PharmaSUG 2025 - Paper DV- 071

Visualizing oncology data through 3D bar charts in R and Python

Girish Kankipati, Pfizer Inc.
Venkatesulu Salla, Pfizer Inc.

ABSTRACT

3D bar charts are powerful tools for visualizing complex data sets, particularly in clinical trial analyses. This paper explores the use of R and Python for creating 3D bar charts to effectively present oncology data, facilitating better analysis and decision making. The visualization of clinical trial results and patient outcomes data in oncology often requires sophisticated tools to highlight patterns and trends. R and Python, known for their robust data analysis and visualization capabilities which provide versatile libraries such as ggplot2, plotly, and matplotlib, are excellent options to support the development of 3D visualizations.

This paper demonstrates practical workflows for implementing 3D bar charts in both programming languages, including data preparation, chart customization, and advanced interactivity. Through comparative analysis, the paper highlights the strengths and limitations of each platform, focusing on their performance, ease of use, and adaptability for different oncology datasets. Examples include visualizing patient response rates across treatment groups or assessing biomarker distributions across demographic categories. By leveraging R and Python, researchers and clinicians can create visually appealing and interactive 3D bar charts to communicate findings effectively. This not only enhances understanding among stakeholders but also aids in identifying critical insights that might be obscured in raw datasets. The study underscores the importance of integrating advanced visualization techniques into oncology workflows, paving the way for more informed decision-making in clinical and research settings.

INTRODUCTION

In the ever-evolving field of biomedical research, data visualization serves as a powerful tool, especially in oncology, where massive datasets are continuously generated from clinical trials, patient records, and genomic studies. The ability to translate complex data into clear, insightful visual representations is essential for researchers, clinicians, and policymakers to identify patterns, uncover hidden trends, and make well-informed decisions about cancer treatment strategies. Among the various visualization techniques available, three-dimensional (3D) bar charts stand out as a compelling way to display multidimensional data, making them particularly valuable for analyzing oncology datasets that incorporate multiple variables, such as cancer types, treatment modalities, and patient survival rates.

While traditional two-dimensional (2D) bar charts are commonly used for categorical data representation, they often fall short when it comes to illustrating relationships between three or more variables. 3D bar charts overcome this limitation by introducing an additional axis, allowing researchers to visualize intricate interactions between multiple factors. In oncology research, where aspects such as tumor staging, treatment responses, and patient demographics play critical roles, well-structured 3D bar charts enhance pattern recognition and enable meaningful comparisons across different patient groups, ultimately leading to deeper insights and better outcomes.

Both R and Python provide powerful tools for creating 3D visualizations, each offering unique advantages. R, a widely used statistical computing language, includes libraries such as plot3D and rgl, which facilitate the development of high-quality, publication-ready 3D bar charts. On the other hand, Python offers libraries like matplotlib, plotly, and seaborn, which enable dynamic, interactive, and visually engaging visualizations ideal for exploratory data analysis and presentation. By harnessing the strengths of both languages, researchers can select the most effective approach tailored to their specific needs—whether for statistical modeling, interactive data exploration, or comprehensive reporting.

This paper dives deep into the application of 3D bar charts in R and Python for oncology data visualization. It explores the methodologies involved in data preparation, chart creation, and interpretation

while emphasizing the advantages of 3D visualization over conventional methods. Through case studies and practical demonstrations, we illustrate how 3D bar charts can revolutionize oncological research, enhance data-driven decision-making, and contribute to more effective cancer treatment and patient care.

PLOTLY PACKAGE

The plotly package is a powerful library in R for creating interactive visualizations, including 3D bar charts, scatter plots, line charts, and more. It is widely used for data visualization in scientific research, including oncology data. Below are the key features.

- Interactivity: Allows zooming and filtering.
- 3D Visualizations: Supports 3D bar charts, scatter plots, surface plots, etc.
- Integration: Works with ggplot2 in R (ggplotly()) and integrates with Pandas, NumPy, and Dash in Python.
- Customization: Allows extensive formatting of axes, labels, legends, and tooltips.

PLOT3D PACKAGE

The plot3D package is used for static 3D plots. This package provides the scatter3D() function for 3D scatter plots. In oncology data (e.g., tumor size, patient response, gene expression), plot3D can help visualize the following, among many others.

- 3D scatter plots to show relationships between tumor markers and response.
- 3D histograms to analyze trial participants response distributions.
- 3D surface plots to visualize gene expression patterns.

RGL PACKAGE

The rgl package in R transforms data visualization by bringing interactivity to 3D plots through open graphics library (OpenGL) rendering. Unlike traditional static graphs, rgl allows users to dynamically rotate, zoom, and explore data, making it an invaluable tool for in-depth analysis. With support for 3D scatter plots, surfaces, wireframes, and histograms, it excels in creating visually rich and customizable representations of complex datasets. Its ability to export interactive WebGL visualizations, along with high-resolution images and 3D object files, enhances rgl's versatility across research, engineering, and scientific computing. Seamlessly integrating with R Shiny and knitr, rgl enables the creation of interactive reports and dashboards, making data exploration more intuitive and engaging.

Key Features of the rgl Package

- Interactive 3D visualization
- Users can rotate, zoom, and pan the 3D plots using the mouse

Wide Range of 3D Plot Types

- Scatter plots (plot3d)
- Surface plots (persp3d)
- Wireframes (wire3d)
- 3D histograms (hist3d)
- 3D text and annotations (text3d)

High-Quality Graphics

- Allows anti-aliasing, lighting, and transparency for enhanced visualization.
- Supports shaders and material properties to modify surface appearance.

Export and Save Options

- Supports exporting plots as interactive web pages (WebGL).
- Can save plots as images (PNG, JPG) or 3D object files (OBJ, STL).

Integration with Other Packages

- Works well with ggplot2 via plotly (ggplotly()) for interactive plots.
- Compatible with R Shiny for web-based dashboards.

TYPES OF BAR CHARTS IN R

3D bar charts in R can be created using various packages. Some of the packages include plot3D, rgl and latticeExtra as described above. Some of the types of bar charts are presented below.

- Simple bar chart
- Static 3D ba chart using plot3D
- Interactive 3D bar chart using rgl
- 3D bar chart using latticeExtra

Simple bar chart

The plot3D package in R offers a user-friendly method for creating 3D bar charts through its hist3D function, rendering a matrix of numerical values as bars in a three-dimensional space. This section details a simple 3D bar chart implementation, emphasizing minimal parameters to visualize clinical trial data effectively. The approach prioritizes ease of use and clarity, making it accessible for basic data representation. Figure 1 shows the three-dimensional bar chart that represents the time, treatment and response variables n X, Y, and Z axis.

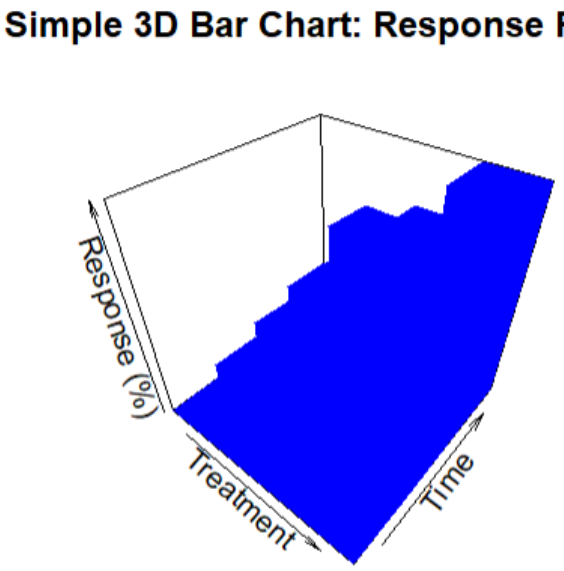

Figure 1 Static 3D bar chart using plot3D

The plot3D package is used for static 3D bar chart generation with enhanced visualization capabilities and addressing a range of plotting needs from scatter and surface representations to bar and ribbon diagrams. Unlike packages such as rgl, which emphasize interactive graphics, plot3D prioritizes static outputs suitable for printed reports, scientific publications, and reproducible research.

Program 1 shown below generates a static 3D bar (see Figure 2) using the hist3D function from the plot3D package to visualize patient response rates across treatment groups and time points in a clinical study. The chart is constructed with treatments defining the x-axis (categorized as Placebo, Drug A, Drug B, and Drug C), timepoints on the y-axis (labeled as 1, 2, 3, 4 i.e Week 2, Week 4, Week 6, and Week 8), and percent response rates on the z-axis. Bars are colored light green with black borders and a slight shading effect (shade = 0.3) for depth. Viewing angles are set with theta = 40 and phi = 25 to optimize 3D perspective. The plot includes a title, "Patient Response Rates by Treatment and Time," and labeled axes: "Treatment Group" (x), "Time Point (Weeks)" (y), and "Response Rate (%)" (z). Detailed tick marks are enabled (ticktype = "detailed") with custom labels for treatments (xticklabs) and time points (yticklabs), and axis text size is reduced (cex.axis = 0.7) for clarity. This static visualization effectively displays response rate variations across four treatment groups over four time points in a single, 3D representation.

```
install.packages("plot3D")
library(plot3D)

# Define data
treatments <- 1:4  # Represents Placebo, Drug A, Drug B, Drug C
time_points <- 1:4  # Represents Week 2, Week 4, Week 6, Week 8
response_rates <- matrix(c(20, 25, 30, 35,  # Placebo
                           30, 40, 50, 60,  # Drug A
                           35, 45, 55, 65,  # Drug B
                           40, 55, 70, 80), # Drug C
                         nrow = 4, ncol = 4, byrow = TRUE)

# Create static 3D bar chart
hist3D(x = treatments, y = time_points, z = response_rates,
       col = "lightgreen", border = "black", shade = 0.3,
       theta = 40, phi = 25,
       main = "Patient Response Rates by Treatment and Time",
       xlab = "Treatment Group", ylab = "Time Point (Weeks)", zlab = "Response Rate (%)",
       ticktype = "detailed", cex.axis = 0.7,
       xticklabs = c("1", "2", "3", "4"),
       yticklabs = c("Wk 2", "Wk 4", "Wk 6", "Wk 8"))
```

Program 1. Simple 3D bar chart program

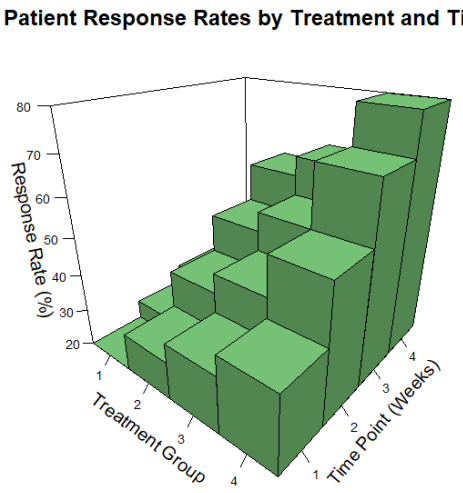

Figure 2. 3D bar for response data for different time points

Interactive 3D Bar Chart using rgl

The rgl package in R leverages OpenGL to produce interactive 3D visualizations, including bar charts, which allow users to rotate, zoom, and explore data dynamically. Unlike the static outputs of plot3D, rgl facilitates real-time manipulation, making it ideal for exploratory data analysis. This example constructs an interactive 3D bar chart to represent patient response rates from a clinical trial across treatment groups and time points, showcasing the package's flexibility and interactivity.

Four treatment groups—Placebo, Drug A, Drug B, and Drug C—are evaluated across four time points: Week 2, Week 4, Week 6, and Week 8. The data is structured as a 4x4 matrix, with rows representing treatment groups and columns representing time points. Each cell value denotes the response rate as a percentage, providing a basis for comparing treatment efficacy over time.

Program 2, provided below, generates an interactive 3D bar chart as shown in Figure 3 using rgl. Since rgl does not provide a direct hist3D-like function for bar charts, the different segments of the bars are constructed manually using 3D cuboids via cube3d and translation3d. The bars, rendered in a shaded blue-gray tone, provide a visual depth, enhancing the perception of varying response rates across the three treatment groups over time.

```
# Load the rgl package
install.packages("rgl")
library(rgl)
# Define data
treatments <- 1:4  # Placebo, Drug A, Drug B, Drug C
time_points <- 1:4  # Week 2, Week 4, Week 6, Week 8
response_rates <- matrix(c(20, 25, 30, 35,  # Placebo
                           30, 40, 50, 60,  # Drug A
                           35, 45, 55, 65,  # Drug B
                           40, 55, 70, 80), # Drug C
                         nrow = 4, ncol = 4, byrow = TRUE)
# Open a new rgl device
open3d()
# Set up plot parameters
bg3d("white")  # White background
view3d(theta = 45, phi = 25)  # Initial viewing angle
# Create bars as cuboids
for (i in 1:4) {  # Loop over treatments (x-axis)
  for (j in 1:4) {  # Loop over time points (y-axis)
    height <- response_rates[i, j] / 100  # Scale height (0 to 0.8)
    bar <- cube3d()  # Create a unit cube
    bar <- scale3d(bar, 0.4, 0.4, height)  # Scale width, depth, height
    bar <- translate3d(bar, i - 0.5, j - 0.5, height / 2)  # Position bar
    shade3d(bar, col = "lightblue", alpha = 0.8)  # Render with color  } }
# Add axes and labels
axes3d(edges = c("x", "y", "z"), labels = TRUE, tick = TRUE)
```

```
title3d(main = "Interactive 3D Bar Chart: Response Rates",
        xlab = "Treatment Group", ylab = "Time Point (Weeks)", zlab = "Response Rate (%)")
# Customize axis labels
rgl.texts(x = 1:4, y = rep(0, 4), z = rep(0, 4),
          text = c("Placebo", "Drug A", "Drug B", "Drug C"), adj = c(0.5, 1.5), cex = 0.8)
rgl.texts(x = rep(0, 4), y = 1:4, z = rep(0, 4),
          text = c("Wk 2", "Wk 4", "Wk 6", "Wk 8"), adj = c(1.5, 0.5), cex = 0.8)
# Load the rgl package
install.packages("rgl")
library(rgl)
# Define data
treatments <- 1:4  # Placebo, Drug A, Drug B, Drug C
time_points <- 1:4  # Week 2, Week 4, Week 6, Week 8
response_rates <- matrix(c(20, 25, 30, 35,  # Placebo
                           30, 40, 50, 60,  # Drug A
                           35, 45, 55, 65,  # Drug B
                           40, 55, 70, 80), # Drug C
                         nrow = 4, ncol = 4, byrow = TRUE)
# Open a new rgl device
open3d()
# Set up plot parameters
bg3d("white")  # White background
view3d(theta = 45, phi = 25)  # Initial viewing angle
# Create bars as cuboids
for (i in 1:4) {  # Loop over treatments (x-axis)
  for (j in 1:4) {  # Loop over time points (y-axis)
    height <- response_rates[i, j] / 100  # Scale height (0 to 0.8)
    bar <- cube3d()  # Create a unit cube
    bar <- scale3d(bar, 0.4, 0.4, height)  # Scale width, depth, height
    bar <- translate3d(bar, i - 0.5, j - 0.5, height / 2)  # Position bar
    shade3d(bar, col = "lightblue", alpha = 0.8)  # Render with color   } }
# Add axes and labels
axes3d(edges = c("x", "y", "z"), labels = TRUE, tick = TRUE)
title3d(main = "Interactive 3D Bar Chart: Response Rates",
        xlab = "Treatment Group", ylab = "Time Point (Weeks)", zlab = "Response Rate (%)")
# Customize axis labels
rgl.texts(x = 1:4, y = rep(0, 4), z = rep(0, 4),
          text = c("1", "2", "3"), adj = c(0.5, 1.5), cex = 0.8)
rgl.texts(x = rep(0, 4), y = 1:4, z = rep(0, 4),
          text = c("Wk 2", "Wk 4", "Wk 6", "Wk 8"), adj = c(1.5, 0.5), cex = 0.8)
```

Program 2. Interactive 3D bar chart program

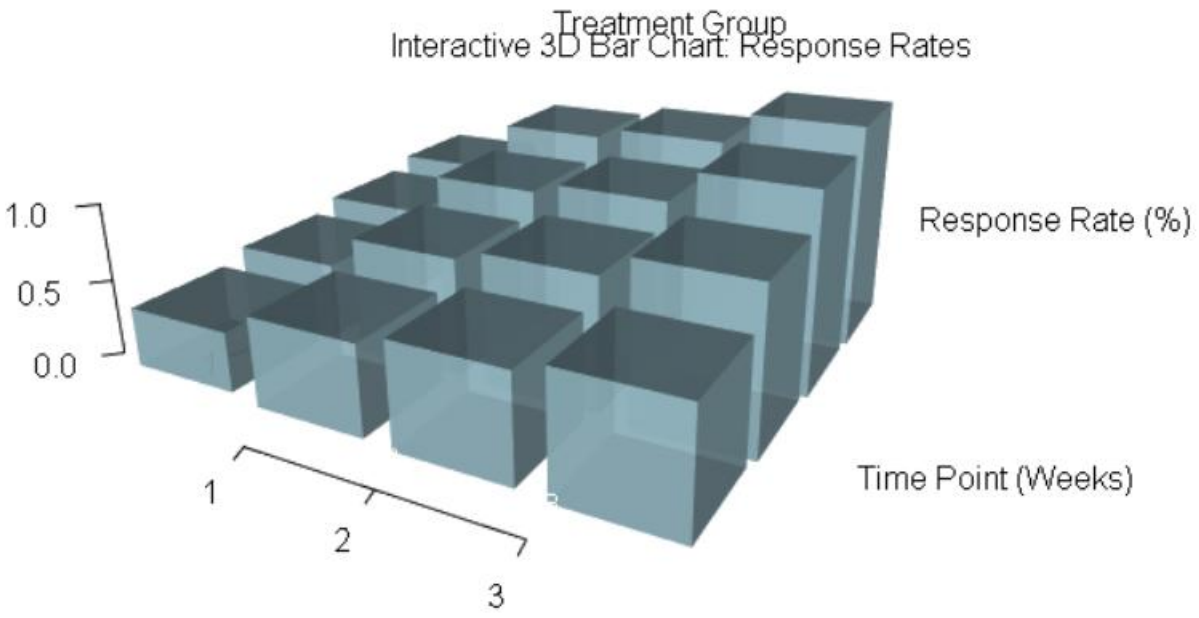

Figure 3. Interactive rotating 3D bar plot for response data for different time points

The resulting interactive 3D bar chart displays:

- The x-axis with treatment groups (1 to 4: Placebo, Drug A, Drug B, Drug C).
- The y-axis with time points (1 to 4: Week 2 to Week 8).
- The z-axis with response rates (0 to 1%), where bar heights correspond to matrix values.

Users can rotate the plot by clicking and dragging, zoom with the mouse wheel, and pan by right-clicking, enabling detailed exploration. For instance, rotating to a side view highlights Drug C's superior response rates (40% to 80%) compared to Placebo (20% to 35%).

3D bar chart using latticeExtra

The lattice package in R, when extended by latticeExtra, provides a framework for creating 3D visualizations, including bar-like representations, through its cloud function. While lattice is primarily known for trellis graphics, it supports 3D plotting via perspective projections, and latticeExtra enhances customization options. Program 3 below constructs a 3D bar chart in Figure 4 to represent participant response rates from a clinical trial across treatment groups and time points, offering a static yet flexible alternative to plot3D and rgl. Note that lattice does not natively produce solid bars as in hist3D or rgl; instead, it approximates a bar chart using points or lines with depth cues.

```
# Load required packages
library(lattice)
library(latticeExtra)
# Define data as a matrix
response_rates <- matrix(c(20, 25, 30, 35,  # Placebo
                           30, 40, 50, 60,  # Drug A
                           35, 45, 55, 65,  # Drug B
                           40, 55, 70, 80), # Drug C
                         nrow = 4, ncol = 4, byrow = TRUE)
# Convert matrix to long-format data frame
data <- expand.grid(Treatment = 1:4, Time = 1:4)
data$Response <- as.vector(t(response_rates))
# Define custom labels
treatment_labels <- c("Placebo", "Drug A", "Drug B", "Drug C")
time_labels <- c("Wk 2", "Wk 4", "Wk 6", "Wk 8")
# Create 3D bar-like chart using cloud
cloud_plot <- cloud(Response ~ Treatment * Time, data = data,
                    type = "h",  # Vertical lines to simulate bars
                    pch = 19,    # Solid points at bar tops
                    cex = 1.5,   # Point size
                    col = "lightblue",  # Color for lines and points
                    main = "3D Bar Chart: Response Rates by Treatment and Time",
                    xlab = "Treatment Group", ylab = "Time Point (Weeks)", zlab = "Response
Rate (%)",
                    scales = list(arrows = FALSE, distance = 1,
                                  x = list(labels = treatment_labels, at = 1:4),
                                  y = list(labels = time_labels, at = 1:4)),
                    screen = list(z = 40, x = -30))  # Viewing angle
# Print the plot
print(cloud_plot)
```

Program 3. LatticeExtra 3D bar chart program

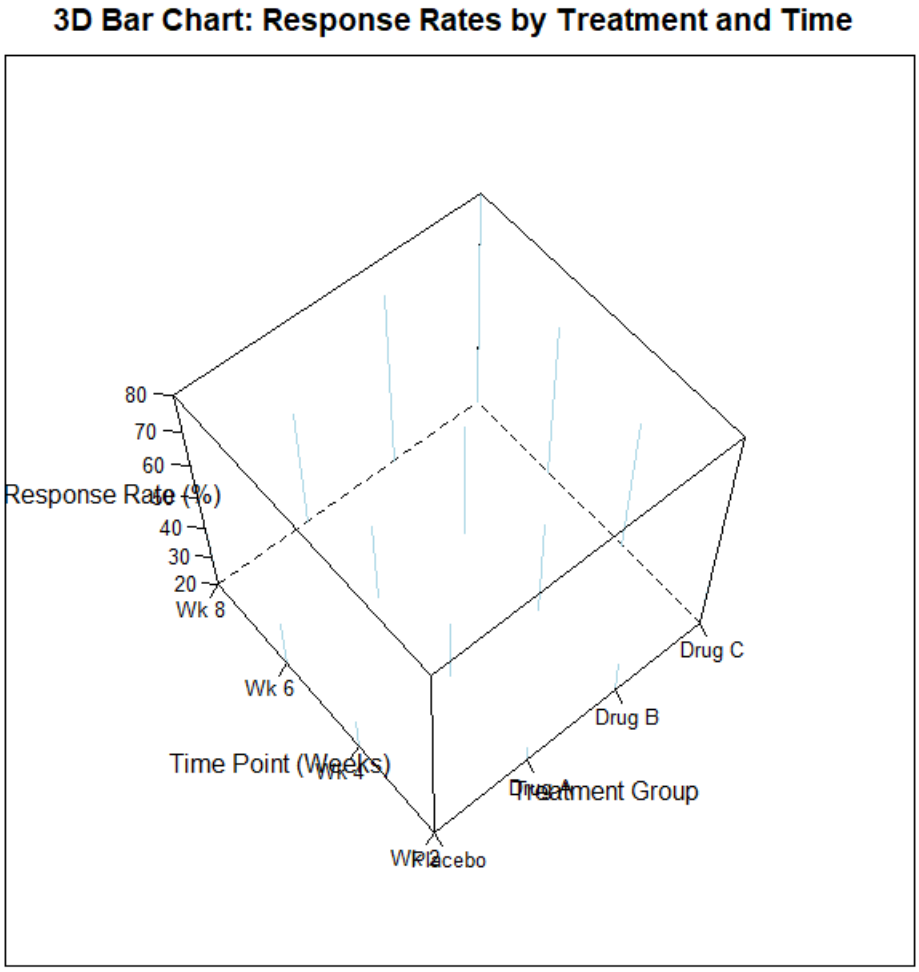

Figure 4. 3D Bar Chart using latticeExtra package

In Figure 5Figure 5, we present sample data from a dummy data set generated in R. The data set consists of tumor response data for 200 oncology trial participants evaluated using RECIST 1.1 criteria across four timepoints: Baseline, Week 6, Week 12, and Week 18. At baseline, all subjects were not evaluable (NE) as expected prior to treatment assessment. By week 6, 5% achieved Complete Response (CR), 20% Partial Response (PR), 50% Stable Disease (SD), and 25% remained NE, indicating early treatment effects. At Week 12, CR increased to 10%, PR to 25%, SD to 40%, with NE steady at 25%, reflecting peak response. By Week 18, CR rose to 15%, PR held at 25%, SD dropped to 35%, and NE remained 25%, suggesting sustained efficacy with some disease progression. These findings highlight a progressive improvement in tumor response over time, consistent with typical oncology trial outcomes. The sample dummy data set below was generated from R and has key variables subject_ID, visit, and overall_response.

	Subject_ID	Visit	Overall_Response
1	SUBJ001	Baseline	NE
2	SUBJ001	Week 6	SD
3	SUBJ001	Week 12	PD
4	SUBJ001	Week 18	PD
5	SUBJ002	Baseline	NE
6	SUBJ002	Week 6	PD
7	SUBJ002	Week 12	PR
8	SUBJ002	Week 18	PR
9	SUBJ003	Baseline	NE
10	SUBJ003	Week 6	PR
11	SUBJ003	Week 12	PD
12	SUBJ003	Week 18	PD
13	SUBJ004	Baseline	NE
14	SUBJ004	Week 6	SD
15	SUBJ004	Week 12	CR

Figure 5. Partially shown sample data set with subject, visit and response

```
# Load required packages
library(plotly)

# Set seed for reproducibility
set.seed(123)

# Number of subjects
n_subjects <- 200

# Create Subject IDs
subject_id <- sprintf("SUB%03d", 1:n_subjects)

# Define visit timepoints
visits <- c("Baseline", "Week 6", "Week 12", "Week 18")

# Generate responses based on paragraph percentages
# Baseline: 100% NE
baseline_response <- rep("NE", n_subjects)
# Week 6: 5% CR, 20% PR, 50% SD, 25% NE
week6_response <- sample(c("CR", "PR", "SD", "NE"), n_subjects, replace = TRUE,
                         prob = c(0.05, 0.20, 0.50, 0.25))
# Week 12: 10% CR, 25% PR, 40% SD, 25% NE
week12_response <- sample(c("CR", "PR", "SD", "NE"), n_subjects, replace = TRUE,
                          prob = c(0.10, 0.25, 0.40, 0.25))
# Week 18: 15% CR, 25% PR, 35% SD, 25% NE
week18_response <- sample(c("CR", "PR", "SD", "NE"), n_subjects, replace = TRUE,
                          prob = c(0.15, 0.25, 0.35, 0.25))

# Create dataset
recist_data <- data.frame(
  Subject_ID = rep(subject_id, times = length(visits)),
  Visit = rep(visits, each = n_subjects),
  Response = c(baseline_response, week6_response, week12_response, week18_response)
```

```
)

# Prepare data for plotly (4 bars per response)
responses <- c("CR", "PR", "SD", "NE")
bar_data <- data.frame(
  Response = rep(responses, 4),
  Count = c(table(recist_data$Response[recist_data$Visit == "Baseline"])[responses],
            table(recist_data$Response[recist_data$Visit == "Week 6"])[responses],
            table(recist_data$Response[recist_data$Visit == "Week 12"])[responses],
            table(recist_data$Response[recist_data$Visit == "Week 18"])[responses]),
  Visit = rep(visits, each = length(responses))
)

# Create interactive bar plot with 3D-like enhancements
p <- plot_ly(bar_data,
             x = ~Response,
             y = ~Count,
             color = ~Visit,
             type = "bar",
             colors = c("#1f77b4", "#ff7f0e", "#2ca02c", "#d62728"),  # Four colors
             marker = list(line = list(color = "black", width = 1)),  # Outlines for depth
             text = ~paste("Response:", Response, "<br>Visit:", Visit, "<br>Count:",
Count),
             hoverinfo = "text",
             opacity = 0.9) %>%
  layout(
    title = list(text = "Figure 5: Tumor Response Over Time (RECIST 1.1)", y = 0.95),
    xaxis = list(
      title = "Response Category",
      tickangle = -45  # Tilt labels for perspective
    ),
    yaxis = list(
      title = "Number of Subjects",
      gridcolor = "gray"  # Grid lines for depth
    ),
    barmode = "group",
    bargap = 0.2,  # Space between bars
    bargroupgap = 0.1,  # Space between groups for 3D effect
    legend = list(
      title = list(text = "Visit"),
      x = 1.05,  # Move legend outside to the right
      y = 0.9,   # Align near top
      xanchor = "left",  # Anchor to avoid overlap
      yanchor = "top",
      bgcolor = "rgba(255, 255, 255, 0.7)"  # Slight background for readability
    ),
    margin = list(l = 50, r = 150, b = 100, t = 100),  # Increase right margin for legend
space
    paper_bgcolor = "#f0f0f0",  # Light background
    plot_bgcolor = "#ffffff",
    updatemenus = list(
      list(
        y = 0.9,
        buttons = list(
          list(method = "restyle",
               args = list("barmode", "group"),
               label = "Grouped"),
          list(method = "restyle",
               args = list("barmode", "stack"),
               label = "Stacked")
        )
      ),
      list(
        y = 0.8,
        buttons = list(
          list(method = "restyle",
               args = list("opacity", 0.9),
```

```
                label = "Solid"),
          list(method = "restyle",
                args = list("opacity", 0.6),
                label = "Semi-Transparent")
        )
      )
    ),
    sliders = list(
      list(
        active = 3,
        steps = list(
          list(method = "restyle",
                args = list("visible", list(TRUE, FALSE, FALSE, FALSE)),
                label = "Baseline"),
          list(method = "restyle",
                args = list("visible", list(FALSE, TRUE, FALSE, FALSE)),
                label = "Week 6"),
          list(method = "restyle",
                args = list("visible", list(FALSE, FALSE, TRUE, FALSE)),
                label = "Week 12"),
          list(method = "restyle",
                args = list("visible", list(FALSE, FALSE, FALSE, TRUE)),
                label = "Week 18"),
          list(method = "restyle",
                args = list("visible", list(TRUE, TRUE, TRUE, TRUE)),
                label = "All Visits")
        ),
        currentvalue = list(prefix = "Show: "),
        pad = list(t = 20)
      )
    )
  )

# Display the plot
p

# Save as HTML
htmlwidgets::saveWidget(p, "figure5_recist_tumor_response.html")
```

Program 4. Interactive 3D bar chart program for response data.

An interactive bar plot was created using R and Plotly (version 4.10.1) to show tumor response categories (CR, PR, SD, PD) based on RECIST 1.1 from a simulated response dataset. This dataset includes 200 subjects with responses set as CR: 10%, PR: 30%, SD: 40%, PD: 20%, and focuses on four visits: Week 0, 12, 18, and 24. Each response category has four bars, one per visit, totaling 16 bars. Simple 3D-like effects come from angled x-axis labels, grid lines, and bar outlines, with colors for each visit (Week 0: blue, Week 6: red, Week 12: orange, Week 18: green). Text gives response, visit, and count details, dropdowns switch between grouped or stacked views and adjust opacity, and a slider filters visits. The legend sits outside the plot with a wider right margin to avoid overlap. Program 4 shown earlier generates the simulated data as well as Figure 6 in the interactive format.

Figure 6. 3D Interactive bar plot of RECIST 1.1 responses

PYTHON BAR CHART

Python is a powerful and easy-to-learn programming language used in data analysis, machine learning, web development, and more. It is widely used in data science because of its versatility and strong libraries for data manipulation and visualization. Creating 3D bhar charts in Python involves these steps: install necessary libraries, prepare your data, and create 3D bhar chart.

To create 3D bar charts, the most commonly used libraries are:

- Matplotlib: A popular library for creating static, animated, and interactive visualizations.
- Plotly: An interactive graphing library that works in the web browser, ideal for more dynamic charts.

The pandas library is used to create a dataframe from the data list. This allows for efficient data manipulation and analysis. The dataframe is a tabular structure, where each row represents a specific combination of treatment arm, response category, and count of trial participants. The python code in Program 5 below generates a 3D bar chart to visualize oncology data, showing how different treatment arms and response categories are related in a clinical study. The data variable is a list of dictionaries, where each dictionary contains information about the treatment arm, response category, and the count of trial participants in each category (e.g., PR, PD, SD, and CR) for two treatment arms - placebo and study drug. Figure 7 shows the 3D representation of response data with corresponding time point using the Program 5. 3D bar chart program for RECIST1.1 responses Program 5.

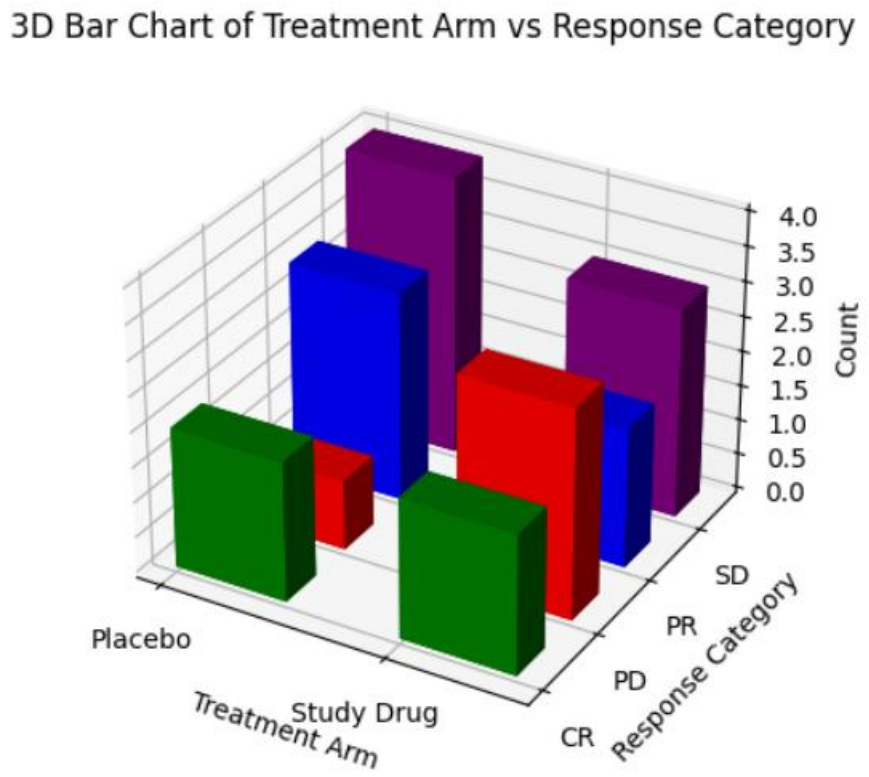

Figure 7. 3D bar chart of RECIST 1.1 responses in Python

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Data
treatment_arms = ['Placebo', 'Placebo', 'Placebo', 'Placebo', 'Study Drug', 'Study Drug',
'Study Drug', 'Study Drug']
response_categories = ['PR', 'PD', 'SD', 'CR', 'PR', 'PD', 'SD', 'CR']
counts = [3, 1, 4, 2, 2, 3, 3, 2]

# Unique values for the axes
unique_treatment_arms = sorted(set(treatment_arms))
unique_response_categories = sorted(set(response_categories))

# Create a dictionary to map each combination of Treatment Arm and Response Category to
its count
count_dict = {(arm, response): 0 for arm in unique_treatment_arms for response in
unique_response_categories}
for arm, response, count in zip(treatment_arms, response_categories, counts):
    count_dict[(arm, response)] = count

# Prepare the grid for the 3D plot
x_pos = []
y_pos = []
z_pos = np.zeros(len(count_dict))  # Bottom of each bar starts at zero

dx = np.ones(len(count_dict))  # Width of each bar
dy = np.ones(len(count_dict))  # Depth of each bar
dz = list(count_dict.values())  # Height of each bar

# Define colors for each response category
```

```
response_colors = {
    'PR': 'green',  # Partial Response
    'PD': 'red',    # Progressive Disease
    'SD': 'blue',   # Stable Disease
    'CR': 'purple'  # Complete Response
}

# Assign positions for each Treatment Arm and Response Category with separation
x_offset = 1  # Adjust this to control separation between treatment arms
y_offset = 1  # Adjust this to control separation between response categories

# Map the treatment arm and response category to positions with separation
for i, (arm, response) in enumerate(count_dict.keys()):
    x_pos.append(unique_treatment_arms.index(arm) * (x_offset + 1))  # Separate by
x_offset
    y_pos.append(unique_response_categories.index(response) * (y_offset + 1))  # Separate
by y_offset

# Create the plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Assign colors to each bar based on the response category
colors = [response_colors[response] for response in response_categories]

# Create the 3D bars
ax.bar3d(x_pos, y_pos, z_pos, dx, dy, dz, color=colors, zsort='average')

# Labels
ax.set_xlabel('Treatment Arm')
ax.set_ylabel('Response Category')
ax.set_zlabel('Count')

# Set ticks for x and y axes with adjusted positions for spacing
ax.set_xticks(np.arange(len(unique_treatment_arms)) * (x_offset + 1))
ax.set_xticklabels(unique_treatment_arms)

ax.set_yticks(np.arange(len(unique_response_categories)) * (y_offset + 1))
ax.set_yticklabels(unique_response_categories)

plt.title('3D Bar Chart of Treatment Arm vs Response Category')

plt.show()
```

Program 5. 3D bar chart program for RECIST1.1 responses

Below is a step-by-step description of the code:

Importing Required Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

- numpy is used for array and mathematical operations.
- matplotlib.pyplot is used for plotting the graph.
- mpl_toolkits.mplot3d enables the creation of 3D plots.

Defining the Data

```
treatment_arms = ['Placebo', 'Placebo', 'Placebo', 'Placebo', 'Study Drug', 'Study Drug',
'Study Drug', 'Study Drug']
response_categories = ['PR', 'PD', 'SD', 'CR', 'PR', 'PD', 'SD', 'CR']
counts = [3, 1, 4, 2, 2, 3, 3, 2]
```

- treatment_arms lists the treatment arms (e.g., Placebo, Study Drug).
- response_categories lists the different types of responses (e.g., PR = Partial Response, PD = Progressive Disease).
- Counts contains the number of trial participants observed for each combination of treatment arm and response category.

Processing the Data

```
unique_treatment_arms = sorted(set(treatment_arms))
unique_response_categories = sorted(set(response_categories))
```

- unique_treatment_arms and unique_response_categories store the unique and sorted treatment arms and response categories.

```
count_dict = {(arm, response): 0 for arm in unique_treatment_arms for response in
unique_response_categories}
for arm, response, count in zip(treatment_arms, response_categories, counts):
count_dict[(arm, response)] = count
```

- count_dict is created to store counts for each unique combination of treatment arm and response category.
- The zip function is used to pair each treatment arm, response category, and count, which are then stored in the dictionary.

.

Preparing Data for 3D Plot

```
x_pos = []
y_pos = []
z_pos = np.zeros(len(count_dict)) # All bars start at height zero
dx = np.ones(len(count_dict)) # Set width of each bar to 1
dy = np.ones(len(count_dict)) # Set depth of each bar to 1
dz = list(count_dict.values()) # Heights of the bars correspond to counts
```

- x_pos, y_pos, and z_pos store the positions of the bars in 3D space.
- dx and dy set the width and depth of the bars, respectively, both set to 1 for uniformity.
- dz corresponds to the height of each bar, which is based on the counts in count_dict.

Assigning Positions and Colors

```
response_colors = {
'PR':'green' ,  # Partial Response
'PD':'red' # Progressive Disease
'SD': 'blue',  # Stable Disease
'CR': 'purple' # Complete Response
}
```

- response_colors is a dictionary to assign colors to each response category.

```
x_offset = 1 # Adjust this to control separation between treatment arms
```

```
y_offset = 1 # Adjust this to control separation between response categories
```

- x_offset and y_offset control the spacing between the bars in the x and y directions.

```
for i, (arm, response) in enumerate(count_dict.keys()):
x_pos.append(unique_treatment_arms.index(arm) * (x_offset + 1))
y_pos.append(unique_response_categories.index(response) * (y_offset + 1))
```

- The x_pos and y_pos are updated to assign unique positions to each bar, based on the indices of treatment arms and response categories. The offset values ensure the separation between adjacent bars.

Creating the 3D Plot

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

- A figure and a 3D axis (ax) are created for plotting.

```
colors = [response_colors[response] for response in response_categories]
```

- colors are a list of colors assigned to each bar based on its corresponding response category.

```
ax.bar3d(x_pos, y_pos, z_pos, dx, dy, dz, color=colors, zsort='average')
```

- ax.bar3d is used to create the 3D bars at the positions specified by x_pos, y_pos, and z_pos. Bar dimensions are set by dx, dy, and dz, and colors are assigned from the colors list.

Setting Labels and Ticks

```
ax.set_xlabel('Treatment Arm')
ax.set_ylabel('Response Category')
ax.set_zlabel('Count')
```

- Labels for the x, y, and z axes are set, corresponding to the treatment arm, response category, and count.

```
ax.set_xticks(np.arange(len(unique_treatment_arms)) * (x_offset + 1))
ax.set_xticklabels(unique_treatment_arms)
ax.set_yticks(np.arange(len(unique_response_categories)) * (y_offset + 1))
ax.set_yticklabels(unique_response_categories)
```

- The x and y ticks are set to match the positions of the treatment arms and response categories, with appropriate spacing for clarity.

Displaying the Plot

```
plt.title('3D Bar Chart of Treatment Arm vs Response Category')
plt.show()
```

- A title is set for the plot, and then plt.show() displays the chart.

CONCLUSION

Visualizing oncology data using 3D bar charts in R and Python can provide additional perspectives on complex datasets. In R, the 'rgl' package facilitates the creation of interactive 3D plots, including bar charts, allowing users to manipulate views for enhanced data comprehension. Similarly, Python's Matplotlib library offers functionalities to generate 3D bar charts, enabling interactive data exploration. While these tools offer advanced visualization capabilities, it's advisable to use 3D visualizations cautiously, ensuring they effectively convey the intended information without compromising clarity.

ACKNOWLEDGMENTS

We would like to acknowledge Avani Kaja and Bala Pitchuka for their valuable suggestions.

REFERENCES

GeeksforGeeks. (n.d.). Displaying 3D images in Python. Retrieved from https://www.geeksforgeeks.org/displaying-3d-images-in-python/

R Core Team & Plotly Team (2016). *plotly: Create interactive web graphics via plotly.js* (Version 4.5.2) [R package]. RDocumentation. https://www.rdocumentation.org/packages/plotly/versions/4.5.2

Adler, D., & Murdoch, D. (2023). *rgl: 3D visualization using OpenGL* (Version 1.2.8) [R package vignette]. CRAN. https://cran.r-project.org/web/packages/rgl/vignettes/rgl.html

McKinney, W., & Pandas Development Team (2023). *pandas: Python data analysis library* (Version 2.2.1) [Python package]. Pandas. https://pandas.pydata.org/

W3Schools. (n.d.). *NumPy introduction* [Online tutorial]. https://www.w3schools.com/python/numpy/numpy_intro.asp

Soetaert, K. (2021). *plot3D: Plotting multi-dimensional data* (Version 1.4) [R package vignette]. CRAN. https://cran.r-project.org/web/packages/plot3D/plot3D.pdf

Sthda. (n.d.). A complete guide to 3D visualization device system in R: R software and data visualization. Retrieved from https://www.sthda.com/english/wiki/a-complete-guide-to-3d-visualization-device-system-in-r-r-software-and-data-visualization

Appsilon (2022, October 27). *Creating stunning 3D charts in R* [Blog post]. https://www.appsilon.com/post/r-3d-charts

RECOMMENDED READING

- *Python Crash Course*
- *R For Dummies*

CONTACT INFORMATION

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

Girish Kankipati
Pfizer Inc.
girish.kankipati@pfizer.com

Venkatesulu Salla
Pfizer Inc.
venkatesulu.salla@pfizer.com