matplotlib.pyplot as plt # import library

### PART 1: Plot proportion and 95% CI using plot() method
plt.figure(figsize = (10, 5))

plt.plot(active['Interval'], active['Prop (%)'], 'r.-', label = 'Active')
plt.plot([active['Interval'], active['Interval']],
         [active['LCL (%)'], active['UCL (%)']], 'r')
plt.plot(pbo['Interval']+offset, pbo['Prop (%)'], 'b.-', label='Pbo')
plt.plot([pbo['Interval']+offset, pbo['Interval']+offset],
         [pbo['LCL (%)'], pbo['UCL (%)']], 'b')

# Format objects such as title, axis and legend
plt.title("Life Table for Time to First Event Onset", fontsize=20)
plt.xticks([1, 2, 3, 4], xlabels, fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel("Interval (Weeks)", fontsize=12)
plt.ylabel('Proportion of Subjects With Event Onset (%)', rotation=90,
fontsize=12)
plt.legend(fontsize=12);

### PART 2: Plotting Bottom Table
mp_table = plt.table(cellText=table_text,
                     colWidths=[0.04, 0.04, 0.2, 0.02, 0.2, 0.02, 0.2, 0.02],
                     rowLabels=rowLabel,
                     loc='bottom',
                     bbox=[0, -0.35, 0.97, 0.15]
                     )

# Table formatting options, such as fontsize, cell text color and alignment
mp_table.auto_set_font_size(False)
mp_table.set_fontsize(12)

for k, cell in mp_table._cells.items():
    cell.set_edgecolor('w')
    if k[1] < 0:
        cell.set_text_props(color='k', fontsize=12)
    elif k[1]%2 ==0:
        cell.set_text_props(color='r', fontsize=12, ha='right') # red text
    elif k[1]%2 !=0:
        cell.set_text_props(color='b', fontsize=12, ha='center') # blue text
```

Another visualization that can give more insight about the data is the distribution of events along the timepoints. Here we use **seaborn** library to create a strip plot to show this distribution which has a basic syntax as:

```
stripplot (x = , y = , data = , options = <>)
```

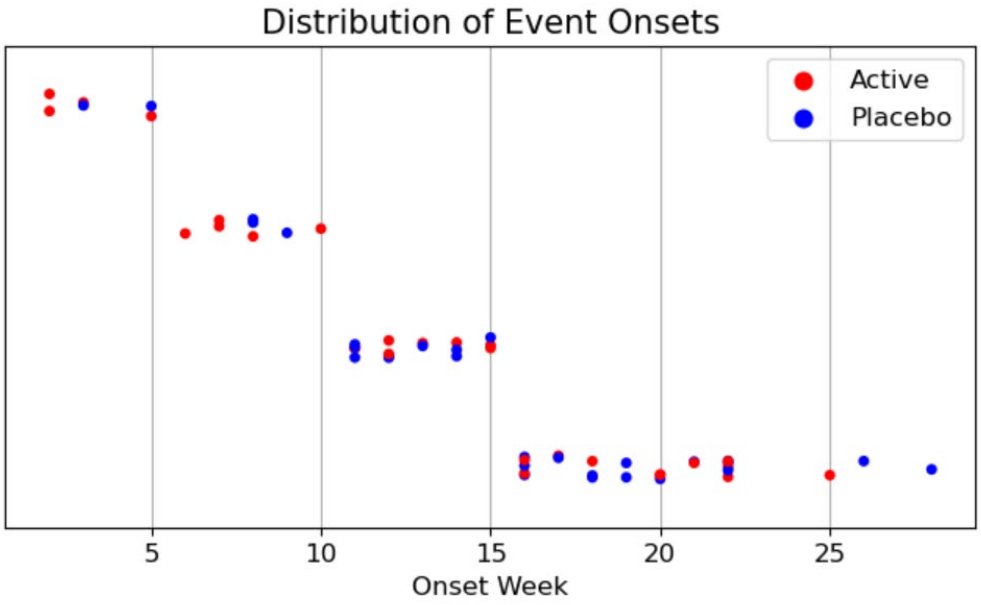

Figure 2: Strip plot of event distribution

The complete code to create Figure 2 is shown below:

```
import seaborn as sns # import seaborn library

category_order = ['Week 1 - 5', 'Week 6 - 10', 'Week 11 - 15', 'Week 16 +']
plt.figure(figsize = (8,4))

# Strip plot. "alt_details" is a another dataframe which has detailed by-
  week data
g = sns.stripplot(x='Onset_week', y='Interval', data = alt_details,
                  hue='Treatment_desc', palette= ['red', 'blue'],
                  order = category_order)

# Formatting title, axis and legend

plt.title("Distribution of Event Onsets", fontsize=15)
plt.xticks(fontsize=12)
plt.yticks([])
plt.xlabel("Onset Week", fontsize=12)
plt.ylabel(" ")
plt.legend(fontsize=12)
plt.grid(axis='x');
```

USER INTERACTION USING PLOTLY AND DASH

We can show strip plot (Figure 2) on-demand at the tooltip of the data points of Figure 1: Life Table Plot, using interactive plot libraries - **plotly** and **dash**.

First, Line plot can be created with `scatter()` function in **plotly**. 95% CI can be displayed by calculating high and low error values using precomputed Mean and 95% CI limits (UCL, LCL) as shown below.

```
# Active Arm Errors: Upper Error = (UCL-Mean), Lower Error = (Mean-LCL)

a_err_ucl = active['UCL (%)'] - active['Prop (%)']
a_err_lcl = active['Prop (%)'] - active['LCL (%)']
```

```
# Placebo Arm Errors: Upper Error = (UCL-Mean), Lower Error = (Mean - LCL)
p_err_ucl = pbo['UCL (%)'] - pbo['Prop (%)']
p_err_lcl = pbo['Prop (%)'] - pbo['LCL (%)']
```

Basic syntax of `scatter()` is:

```
scatter(x = , y = ,
        error_y=dict(
                     type = 'data',
                     symmetric = False,
                     array = ,           # show high error values
                     arrayminus = )      # show low error values
        )
```

Complete code for creating Line plot using **plotly** is shown below:

```
import plotly.graph_objects as go
fig = go.Figure()

# show proportion and 95% CI for Active arm
trace1 = go.Scatter(x= active['Interval'], y= active['Prop (%)'],
                    marker = dict(color='red'),
                    error_y=dict(
                        type='data',
                        symmetric=False,
                        array=a_err_ucl,
                        arrayminus=a_err_lcl),
                    name = 'Active'
        )
# show proportion and 95% CI for Placebo arm
trace2 = go.Scatter(x= pbo['Interval']+offset, y= pbo['Prop (%)'],
                    marker = dict(color='blue'),
                    error_y=dict(
                        type='data',
                        symmetric=False,
                        array=p_err_ucl,
                        arrayminus=p_err_lcl),
                    name = 'Placebo'
        )
fig.add_trace(trace1)
fig.add_trace(trace2)

# Format other objects such as title, axis and legend
fig.update_layout(
    title = dict(
        text="Life Table for Time to First Event Onset",
        x = 0.5,
        font = dict(size=20)
    ),
    yaxis_title="Proportion of Subjects With Event Onset (%)",
    legend_title="Treatment",
    xaxis = dict(
        title = "Interval (Weeks)",
        tickvals = [1, 2, 3, 4],
        ticktext = xlabels
    )
)
```

```
# Remove all hover content as it will be updated with dash later
fig.update_traces(
    hoverinfo="none",
    hovertemplate=None)
```

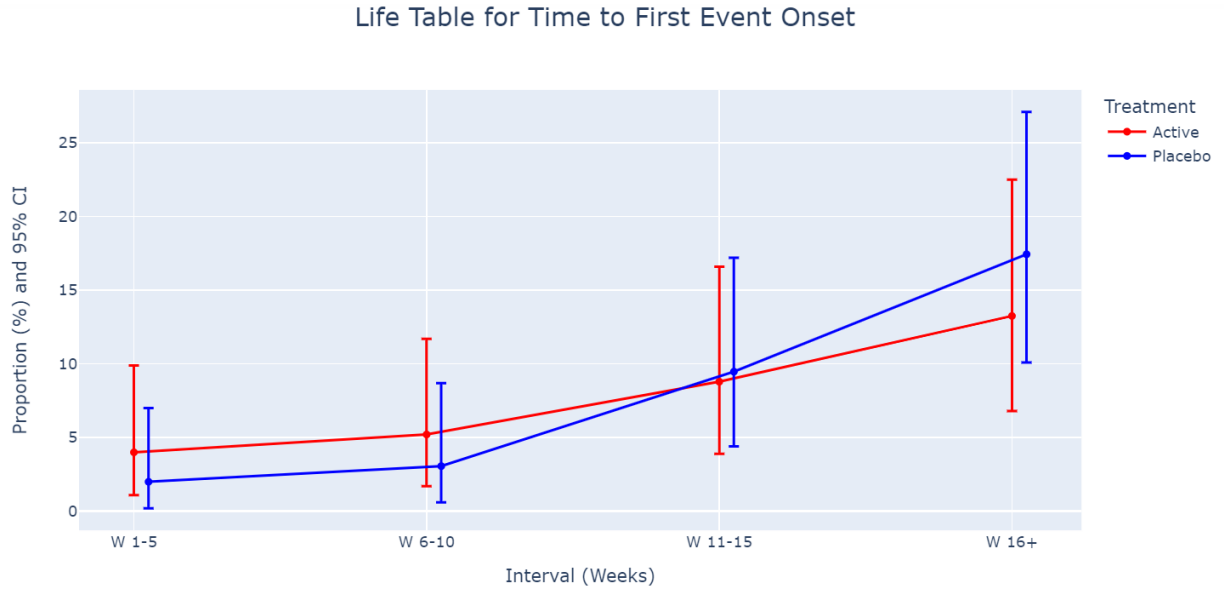

Figure 3: Line Plot using Plotly

Next, we can show a strip plot (Figure 2: Strip plot of event distribution) at the tooltip of the above plot when a user hovers over a data point. For example, as the user hovers over "Week 16+" category data point for Placebo arm, the underlying data will get filtered to include only Placebo "Week 16+" records, and a strip plot will be shown for that particular interval (Figure 4). This interactivity can be built using **dash** library.

In Figure 2, we used seaborn's `stripplot()` function, but here we will use plotly's `strip()` function to re-create the same plot. The basic syntax is shown below:

```
import plotly.express as px

px.strip(data = , x = , y = , <options> )
```

The code to build the complete dashboard with tooltip user-interaction is provided in the APPENDIX section of the paper for reference.

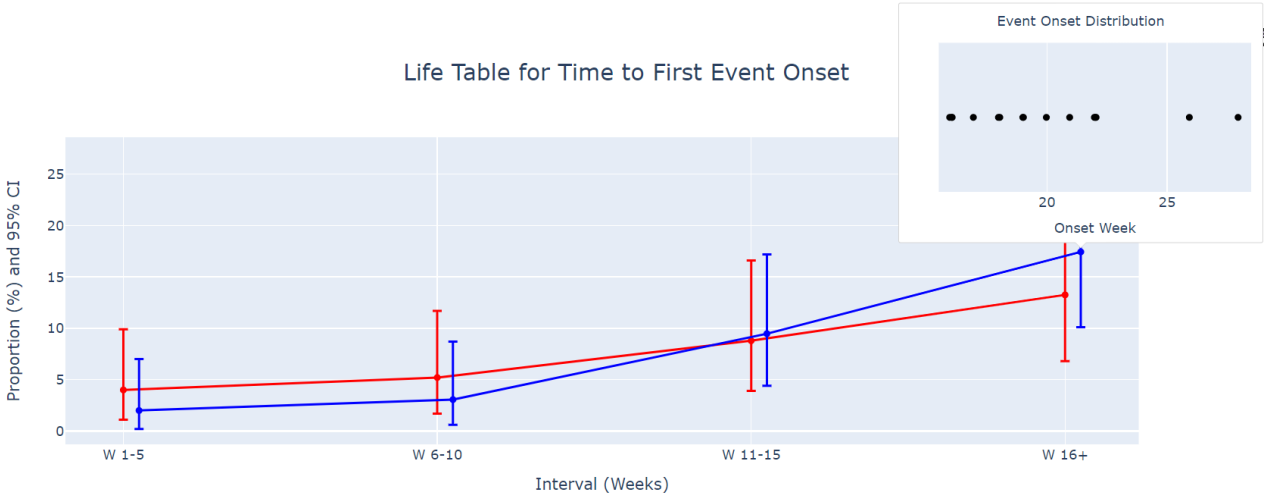

Figure 4: Line plot with event distribution (strip plot) at the Tooltip

CONCLUSION

Life table provides a useful representation for quantifying subjects who encountered an event of interest during the course of a clinical trial. We learned about data pre-processing steps before computing metrics such as proportions and 95% CI. Lastly, we explored Python's open-source libraries which can be used for visualizing results in only a few lines of code.

REFERENCES

[1] SAS Documentation on Binomial Proportions:
https://documentation.sas.com/doc/en/statcdc/14.2/statug/statug_freq_details37.htm

ACKNOWLEDGMENTS

I would like to acknowledge Yu-Lin Chang (Director, Biostatistics at Exelixis) for guidance on Life table analysis.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Abhinav Srivastva
Exelixis Inc
asrivastva@exelixis.com

TRADEMARK CITATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

APPENDIX

```
# Create a copy of the dataframe "alt_details" which has detailed by-week
data

alt_details_updt = alt_details.copy()

# Create a numeric interval variable with an offset of 0.05 to align with
placebo group

import numpy as np

def get_intervals(interval, trt, offset = 0.05):
    if interval == 'Week 1 - 5':
        group = 1.0
    elif interval == 'Week 6 - 10':
        group = 2.0
    elif interval == 'Week 11 - 15':
        group = 3.0
    elif interval == 'Week 16 +':
        group = 4.0

    if trt == 1:
        group += offset

    return group

# apply the above function

alt_details_updt['Interval_id'] = np.vectorize(get_intervals)
                  (alt_details_updt['Interval'],
                   alt_details_updt['Treatment'])

# Build the Dashboard

import dash
import plotly.express as px
from dash import Dash, dcc, html, Input, Output, no_update

app = Dash(__name__)

app.layout = html.Div(
    children=[
        dcc.Graph(id="graph-5", figure=fig, clear_on_unhover=True),
        dcc.Tooltip(id="graph-tooltip-5", direction="top"),
    ],
    style={"height": 800, "padding": 50},
)

# Define inputs and outputs

@app.callback(
    Output("graph-tooltip-5", "show"),
    Output("graph-tooltip-5", "bbox"),
    Output("graph-tooltip-5", "children"),
    Input("graph-5", "hoverData"),
)
```

```
def update_tooltip_content(hoverData):
    if hoverData is None:
        return no_update

    pt = hoverData["points"][0]
    bbox = pt["bbox"]
    num = pt["pointNumber"]

    # subset the data for strip plot and plot it
    dff = alt_details_updt[alt_details_updt.Interval_id == pt["x"]]
    fig_s = px.strip(dff, y="Interval_id", x="Onset_week",
                     color_discrete_sequence = ["black"])

    # other cosmetic updates
    fig_s.update_layout(
        title = dict(
            text="Event Onset Distribution",
            x = 0.5,
            font = dict(size=12)
        ),
        xaxis = dict(
            showgrid = True,
            title = "Onset Week",
            titlefont = dict(size=12)
        ),
        yaxis = dict(
            title = ' ',
            tickvals = [],
            ticktext = []
        ),
        margin=dict(l=0, r=0, t=30, b=0)
    )
    children = [dcc.Graph(figure=fig_s, style={"height": 200, 'width' :
                          300})]

    return True, bbox, children


if __name__ == "__main__":
    app.run_server(debug=True, use_reloader=False)
```