



Oncology trial endpoint tracking using RShiny

PHUSE US Connect 2021

Michael Briganti and Kavitha Nagarajan
mbrigant@its.jnj.com / knagara2@its.jnj.com
Janssen
April 26, 2021

REIMAGINE CARE
REDEFINE LIVING

janssen  Oncology
PHARMACEUTICAL COMPANIES OF 

Presentation Outline

- | | | |
|----------|------------------------------------|-------------------|
| 1 | Leaky Pipe - Background | Kavitha Nagarajan |
| 2 | Network Package Comparisons | Michael Briganti |
| 3 | Lessons Learned | |
| 4 | Example RShiny Application | |
| 5 | Conclusion | |

Leaky Pipe – A background



Why do we need this?

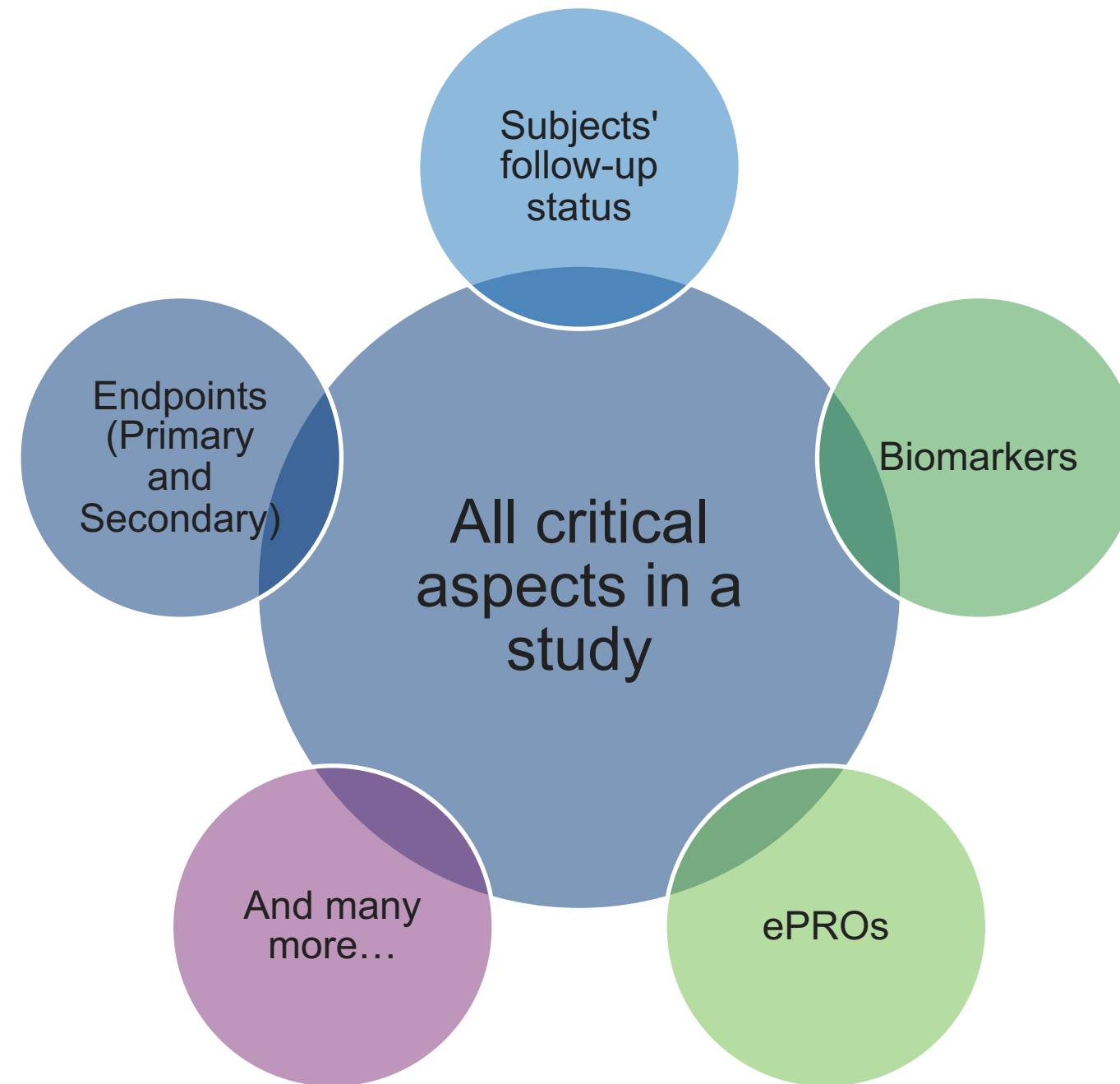
- Oncology trials are based upon attaining certain number of events. It is critical to ensure subjects are followed diligently for planned evaluations and tracking.
- Endpoint tracking and key datapoints checking are critical for data quality and monitoring.
- Cross-functional teams find it immensely useful if relevant study information are presented in a concise manner. Enables teams to focus on areas that needs monitoring and/or tracking.



How to do this?

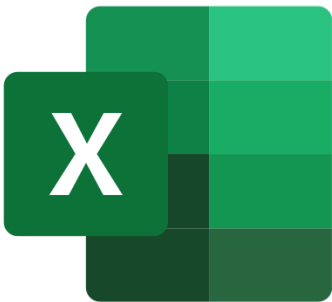
- A leaky pipe report facilitates review of such critical data and provide detailed information about subjects lost to follow-up during the trial.
- Data is summarized periodically and presented in a visual format that is concise and clear.
- Customized based on study needs.

What can be tracked?

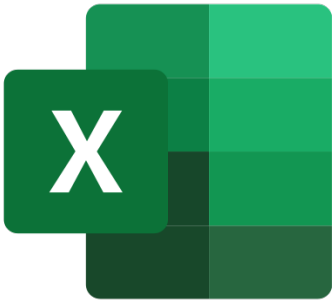


Visualization Workflow

OLD



NEW





Network Packages and RShiny Endpoint Tracker Application

Four Visualization Solutions Explored

Tiles & Text Graph

(Via ggplot2)

- Use Cartesian Coordinates to manually place boxes and text
- Mimic requested output almost exactly

Graphviz Graph

(Via diagrammeR)

- Script Hierarchy of Nodes
- diagrammeR automatically and dynamically renders graph

Sankey Diagram

(via Plotly)

- Standard Sankey diagram
- Endpoints pushed to the final time point (Important difference between Network D3 Sankey)

Sankey Diagram

(via NetworkD3)

- Standard Sankey Diagram
- Endpoints only move one tier below previous
 - Do not get pushed all the way to final timepoint

Tiles and Text – ggplot2

Pros and Cons

Pros

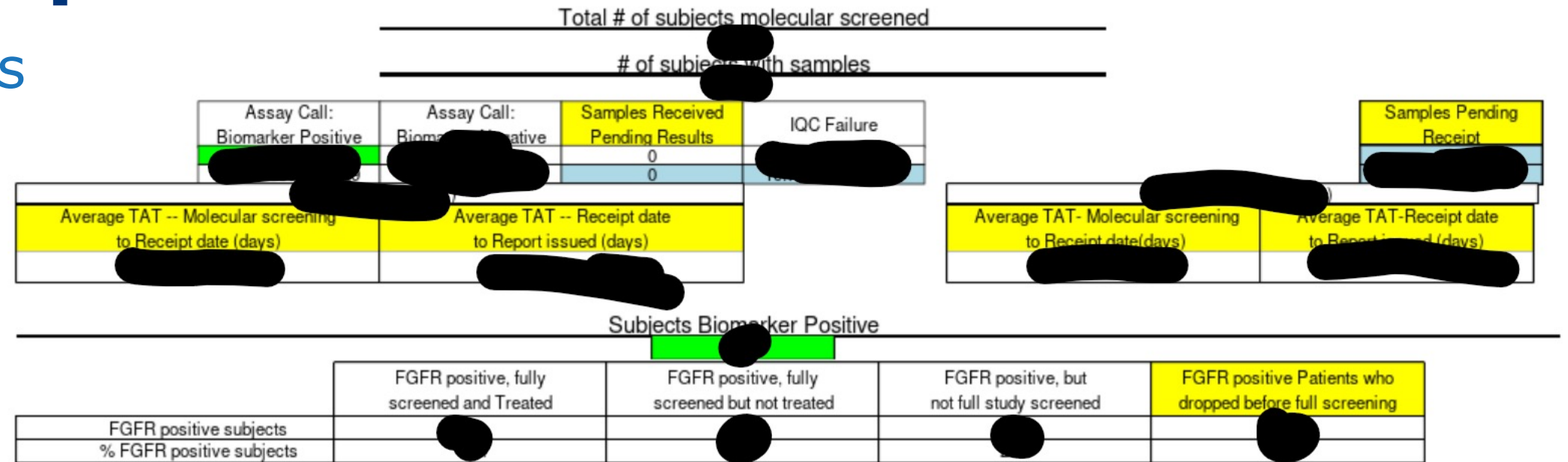
- Programmer Familiarity
- Straight-forward Code
- RShiny Integration
- Match requested output nearly identically

Cons

- Lengthy Code
- Tedious (Not designed to visualize flows)
- Time consuming to modify existing graph

ggplot2 Example

Text and Tile Objects



```
geom_text(aes(label = "Subjects Biomarker Positive", x = 5, y = 60),check_overlap = TRUE)+
geom_segment(aes(x = 1, xend = 9.5, y = 58.5, yend = 58.5))+
geom_tile(aes(x=5,y=57),fill="green",width=1,height=3,color="black")+
geom_text(aes(label=`Count(n)`[4],fontface="bold",x=5,y=57),check_overlap = TRUE)+

#LINE6
geom_tile(mapping=aes(x = 3.5, y = 51.5),fill = "#FFFFFF",height=7,width=1.5,color = "black") +
geom_text(mapping=aes(x = 3.5, y = 51.5, label = "FGFR positive, fully\nscreened and Treated"),
  color = "black", size = 3,check_overlap = TRUE) +
geom_tile(mapping=aes(x = 5, y = 51.5),fill = "#FFFFFF",height=7,width=1.5,color = "black") +
geom_text(mapping=aes(x = 5, y = 51.5, label = "FGFR positive, fully\nscreened but not treated"),
  color = "black", size = 3,check_overlap = TRUE) +
geom_tile(mapping=aes(x = 6.5, y = 51.5),fill = "#FFFFFF",height=7,width=1.5,color = "black") +
geom_text(mapping=aes(x = 6.5, y = 51.5, label = "FGFR positive, but\nnot full study screened"),
  color = "black", size = 3,check_overlap = TRUE) +
geom_tile(mapping=aes(x = 8, y = 51.5),fill = "yellow",height=7,width=1.5,color = "black") +
geom_text(mapping=aes(x = 8, y = 51.5, label = "FGFR positive Patients who\ndropped before full screening"),
  color = "black", size = 3,check_overlap = TRUE) +
```

GraphVIZ– diagrammeR

Pros and Cons

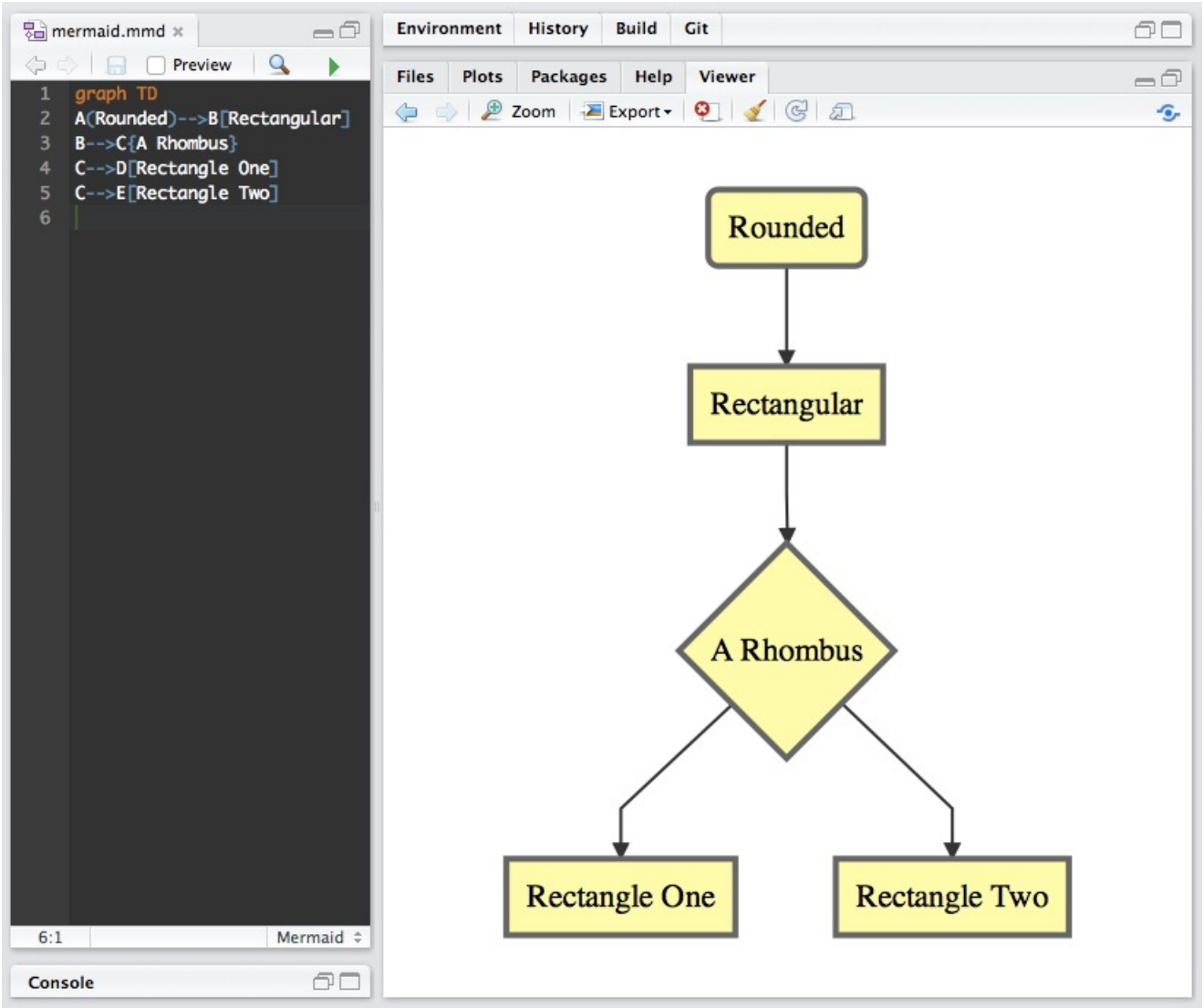
Pros

- Short, straightforward code
- Numerous network graph possibilities
 - Mermaid plots, DiGraphs, Bidirectional Graphs, etc.
- Dynamic rendering
 - Package determines best way to layout nodes

Cons

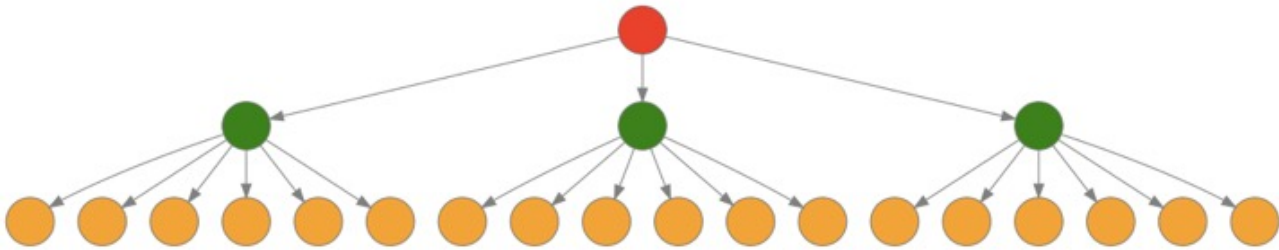
- Some difficulty in RShiny and Rmarkdown
- Lack of control on specific node placement
 - Just set size, shape, hierarchy

diagrammeR Examples

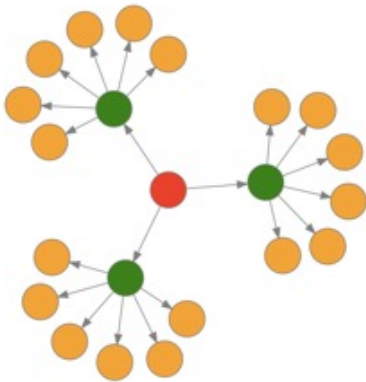


Graphviz Engines

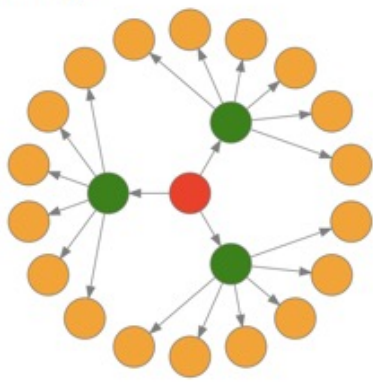
dot



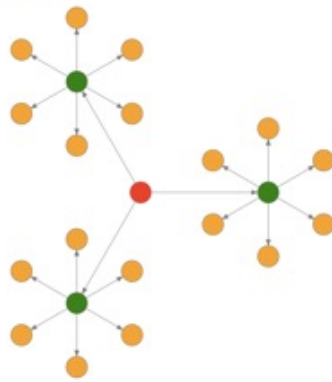
neato



twopi



circo



Sankey Diagram – Plotly

Pros and Cons

Pros

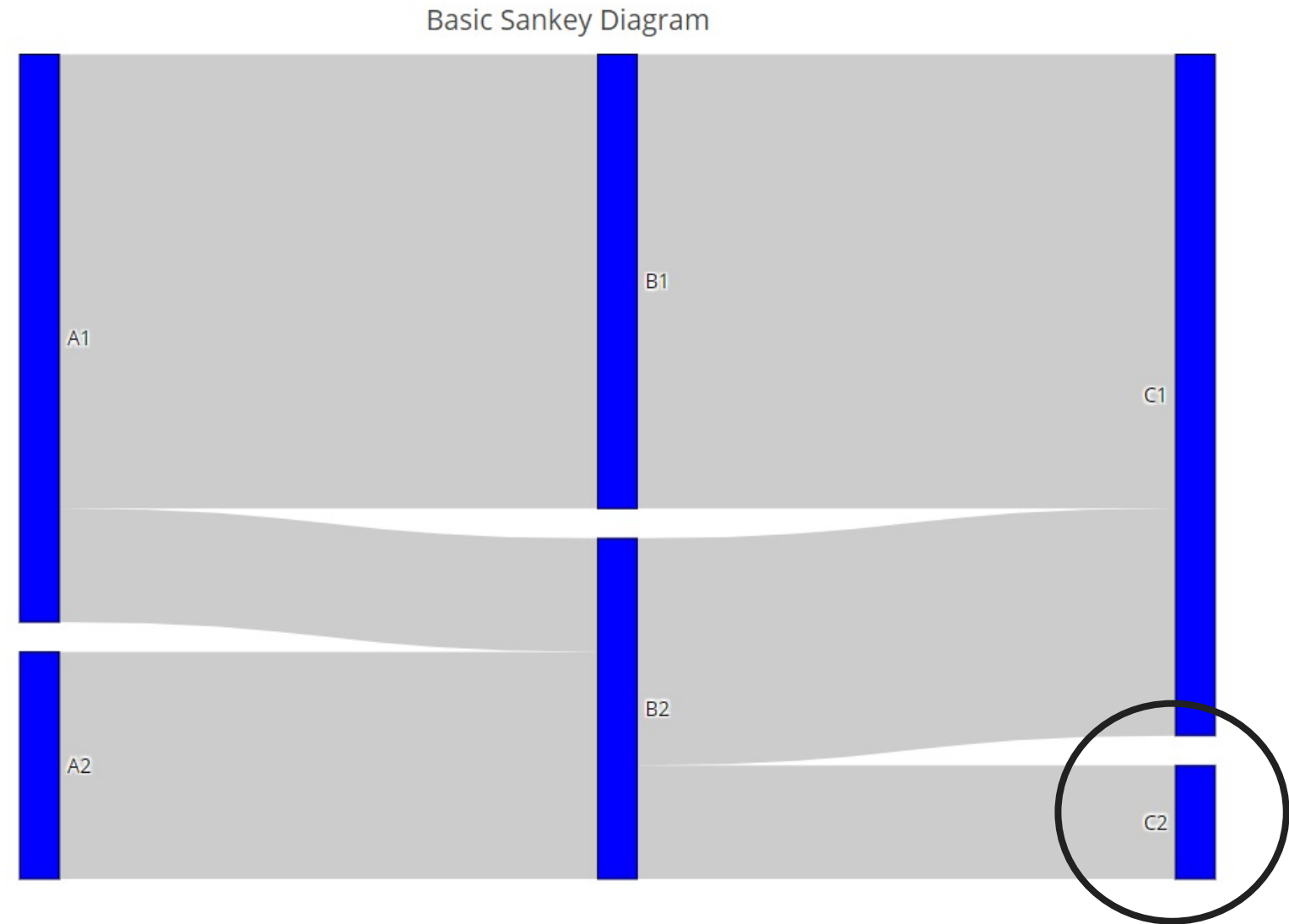
- Easy to code
 - Easy to change in future
- Plotly is feature-rich
 - Zoom, Download, Hover-Info
- Extremely Customizable
- Intuitive – Magnitude of node represents size of population

Cons

- Not a “true” Sankey Diagram
- More similar to “Alluvial Diagram”, all terminal nodes are considered endpoints
 - Can be a pro or con, depending on potential use

Plotly Sankey Diagram Example

```
fig <- plot_ly(  
  type = "sankey",  
  orientation = "h",  
  
  node = list(  
    label = c("A1", "A2", "B1", "B2", "C1", "C2"),  
    color = c("blue", "blue", "blue", "blue", "blue", "blue"),  
    pad = 15,  
    thickness = 20,  
    line = list(  
      color = "black",  
      width = 0.5  
    )  
  ),  
  
  link = list(  
    source = c(0,1,0,2,3,3),  
    target = c(2,3,3,4,4,5),  
    value = c(8,4,2,8,4,2)  
  )  
)  
fig <- fig %>% layout(  
  title = "Basic Sankey Diagram",  
  font = list(  
    size = 10  
  )  
)
```



Sankey Diagram – networkD3

Pros and Cons

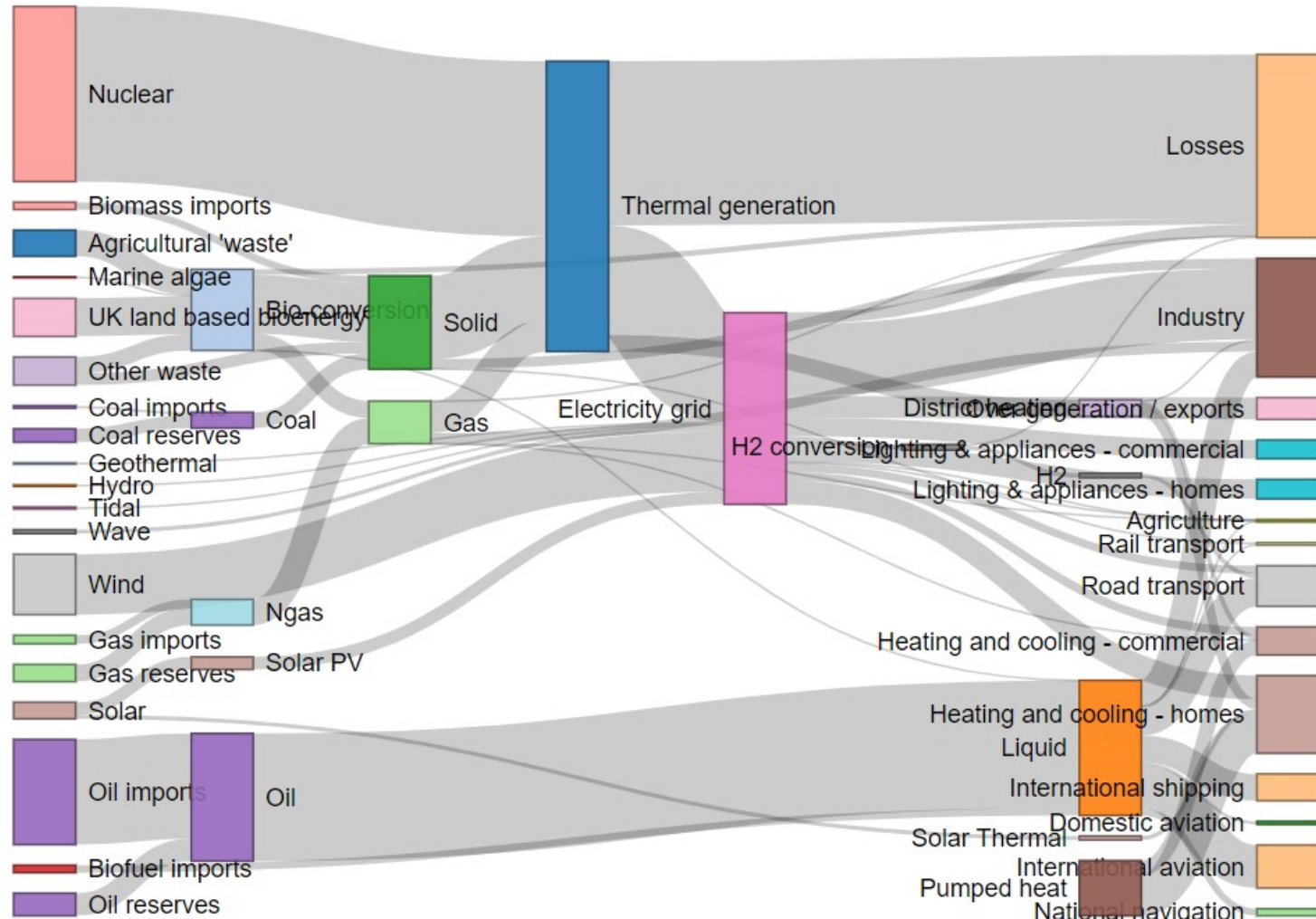
Pros

- Easy to code
 - Easy to change in future
- networkD3 designed to create these graphs
- Numerous graph types
 - Simple & Forced Networks, Sankey, Radial, Dendro
- Built in Javascript, renders to HTML

Cons

- Not as feature-rich as Plotly's graphing package
 - Can't download, hover, zoom without additional programming

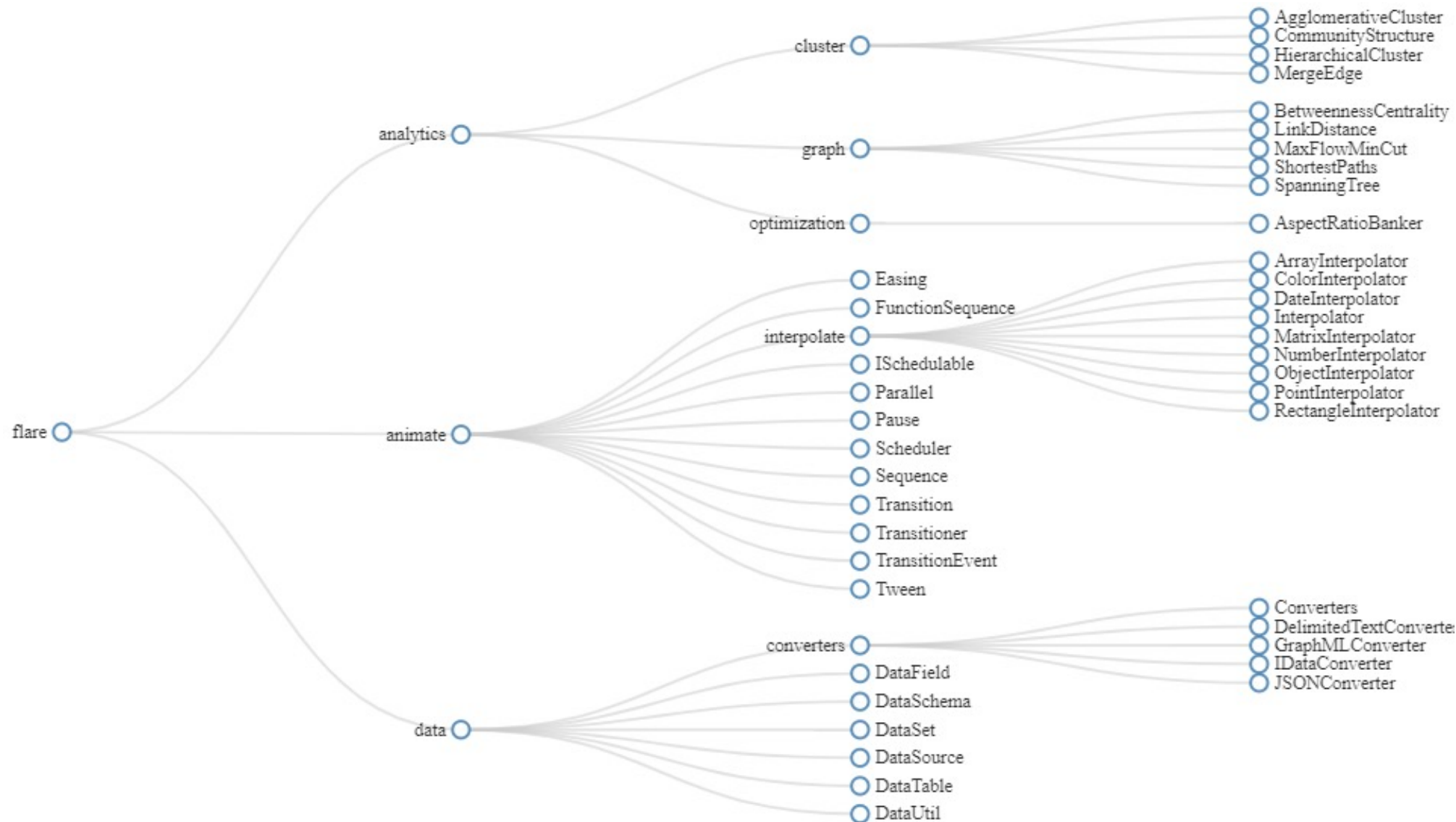
networkD3 Examples



```
# Load energy projection data
# Load energy projection data
URL <- paste0(
  "https://cdn.rawgit.com/christophergandrud/networkD3/",
  "master/JSONdata/energy.json")
Energy <- jsonlite::fromJSON(URL)
# Plot
sankeyNetwork(Links = Energy$links, Nodes = Energy$nodes, Source = "source",
  Target = "target", Value = "value", NodeID = "name",
  units = "TWh", fontSize = 12, nodeWidth = 30)
```

networkD3 Examples - Continued

```
diagonalNetwork(List = Flare, fontSize = 10, opacity = 0.9)
```



Learnings

- Some packages are more modern, have more capabilities for future RShiny interaction
- Trade-off between precise output, development time, available features
 - Envision endproduct to guide package choice
- Use multiple packages when appropriate
- Explore integrating and downloading in Rmarkdown document for enhanced report generation

Example RShiny Application

- Uses excel input of summary statistics
 - Convert to data frame using **readxl** package
- Uses networkD3 and ggplot2 text/tile where appropriate
- Download graphs via RShiny downloadHandler and Rmarkdown
 - Download graphs directly as .png files or compile all graphs in .ppt using RMD
- Most important step when working with multiple teams is standardizing input
 - Data manipulation was the most time consuming step
 - Graph generation is seamless and quick, once data is formatted properly
 - Read directly from datasets (STDM) to calculate statistics directly in app, rather than reading from summary statistic datasets



PHARMACEUTICAL COMPANIES OF *Johnson & Johnson*